Probabilistic Graphical Models: Distributed Inference and Learning Models with Small Feedback Vertex Sets

by

Ying Liu

B.E., Electronic Engineering, Tsinghua University, 2008
S.M., Electrical Engineering and Computer Science, MIT, 2010

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology
June 2014

Signature of Author: Signature redacted

Department of Electrical Engineering and Computer Science
May 21, 2014

Certified by: Signature redacted

Alan S. Willsky
Edwin Sibley Webster Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: Signature redacted

Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Probabilistic Graphical Models: Distributed Inference and Learning Models with Small Feedback Vertex Sets

by Ying Liu

Submitted to the Department of Electrical Engineering
and Computer Science on May 21, 2014
in Partial Fulfillment of the Requirements for the Degree
of Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

In undirected graphical models, each node represents a random variable while the set of edges specifies the conditional independencies of the underlying distribution. When the random variables are jointly Gaussian, the models are called Gaussian graphical models (GGMs) or Gauss Markov random fields. In this thesis, we address several important problems in the study of GGMs.

The first problem is to perform inference or sampling when the graph structure and model parameters are given. For inference in graphs with cycles, loopy belief propagation (LBP) is a purely distributed algorithm, but it gives inaccurate variance estimates in general and often diverges or has slow convergence. Previously, the hybrid feedback message passing (FMP) algorithm was developed to enhance the convergence and accuracy, where a special protocol is used among the nodes in a pseudo-FVS (an FVS, or feedback vertex set, is a set of nodes whose removal breaks all cycles) while standard LBP is run on the subgraph excluding the pseudo-FVS. In this thesis, we develop recursive FMP, a purely distributed extension of FMP where all nodes use the same integrated message-passing protocol. In addition, we introduce the subgraph perturbation sampling algorithm, which makes use of any pre-existing tractable inference algorithm for a subgraph by perturbing this algorithm so as to yield asymptotically exact samples for the intended distribution. We study the stationary version where a single fixed subgraph is used in all iterations, as well as the non-stationary version where tractable

subgraphs are adaptively selected.

The second problem is to perform model learning, i.e. to recover the underlying structure and model parameters from observations when the model is unknown. Families of graphical models that have both large modeling capacity and efficient inference algorithms are extremely useful. With the development of new inference algorithms for many new applications, it is important to study the families of models that are most suitable for these inference algorithms while having strong expressive power in the new applications. In particular, we study the family of GGMs with small FVSs and propose structure learning algorithms for two cases: 1) All nodes are observed, which is useful in modeling social or flight networks where the FVS nodes often correspond to a small number of high-degree nodes, or hubs, while the rest of the networks is modeled by a tree. 2) The FVS nodes are latent variables, where structure learning is equivalent to decomposing an inverse covariance matrix (exactly or approximately) into the sum of a tree-structured matrix and a low-rank matrix. We perform experiments using synthetic data as well as real data of flight delays to demonstrate the modeling capacity with FVSs of various sizes.

Thesis Supervisor: Alan S. Willsky

Title: Edwin Sibley Webster Professor of Electrical Engineering and Computer Science

Acknowledgments

This has been an amazing six-year intellectual journey that would be impossible without the help of many wonderful people.

First and foremost, I am extremely fortunate to have Prof. Alan Willsky as my thesis supervisor. Since our very first grouplet meeting, I have never stopped being amazed by his incredibly deep knowledge and his ability to quickly grasp both high-level ideas and technical details. Alan has allowed me to freely pursue my research ideas while giving me invaluable guidance. His tremendous intellectual enthusiasm and remarkable energy has always been an inspiration to me. Without his support and help, none of my PhD studies would be possible. Thanks, Alan, for everything.

I am very grateful to my thesis committee members Prof. Devavrat Shah and Prof. Yury Polyanskiy. I thank them for their encouragement and for many helpful discussions. I benefited greatly both from talking with them about my research and from reading their research work on my own. Their knowledge in a variety of fields has provided me new perspectives and shaped how I view my research on a broader level.

I thank Prof. Bill Freeman who served on my RQE committee with Devavrat. At this early stage of my research, they gave me advice on presentation skills and provided insights on how to formulate research problems. I also thank Prof. Alan Oppenheim, who has been my academic advisor at MIT and guided me through every step during my study.

I am fortunate to have been a teaching assistant with Prof. Polina Golland and Prof. Greg Wornell, from whom I learned how to explain difficult concepts intuitively and clearly.

In addition to Alan, I enjoyed collaborating with Anima Anandkumar, Venkat Chandrasekaran, and Oliver Kosut. In particular, I would like to thank Venkat for helping me jump-start my research journey by giving me frequent feedback on

my half-baked ideas.

I am grateful to Rachel Cohen, Jennifer Donovan, Janet Fischer, and Brian Jones, who assisted me to ensure my progress was smooth.

I have interacted with many other great people in LIDS, RLE, and CSAIL who have helped me in many aspects both in research and life. I thank Jason Chang, George Chen, Myung Jin Choi, Justin Dauwels, Rose Faghih, Audrey Fan, Emily Fox, Roger Grosse, Qing He, Ying-zong Huang, Matt Johnson, Na Li, Dahua Lin, Dmitry Malioutov, Sidhant Misra, James Saunderson, Parikshit Shah, Ramesh Sridharan, John Sun, Vincent Tan, Kush Varshney, Lav Varshney, Ermin Wei, Yehua Wei, Kuang Xu, Ying Yin, Lei Zhang, Yuan Zhong, and Hongchao Zhou.

It is impossible to enumerate all my friends who have made my life at MIT full of excitement and fun. I thank them all the same.

Finally, I thank my family for their unreserved love and support. This thesis would have certainly been impossible to complete without them. They are the source of my determination and perseverance in pursuing all my endeavors.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In undirected graphical models or Markov random fields (MRFs), each node represents a random variable while the set of edges specifies the conditional independencies of the underlying distribution. When the random variables are jointly Gaussian, the models are called *Gaussian graphical models* (GGMs) or *Gauss Markov random fields* (GMRFs). GGMs, such as linear state space models, Bayesian linear regression models, and thin-membrane/thin-plate models, have been widely used in communication, image processing, medical diagnostics, oceanography, and gene regulatory networks [1, 2, 3, 4].

There are two fundamental problems in the study of GGMs. The first problem is to perform inference or sampling when the graph structure and model parameters are given. Inference refers to computing the marginal distributions or the most likely state, while sampling refers to drawing samples from the underlying probabilistic distribution. In some contexts, sampling is considered as a type of inference as the generated samples are often used to approximately compute some inference results when direct inference is prohibitively costly. In the era of big data, a central challenge in many applications of machine learning is how to efficiently process the gigantic amount of data available and make near real-time estimation and prediction. In the modern computational infrastructure (such as cloud computing), distributed and parallel algorithms are of great importance, and they significantly outperform many traditional algorithms developed for the traditional single-machine framework. The second problem is to perform model learning, i.e., to recover the underlying structure and model parameters from observations when the model is unknown. Families of graphical models that have both large modeling capacity and efficient inference algorithms are extremely useful. With the development of new inference algorithms for many new applications, it is important to study the family of models that are most suitable for these inference algorithms while having strong expressive power in the new applications.

In this thesis, we propose (1) the recursive feedback message passing algorithm, which is a purely distributed message-passing algorithm for inference; (2) a sampling framework based on perturbing models on subgraphs; and (3) learning algorithms for several different cases in learning the family of models with small feedback vertex sets. We motivate our algorithms and provide a brief literature review in Sections 1.1–1.3. Next, in Section 1.4, we outline the thesis organization and give a overview of the contributions.

1.1 Recursive Feedback Message Passing for Distributed Inference

For GGMs of moderate size, exact inference can be solved by algorithms such as direct matrix inversion, Cholesky factorization, and nested dissection, but these algorithms cannot be used for large-scale problems due to the computational complexity [4, 5].

For tree-structured graphs, a message-passing algorithm called belief propagation (BP) can give exact results in linear time. When there are cycles in the graphs, loopy belief propagation (LBP) is often used, where the message-update protocol is the same as BP. LBP is distributed in nature: messages from all nodes may be updated in parallel using only local information. However, LBP is not guaranteed to converge or give accurate results [6, 7, 8, 9]. Some extensions to LBP include generalized belief propagation [10], tree-reweighted message passing [11], double-loop belief propagation [12], and relaxed Gaussian belief propagation [13]. LBP in the context of quadratic minimization has also been studied in [14, 15]. For inference in Gaussian graphical models with cycles, LBP performs well for some graphs, but often diverges or has slow convergence. When LBP does converge, the variance estimates are incorrect in general.

In [16] the authors have proposed the feedback message passing (FMP) algorithm. FMP uses a different protocol among a special set of vertices called a *feedback vertex set* or FVS, a set of nodes whose removal breaks all cycles in the graph. When the size of the FVS is large, a pseudo-FVS is used instead of an FVS. By performing two rounds of standard LBP among the non-feedback nodes and solving a small inference problem among the feedback nodes, FMP improves the convergence and accuracy significantly compared with running LBP on the entire graph. In addition, choosing the size of the pseudo-FVS enables us to make the trade-off between efficiency and accuracy explicit. FMP is partially distributed, but the algorithm in [16] still requires centralized communication among the feedback nodes. One can ask some natural questions: Is it possible

to select the feedback nodes in a purely distributed manner? Can we further eliminate the centralized computations among the feedback nodes in FMP without losing the improvements on convergence and accuracy?

In Chapter 3, we propose *recursive FMP*, a recursive and purely distributed extension of FMP where all nodes use the same message-passing protocol. In recursive FMP, an inference problem on the entire graph is recursively reduced to smaller subgraphs until inference can be solved efficiently by an exact or approximate message-passing algorithm. A purely distributed algorithm is of great importance because in many scenarios, such as wireless sensor networks, it is easy to implement the same protocol on all nodes while centralized computations are often expensive or impractical. In this recursive approach, there is only one active feedback node at a time, and thus centralized communication among feedback nodes in FMP is reduced to message forwarding from the single feedback node. While under certain conditions, essentially those that are identical to those required for the original FMP algorithm, our recursive algorithm produces the same results but in a distributed manner. However, our distributed algorithm is far more flexible, as the feedback nodes used by different parts of a very large graph may be different, allowing each node in the graph to adapt and respond to those nodes of most importance locally.

1.2 Sampling Gaussian Graphical Models Using Subgraph Perturbations

As a fundamental problem by itself, sampling also has the relative advantage of allowing estimation of arbitrary statistics from the random field, rather than only the mean and variance. Moreover, sampling is useful for statistical models in which a GGM is one of several interacting components. In such a setting, a sampler for the GGM is an essential piece of any Markov chain Monte-Carlo (MCMC) framework for the entire system. Efficient sampling algorithms have been used to solve inference problems [17], to estimate model parameters [18], and used for model determination [19].

Very efficient algorithms for both inference and sampling exist for GGMs in which the underlying graph is a tree (i.e., it has no cycles). Such models include hierarchical hidden Markov models [20], linear state space models [21], and multi-scale auto-regressive models [22]. For these models exact inference can be computed in linear time using BP [23] (which generalizes the Kalman filter and the Rauch-Tung-Striebel smoother [21]), and exact samples can be generated using the forward sampling method [23]. However,

the modeling capacity of trees is limited. Graphs with cycles can more accurately model real-world phenomena, but exact sampling is often prohibitively costly for large-scale models with cycles.

MCMC samplers for general probabilistic models have been widely studied and can generally be applied directly to GGMs. The most straightforward is the Gibbs sampler, wherein a new sample for each variable is generated by conditioning on the most recent sample of its neighbors [24]. However, the Gibbs sampler can have extremely slow convergence even for trees, making it impractical in large networks. For this reason, many techniques, such as reordering [25], blocking [26, 27], or collapsing [28], have been proposed to improve Gibbs sampling. In particular, the authors of [29] have proposed a blocked Gibbs sampler where each block includes a set of nodes whose induced subgraph does not have cycles; in [17] a Metropolis-Hastings sampler is studied, where a set of "control variables" are adaptively selected.

There are also sampling algorithms for GGMs that make explicit use of the joint Gaussianity. Since inference in a GGM is equivalent to solving a linear system, sampling algorithms are often closely related to direct or iterative linear solvers. One approach is using the Cholesky decomposition to generate exact samples. If a sparse Cholesky decomposition is provided directly from the problem formulation, then generating samples using that decomposition is the preferred approach. Similarly, in [30] the problem formulation leads directly to a decomposition into sparse "filters", which are then used, together with random perturbations to solve linear equations that produce samples. Once again, for problems falling into this class, using this method is unquestionably preferred. However, for other Gaussian models for which such sparse decompositions are not directly available, other approaches need to be considered. In particular, the computation of the Cholesky decomposition has cubic complexity and a quadratic number of fills in general, even for sparse matrices as arise in graphical models [31]. While this complexity is acceptable for models of moderate size, it can be prohibitively costly for large models, e.g., those involving millions or even billions of variables.

In Chapter 4, we propose a general framework to convert iterative linear solvers based on graphical splittings to MCMC samplers by adding a random perturbation at each iteration. In particular, our algorithm can be thought of as a stochastic version of graph-based solvers and, in fact, is motivated by the use of embedded trees in [32, 33] for the computation of the mean of a GGM. That approach corresponds to decomposing the underlying graph of the model into a tractable graph, i.e., one for which sampling

is easy (e.g., a tree), and a "cut" matrix capturing the edges removed to form the tractable subgraph. The subgraphs used can have any structure for which efficient inference algorithms exist: for example, tree-structured graphs, graphs with low tree-width, or graphs having a small FVS [16]. Much more importantly, in order to obtain a valid sampling algorithm, we must exercise some care, not needed or considered for the linear solvers in [32, 33], in constructing the graphical models corresponding to both the tractable subgraph and to the set of variables involved in the cut edges.

We give general conditions under which graph-based iterative linear solvers can be converted into samplers and we relate these conditions to the so-called P-regularity condition [34]. We then provide a simple construction that produces a splitting satisfying those conditions. Once we have such a decomposition our algorithm proceeds at each iteration by generating a sample from the model on the subgraph and then randomly perturbing it based on the model corresponding to the cut edges. That perturbation obviously must admit tractable sampling itself and also must be shaped so that the resulting samples of the overall model are asymptotically exact. Our construction ensures both of these. As was demonstrated in [32, 33], using non-stationary splittings, i.e., different graphical decompositions in successive iterations, can lead to substantial gains in convergence speed. We extend our subgraph perturbation algorithm from stationary graphical splittings to non-stationary graphical splittings and give theoretical results for convergence guarantees. We propose an algorithm to select tractable subgraphs for stationary splittings and an adaptive method for selecting non-stationary splittings.

1.3 Learning Gaussian Graphical Models with Small Feedback Vertex Sets

The trade-off between the modeling capacity and the efficiency of learning and inference has been an important research problem in the study of GGMs. In general, a larger family of graphs represents a larger collection of distributions and thus can better approximate arbitrary empirical distributions. However, many graphs lead to computationally expensive inference and learning algorithms. Hence, it is important to study the trade-off between modeling capacity and efficiency.

Both inference and learning are efficient for tree-structured graphs (graphs without cycles): inference can be computed exactly in linear time (with respect to the size of the graph) using BP [35] while the learning problem can be solved exactly in quadratic time using the Chow-Liu algorithm [36]. Since trees have limited modeling capacity, many

families of models beyond trees have been proposed [37, 38, 39, 40]. Thin junction trees (graphs with low tree-width) are extensions of trees, where inference can be solved efficiently using the junction algorithm [23]. However, learning junction trees with tree-width greater than one is NP-complete [40] and tractable learning algorithms (e.g., [41]) often have constraints on both the tree-width and the maximum degree. Since graphs with large-degree nodes are important in modeling applications such as social networks, flight networks, and robotic localization, we are interested in finding a family of models that allow arbitrarily large degrees while being tractable for learning.

Beyond thin-junction trees, the family of sparse GGMs is also widely studied [42, 43]. These models are often estimated using methods such as graphical lasso (or l-1 regularization) [44, 45]. However, a sparse GGM (e.g., a grid) does not automatically lead to efficient algorithms for exact inference. Hence, we are interested in finding a family of models that are not only sparse but also have guaranteed efficient inference algorithms.

In the context of classification, the authors of [46] have proposed the tree augmented naive Bayesian model, where the class label variable itself can be viewed as a size-one observed FVS; however, this model does not naturally extend to include a larger FVS. In [47], a convex optimization framework is proposed to learn GGMs with latent variables, where conditioned on a small number of latent variables, the remaining nodes induce a sparse graph. In our setting with latent FVSs, we further require the sparse subgraph to have tree structure.

In Chapter 5, we study the family of GGMs with small FVSs. In [16] the authors have presented results showing that for models with larger FVSs, approximate inference (obtained by replacing a full FVS by a pseudo-FVS) can work very well, with empirical evidence indicating that a pseudo-FVS of size $\mathcal{O}(\log n)$ gives excellent results. We will provide some additional analysis of inference for such models (including the computation of the partition function), but the main focus is maximum likelihood (ML) learning of models with FVSs of modest size, including identifying the nodes to include in the FVS. In particular, we present several learning algorithms for different cases. For the case where all of the variables are observed, we provide an efficient algorithm for exact ML estimation, as well as an approximate and much faster greedy algorithm for this case when the FVS is unknown and large. For a second case where the FVS nodes are taken to be latent variables, we propose an alternating *low-rank* projection algorithm for model learning and show the equivalence between the structure learning problem

and the decomposition of an inverse covariance matrix into the sum of a tree-structured matrix and a low-rank matrix.

1.4 Thesis Organization and Overview of Contributions

1.4.1 Chapter 2: Background

In this background chapter, we provide necessary background for the subsequent chapters including the definitions, existing inference algorithms, common sampling algorithms, as well as some learning algorithms. In Section 2.1, we start with preliminaries on graphical models including basic graph theory, general graphical models, and specifically Gaussian graphical models. Next in Section 2.2, we describe inference algorithms for graphical models, including loopy belief propagation and the feedback message passing algorithm. We then summarize some common sampling algorithms such as using the Cholesky decomposition, forward sampling, basic Gibbs sampling, and variants of Gibbs sampling in Section 2.3. Finally in Section 2.4, we introduce preliminaries of the learning problem, including information quantities, the maximum likelihood criterion and the Chow-Liu algorithm.

1.4.2 Chapter 3: Recursive Feedback Message Passing for Distributed Inference

The primary contributions of this chapter include: (1) We propose *recursive FMP*, a purely distributed extension of FMP, where all nodes use the same message-passing protocol. An inference problem on the entire graph is recursively reduced to those on smaller subgraphs in a distributed manner. (2) We show that one advantage of this recursive approach compared with FMP is that centralized communication among feedback nodes can be turned into distributed message forwarding. (3) We characterize this algorithm using walk-sum analysis and provide theoretical results for convergence and accuracy. (4) We also demonstrate the performance using both simulated models on grids and large-scale sea surface height anomaly data.

This chapter is organized as follows. After motivating the problem in Section 3.1, we describe the recursive FMP algorithm in three separate stages in Section 3.2. Then in Section 3.3, we summarize the recursive FMP algorithm as a single integrated protocol without the separation of stages. Next we present and prove our theoretical results

using walk-sum analysis in Section 3.4. Finally in Section 3.5, we demonstrate the performance of the algorithm using simulated models on grids as well as real data for estimating sea surface height anomaly.

1.4.3 Chapter 4: Sampling Gaussian Graphical Models Using Subgraph Perturbations

The primary contributions of this chapter include: (1) We provide a general framework for converting subgraph-based iterative solvers to samplers with convergence guarantees. In addition, we provide a construction where the injected noise at each iteration can be generated simply using a set of *i.i.d.* scalar Gaussian random variables. (2) We extend our perturbation sampling algorithm from stationary graphical splittings to non-stationary graphical splittings. In the previous studies on linear solvers, it has been observed that using multiple subgraphs may give much better convergence than using any of the individual subgraphs. We prove that if we choose from a finite collection of P-regular graphical splittings, then the convergence is always guaranteed. (3) We study the use of different kinds of tractable subgraphs and we also propose an algorithm to adaptively select the subgraphs based on an auxiliary inference problem.

This chapter is organized as follows. In Section 4.2, we propose the subgraph perturbation algorithm with stationary splittings, providing efficient implementation as well as theoretical results on the convergence rate. Next in Section 4.3, we present the use of non-stationary splittings and theoretical results on convergence. We then discuss how to select tractable subgraphs for both the stationary and the non-stationary settings in Section 4.4. Finally in Section 4.5, we present experimental results using simulated data on various graph structures as well as using large-scale real data.

1.4.4 Chapter 5: Learning Gaussian Graphical Models with Small Feedback Vertex Sets

The primary contributions of this chapter include: (1) We investigate the case where all of the variables, including any to be included in the FVS are observed. We provide an algorithm for exact ML estimation that, regardless of the maximum degree, has complexity $\mathcal{O}(kn^2 + n^2 \log n)$ if the FVS nodes are identified in advance and polynomial complexity if the FVS is to be learned and of bounded size. Moreover, we provide an

approximate and much faster greedy algorithm when the FVS is unknown *and* large. (2) We study a second case where the FVS nodes are taken to be latent variables. In this case, the structure learning problem corresponds to the (exact or approximate) decomposition of an inverse covariance matrix into the sum of a tree-structured matrix and a low-rank matrix. We propose an algorithm that iterates between two projections, which can also be interpreted as alternating *low-rank* corrections. We prove that even though the second projection is onto a highly non-convex set, it is carried out exactly, thanks to the properties of GGMs of this family. By carefully incorporating efficient inference into the learning steps, we can further reduce the complexity to $\mathcal{O}(kn^2 + n^2 \log n)$ per iteration. (3) We also perform experiments using both synthetic data and real data of flight delays to demonstrate the modeling capacity with FVSs of various sizes. We show that empirically the family of GGMs with FVSs of size $\mathcal{O}(\log n)$ strikes a good balance between the modeling capacity and efficiency.

This chapter is organized as follows. In Section 5.3, we study the case where nodes in the FVS are observed. We propose the *conditioned Chow-Liu* algorithm for structure learning and prove its correctness and complexity. Next, we study the case where the FVS nodes are latent variables and propose an alternating low-rank correction algorithm for structure learning in Section 5.4. We then present experimental results for learning GGMs with small FVSs, observed or latent, using both synthetic data and real data of flight delays in Section 5.5.

1.4.5 Chapter 6: Conclusion

In this chapter, we highlight the important contributions of this thesis and discuss future research directions.

Chapter 2

Background

In this chapter, we give a brief introduction to graphical models including the definitions, existing inference algorithms, common sampling algorithms, as well as well as some learning algorithms. We outline this chapter as follows. In Section 2.1 we start with preliminaries on graphical models including basic graph theory, general graphical models, and specifically Gaussian graphical models. Next in Section 2.2, we describe inference algorithms for graphical models, including loopy belief propagation and the feedback message passing algorithm. We then summarize some common sampling algorithms such as using the Cholesky decomposition, forward sampling, basic Gibbs sampling, and variants of Gibbs sampling in Section 2.3. Finally in Section 2.4, we introduce preliminaries of the learning problem, including information quantities, the maximum likelihood criterion and the Chow-Liu algorithm.

2.1 Graphical Models

Graphical models are widely used to represent the structures of multivariate distributions using graphs [23]. The graphs used can be undirected graphs, directed graphs, or factor graphs resulting in *undirected graphical models* (or *Markov random fields*), directed graphical models (or Bayesian networks) and factor graph models. In this thesis, we focus on undirected graphical models where the underlying undirected graphs are used to model the conditional independencies in the distributions. In the following, we first briefly review basic notions from graph theory; next we introduce graphical models in a general setting; and then we describe Gaussian graphical models, the main models used in our subsequent chapters.

2.1.1 Notions in Graph Theory

A *graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of *nodes* or *vertices* $\mathcal{V}$ and a set of *edges* $\mathcal{E}$. An edge (i, j) is a pair of distinct nodes (i, j) with $i,\ j \in \mathcal{V}$. In undirected graphs, the edges are unordered pairs, i.e., (i, j) and (j, i) denote the same edge. The *neighborhood* (also called the set of *neighbors*) of a node i is the set $\mathcal{N}(i) = \{j | (i, j) \in \mathcal{E}\}$. Two nodes are *connected* if they are neighbors. The *degree* of node i, denoted as $\deg(i)$, is the number of its neighbors, which equals $|\mathcal{N}(i)|$.[1] In this thesis, we also refer to the size of $\mathcal{V}$, or $|\mathcal{V}|$, as *the size of the graph.*

A graph is called a *complete* or *fully connected* graph if any two nodes are connected. A *walk* $w = (w_0, w_1, \ldots, w_n)$ or $w = (w_0, w_1, w_2, \ldots)$ on a graph is a finite or infinite sequence of nodes where the consecutive nodes are neighbors. The *length of a walk* is the number of nodes in its sequence minus one, i.e, the length of $w = (w_0, w_1, \ldots, w_n)$ is n.[2] A *path* is a walk where all nodes in the sequence are distinct. A graph is called a *connected graph* if there exists a path between any pair of nodes. A *cycle* or *loop* is a walk that starts and ends at the same node but all other nodes are distinct. The set of The *distance* between two nodes (i, j) in a graph, denoted as $d(i, j)$, is the minimum length of all paths between i and j. The *diameter* of a graph is the maximum distance between any pair of nodes in the graph.

A *chain* is a connected graph where two nodes have degree one and all other nodes have degree two. A *forest* is a graph without cycles. If a forest is a connected graph, it is also called a *tree.* In this thesis, we use the term *tree-structured graphs* to refer to forests in general.

A graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is a *subgraph* of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ if $\mathcal{V}' \subset \mathcal{V}$ and $\mathcal{E}' \subset \mathcal{E}$. $\mathcal{G}'$ is a *spanning subgraph* of $\mathcal{G}$ if $\mathcal{V}' = \mathcal{V}$ and $\mathcal{E}' \subset \mathcal{E}$. The graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is a subgraph of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *induced* by $\mathcal{V}'$ if $\mathcal{V}' \subset \mathcal{V}$ and that $(i, j) \in \mathcal{E}'$ if and only if $i, j \in \mathcal{V}'$ and $(i, j) \in \mathcal{E}$. A subgraph is called a *clique* if it is a fully connected. A *maximal clique* is a clique that is not a proper subgraph of any larger clique. A graph is called *chordal* if every cycle of length at least four contains two nodes that are not adjacent in the cycle but are connected in the graph. The *treewidth* of a chordal graph is the size of its largest clique minus one. The treewidth of a non-chordal graph is minimum tree-width of all chordal graphs of which the non-chordal graph is a subgraph. We say that set S *separates* set A and set B if any path between a node in A and a node in B contains at

[1] We use $|A|$ to denote the cardinality of a set A.

[2] In the special case of a walk with a single node, the length is zero.

least one node in S.

2.1.2 Graphical Models and Exponential Families

Markov random fields (MRFs) are graphical models in which the conditional independence structure of a set of random variables is represented by an undirected graph [48, 23]. Each node $s \in \mathcal{V}$ corresponds to a random variable x_s. For any subset $A \subset \mathcal{V}$, the random vector $\mathbf{x}_A$ corresponds to the set of random variables $\{x_s | s \in A\}$ and we will also simply write $\mathbf{x}$ for $\mathbf{x}_\mathcal{V}$. A random vector has the Markov property with respect to the graph if for any subsets A, B, $S \subset \mathcal{V}$ where S separates A and B in the graph, $\mathbf{x}_A$ and $\mathbf{x}_B$ are independent conditioned on $\mathbf{x}_S$, i.e., $\mathbf{x}_A \perp \mathbf{x}_B | \mathbf{x}_S$. Figure 2.1 provides an illustrating example of this Markov property.

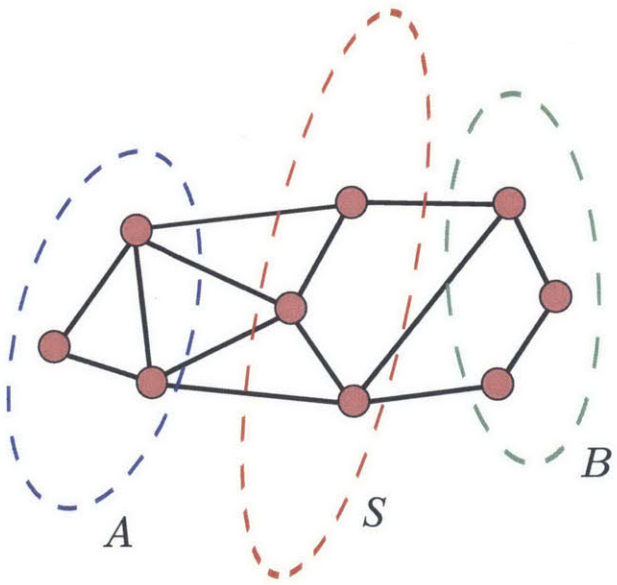


Figure 2.1: Markov property of a graphical model: $\mathbf{x}_A \perp \mathbf{x}_B || \mathbf{x}_S$ since S separates A and B.

By the Hammersley-Clifford theorem, if the probabilistic distribution function ($p.d.f.$) $p(\mathbf{x})$ of a distribution is Markov with respect to graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and is positive every-

where, then $p(\mathbf{x})$ can be factored according to

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \phi_C(\mathbf{x}_C), \tag{2.1}$$

where $\mathcal{C}$ is the collection of cliques and Z is the normalization factor or *partition function*. Each factor ϕ_C is often represented by $\phi_C(\mathbf{x}_C) = \exp\{\psi_C(\mathbf{x}_C)\}$ and thus the factorization of $p(\mathbf{x})$ can be written as

$$p(\mathbf{x}) = \frac{1}{Z} \exp\{\sum_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C)\}. \tag{2.2}$$

A graphical model is a pairwise model if the only nonzero ψ_C are for cliques of size one or two. In particular, if the underlying model is tree-structured, the p.d.f. of the distribution can be factored according to Proposition 2.1.1.

Proposition 2.1.1 : *The p.d.f. of a tree-structured model $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ can be factorized according to either of the following two equations:*

1.
$$p(\mathbf{x}) = p(\mathrm{x}_r) \prod_{i \in \mathcal{V} \setminus \{r\}} p(\mathrm{x}_i | \mathrm{x}_{\pi(i)}), \tag{2.3}$$

where r is an arbitrary node selected as the root and $\pi(i)$ is the unique parent of node i in the tree rooted at r.

2.
$$p(\mathbf{x}) = \prod_{i \in \mathcal{V}} p(\mathrm{x}_i) \prod_{(i,j) \in \mathcal{E}} \frac{p(\mathrm{x}_i, \mathrm{x}_j)}{p(\mathrm{x}_i) p(\mathrm{x}_j)}. \tag{2.4}$$

2.1.3 Gaussian Graphical Models

An important sub-class of MRFs are *Gaussian Markov random fields* (GMRFs) or *Gaussian graphical models* (GGMs), where the joint distribution is Gaussian. GGMs have been widely used in computer vision [2], computational biology [49], medical diagnostics [50], and communication systems [51]. GGMs are particularly important in very large probabilistic networks involving millions of variables [4, 5].

Using the representation of (2.2), the p.d.f. of a GGM can be written as

$$p(\mathbf{x}) \propto \exp\{\sum_{i\in\mathcal{V}} \psi_i(x_i) + \sum_{(i,j)\in\mathcal{E}} \psi_{ij}(\mathbf{x}_{ij})\}, \tag{2.5}$$

where

$$\psi_i(x_i) = -\frac{1}{2}J_{ii}x_i^2 + h_i x_i \tag{2.6}$$

$$\psi_{ij}(\mathbf{x}_{ij}) = -J_{ij}x_i x_j. \tag{2.7}$$

Hence, the p.d.f. of the distribution can be parametrized by

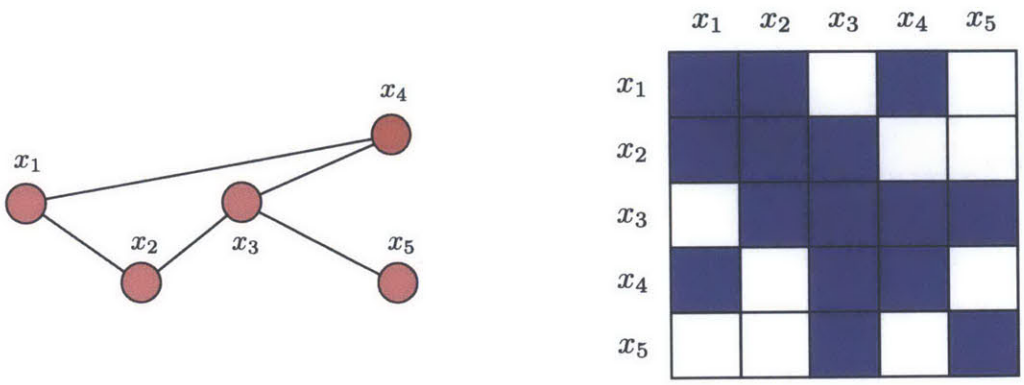
$$p(\mathbf{x}) \propto \exp\{-\frac{1}{2}\mathbf{x}^T J\mathbf{x} + \mathbf{h}^T\mathbf{x}\}, \tag{2.8}$$

where J is the *information matrix* or *precision matrix* and $\mathbf{h}$ is the *potential vector*. For a valid Gaussian graphical model,the information matrix J is positive definite. The parameters J and $\mathbf{h}$ are related to the mean $\boldsymbol{\mu}$ and covariance matrix Σ by $\boldsymbol{\mu} = J^{-1}\mathbf{h}$ and $\Sigma = J^{-1}$. We denote this distribution by either $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ or $\mathcal{N}^{-1}(\mathbf{h}, J)$.

The structure of the underlying graph can be constructed using the sparsity pattern of J, i.e., there is an edge between i and j if and only if $J_{ij} \neq 0$. Hence, the conditional independence structure can be read immediately from the sparsity pattern of the information matrix as well as that of the underlying graph (See Figure 2.2). Our starting point will simply be the specification of $\mathbf{h}$ and J (and with it the graphical structure). One setting in which such a specification arises (and which we will illustrate with our large-scale example) is in estimation problems, that in which $\mathbf{x}$ represents a large random field, which has prior distribution $\mathcal{N}^{-1}(0, J_0)$ according to a specified graph[3] (e.g., the thin-membrane or the thin-plate model [1]) and where we have potentially sparse and noisy measurements of components of $\mathbf{x}$ given by $\mathbf{y} = C\mathbf{x} + \mathbf{v}$, $\mathbf{v} \sim \mathcal{N}(0, R)$, where C is a selection matrix (a single 1 in each row, all other row elements being 0) and R is a (blocked) diagonal matrix. In this case, the posterior distribution $p(\mathbf{x}|\mathbf{y})$ is $\mathcal{N}^{-1}(\mathbf{h}, J)$, where $\mathbf{h} = C^T R^{-1}\mathbf{y}$ and $J = J_0 + C^T R^{-1} C$.

In the following chapters of this thesis, we focus on GGMs to demonstrate our inference and learning algorithms while some of the ideas can be extended to other

[3]Without loss of generality we can assume that the prior mean of $\mathbf{x}$ is 0 simply by subtracting it from the random field and from the measurements.



(a) (b)

Figure 2.2: Sparsity relationship between the underlying undirected graph and the information matrix: (a) The sparsity pattern of the undirected graph; (b) The sparsity pattern of the information matrix.

pairwise models such as the Ising models [23].

2.2 Inference Algorithms

The inference problems in graphical models refer to computing the marginal distributions of individual variables or the maximum likelihood state (i.e., the variable configuration with the highest probability density) given model parameters. In Gaussian graphical models, inference refers to computing (exactly or approximately) the means μ_i and variances Σ_{ii} for all $i \in \mathcal{V}$ given J and $\mathbf{h}$. In this section, we review the message-passing algorithm belief propagation (BP), the walk-sum analysis framework, as well as the feedback message passing (FMP) algorithm.

2.2.1 Belief Propagation

BP is an efficient message-passing algorithm that gives exact inference results in linear time for tree-structured graphs [23]. The Kalman filter for linear Gaussian estimation and the forward-backward algorithm for hidden Markov models can be viewed as special instances of BP. Though widely used, tree-structured models (also known as cycle-free

graphical models) possess limited modeling capabilities, and many stochastic processes and random fields arose in real-world applications cannot be well-modeled using cycle-free graphs. Loopy belief propagation (LBP) is an application of the message-passing protocol of BP on loopy graphs using the same local message update rules. Without loss of generality, we use BP and LBP interchangeably throughout this thesis, as the protocols are the same. Empirically, it has been observed that LBP performs reasonably well for certain graphs with cycles [7, 52]. Indeed, the decoding method employed for turbo codes has also been shown to be a successful instance of LBP [53]. A desirable property of LBP is its distributed nature—as in BP, message updates in LBP only involve local model parameters and local incoming messages, so all nodes can update their messages in parallel.

In Gaussian graphical models, the set of messages can be represented by $\{\Delta J_{i\to j} \cup \Delta h_{i\to j}\}_{(i,j)\in\mathcal{E}}$, where $\Delta J_{i\to j}$ and $\Delta h_{i\to j}$ are scalar values. Consider a Gaussian graphical model: $p(\mathbf{x}) \propto \exp\{-\frac{1}{2}\mathbf{x}^T J\mathbf{x} + \mathbf{h}^T\mathbf{x}\}$. BP (or LBP) proceeds as follows [54]:

1. Message Passing

   The messages are initialized as $\Delta J_{i\to j}^{(0)}$ and $\Delta h_{i\to j}^{(0)}$, for all $(i,j) \in \mathcal{E}$. These initializations may be chosen in different ways. In this thesis we initialize all messages with the value 0.

   At each iteration t, the messages are updated based on previous messages as

$$\Delta J_{i\to j}^{(t)} = -J_{ji}(\hat{J}_{i\backslash j}^{(t-1)})^{-1} J_{ij}, \tag{2.9}$$

$$\Delta h_{i\to j}^{(t)} = -J_{ji}(\hat{J}_{i\backslash j}^{(t-1)})^{-1} \hat{h}_{i\backslash j}^{(t-1)}, \tag{2.10}$$

   where

$$\hat{J}_{i\backslash j}^{(t-1)} = J_{ii} + \sum_{k\in\mathcal{N}(i)\backslash j} \Delta J_{k\to i}^{(t-1)}, \tag{2.11}$$

$$\hat{h}_{i\backslash j}^{(t-1)} = h_i + \sum_{k\in\mathcal{N}(i)\backslash j} \Delta h_{k\to i}^{(t-1)}. \tag{2.12}$$

   The fixed-point messages are denoted as $\Delta J_{i\to j}^{*}$ and $\Delta h_{i\to j}^{*}$ if the messages converge.

2. Computation of Means and Variances:

The variances and means are computed based on the fixed-point messages as

$$\hat{J}_i = J_{ii} + \sum_{k \in \mathcal{N}(i)} \Delta J^*_{k \to i} \tag{2.13}$$

$$\hat{h}_i = h_i + \sum_{k \in \mathcal{N}(i)} \Delta h^*_{k \to i}. \tag{2.14}$$

The variances and means can then be obtained by $\Sigma_{ii} = \hat{J}_i^{-1}$ and $\mu_i = \hat{J}_i^{-1} \hat{h}_i$.

2.2.2 Walk-sum Analysis

Computing the means and variances of a Gaussian graphical model corresponds to solving a set of linear equations and obtaining the diagonal elements of the inverse of J respectively. There are many ways in which to do this – e.g., by direct solution, or using various iterative methods. As we outline in this section, one way to interpret the exact or approximate solution of this problem is through walk-sum analysis, which is based on a simple power series expansion of J^{-1}. In [54, 33] walk-sum analysis is used to interpret the computations of means and variances formally as collecting all required "walks" in a graph. In particular, the analysis in [54] identifies that when the required walks can be summed in arbitrary orders, i.e., when the model is walk-summable, LBP converges and gives the correct mean.[4] One of the important benefits of walk-sum analysis is that it allows us to understand what various algorithms compute and relate them to the required exact computations. For example, as shown in [54], LBP collects all of the required walks for the computation of the means (and, hence, always yields the correct means if it converges) but only some of the walks required for variance computations for loopy graphs (so, if it converges, its variance calculations are not correct).

Frequently it will be convenient to assume without loss of generality that the information matrix J has been normalized such that all its diagonal elements are equal to unity. Let $R = I - J$, and note that R has zero diagonal. The matrix R is called the *edge-weight matrix*.[5]

[4] As will be formally defined later, walk-summability corresponds to the absolute convergence of the series corresponding to the walk-sums needed for variance computation in a graphical model [54].

[5] The matrix R, which has the same off-diagonal sparsity pattern as J, is a matrix of partial correlation coefficients: R_{ij} is the conditional correlation coefficient between x_i and x_j conditioned on all of the other variables in the graph.

In GGMs, the *weight of a walk* is defined as the product of the edge weights,

$$\phi(w) = \prod_{l=1}^{l(w)} R_{w_{l-1},w_l}, \tag{2.15}$$

where $l(w)$ is the length of walk w. Also, we define the weight of a zero-length walk, i.e., a single node, as one. By the Neumann power series for matrix inversion, the covariance matrix can be expressed as

$$\Sigma = J^{-1} = (I - R)^{-1} = \sum_{l=0}^{\infty} R^l. \tag{2.16}$$

This formal series converges (although not necessarily absolutely) if the spectral radius, $\rho(R)$, i.e., the magnitude of the largest eigenvalue of R, is less than 1.
Let $\mathcal{W}$ be a set of walks. We define the walk-sum of $\mathcal{W}$ as

$$\phi(\mathcal{W}) \triangleq \sum_{w \in \mathcal{W}} \phi(w). \tag{2.17}$$

We use $\phi(i \to j)$ to denote the sum of all walks from node i to node j. In particular, we call $\phi(i \to i)$ the *self-return walk-sum* of node i. It is easily checked that the (i, j) entry of R^l equals $\phi^l(i \to j)$, the sum of all walks of length l from node i to node j. Hence

$$\Sigma_{ij} = \phi(i \to j) = \sum_{l=0}^{\infty} \phi^l(i \to j). \tag{2.18}$$

A Gaussian graphical model is *walk-summable* (WS) if for all $i, j \in \mathcal{V}$, the walk-sum $\phi(i \to j)$ converges for any order of the summands in (2.18) (note that the summation in (2.18) is ordered by walk-length). In walk-summable models, $\phi(i \to j)$ is well-defined for all $i, j \in \mathcal{V}$. The covariances and the means can be expressed as

$$\Sigma_{ij} = \phi(i \to j), \tag{2.19}$$

$$\mu_i = \sum_{j \in \mathcal{V}} h_j P_{ij} = \sum_{j \in \mathcal{V}} h_j \phi(i \to j). \tag{2.20}$$

As shown in [54] for non-WS models, LBP may not converge and can, in fact, yield oscillatory variance estimates that take on negative values. Here we list some useful

results from [54] that will be used in this thesis.

Proposition 2.2.1 : *The following conditions are equivalent to walk-summability:*

(i) $\sum_{w \in \mathcal{W}_{i \to j}} |\phi(w)|$ *converges for all* $i, j \in \mathcal{V}$, *where* $\mathcal{W}_{i \to j}$ *is the set of walks from* i *to* j.

(ii) $\rho(\bar{R}) < 1$, *where* $\bar{R}$ *is the matrix whose elements are the absolute values of the corresponding elements in* R.

Proposition 2.2.2 : *A Gaussian graphical model is walk-summable if it is attractive, i.e., every edge weight* R_{ij} *is nonnegative; a valid Gaussian graphical model is walk-summable if the underlying graph is cycle-free.*

Proposition 2.2.3 : *For a walk-summable Gaussian graphical model, LBP converges and gives the correct means.*

Proposition 2.2.4 : *In walk-summable models, the estimated variance from LBP for a node is the sum over all backtracking walks*[6]*, which is a subset of all self-return walks needed for computing the correct variance.*

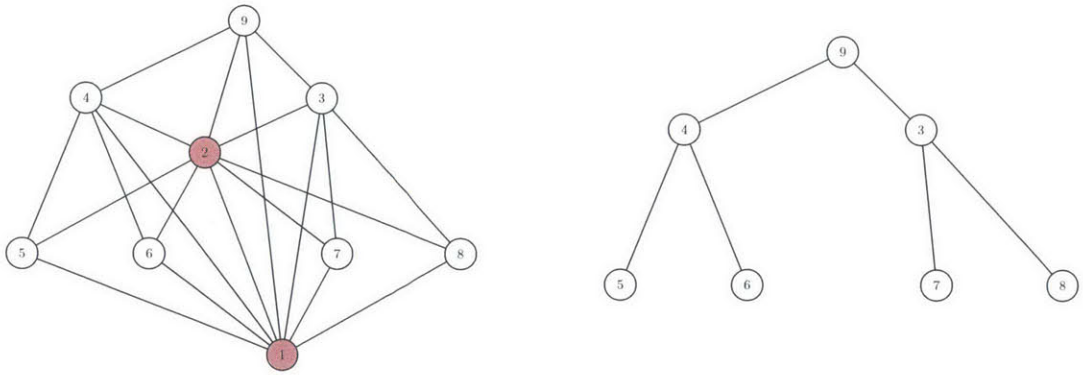
2.2.3 Feedback Message Passing

A *feedback vertex set* (FVS) is defined as a set of vertices whose removal (with the removal of the incident edges) results in an cycle-free graph [55]. An example of a graph and its FVS is given in Figure 2.3, where the full graph (Figure 2.3a) becomes a cycle-free graph (Figure 2.3b) if nodes 1 and 2 are removed, and thus the set $\{1, 2\}$ is an FVS. A pseudo-FVS is a subset of an FVS that breaks not all but most crucial cycles. Frequently we refer to an FVS as a *full* FVS to emphasize the distinction.

The FMP algorithm is a message-passing algorithm that can compute the means and variances of all nodes exactly with a computational complexity of $\mathcal{O}(k^2 n)$, where k is the size of the FVS used in the algorithm, and n is the total number of nodes. When the size of the full FVS is too large, approximate FMP can be used, where a pseudo-FVS

[6] A backtracking walk of a node is a self-return walk that can be reduced consecutively to a single node. Each reduction is to replace a subwalk of the form $\{i, j, i\}$ by the single node $\{i\}$. For example, a self-return walk of the form 12321 is backtracking, but a walk of the form 1231 is not.

is selected instead of an FVS, and where inference in the non-cycle-free graph obtained by removing the pseudo-FVS is carried out approximately using LBP. With a slight abuse of terminology, in this thesis, we use FMP to refer to both FMP and approximate FMP in [56] because the procedures are similar except for whether the feedback nodes constitute a full FVS. In the following, we use F to denote the set of feedback nodes and T to denote the set of non-feedback nodes. We also use $\mathcal{T}$ in the calligraphic font to denote the subgraph induced by the set T, where the subgraph is cycle-free when F is an FVS and has cycles when F is a pseudo-FVS. We also use the calligraphic $\mathcal{T}$ instead of T in the superscripts to avoid confusion with matrix transposition. The FMP algorithm works as follows.



(a) (b)

Figure 2.3: A graph with an FVS of size 2. (a) Full graph; (b) Tree-structured subgraph after removing nodes 1 and 2

Step 1: Before running FMP, an FVS or a pseudo-FVS is selected by a greedy algorithm to break the most crucial cycles. The selected nodes are called *feedback nodes*. After graph cleaning (i.e., the process of eliminating the tree branches[7]), the greedy algorithm computes the "priority score"

$$p_i = \sum_{j \in \mathcal{N}(i)} |J_{ij}| \tag{2.21}$$

[7]This procedure of eliminating "tree branches" simply removes nodes and edges corresponding to loop-free components of the current graph. One looks for nodes with only one neighbor, eliminating it and the edge associated with it and continues removing nodes and associated solitary edges until there are no more.

for each node i, where the definition of the scores are motivated by the theoretical results on the convergence and accuracy of FMP (c.f. [56]). Next the node with the highest score is selected as a feedback node. These steps (including graph cleaning and recomputing the priority scores) are then repeated until k feedback nodes are selected.[8] We summarize the greedy selection procedure in Algorithm 2.2.1. Note that Algorithm 2.2.1 is a centralized algorithm and the information about the selected feedback nodes is shared everywhere. After the selection, all of the priority scores are dropped and are not used again in the subsequent steps. Without loss of generality, we re-order the nodes so that the first k nodes are the selected feedback nodes and the remaining $n-k$ nodes are the non-feedback nodes. According to this ordering, the information matrix J and the potential vector $\mathbf{h}$ can be partitioned as

$$J = \begin{bmatrix} J_F & J_M' \\ J_M & J_T \end{bmatrix} \tag{2.22}$$

$$\mathbf{h} = \begin{bmatrix} \mathbf{h}_F \\ \mathbf{h}_T \end{bmatrix}. \tag{2.23}$$

Step 2: In this step, LBP is employed in the subgraph excluding the feedback nodes to compute the partial inference results with the model parameters on the subgraph as well as to compute the "feedback gains" using a set of auxiliary "mean" computations, each corresponding to a feedback node. Specifically, we construct a set of additional potential vectors $\{\mathbf{h}^1, \mathbf{h}^2, \ldots, \mathbf{h}^k\}$ with

$$\mathbf{h}^p = J_{T,p}, \ p = 1, 2, \ldots, k, \tag{2.24}$$

i.e., $\mathbf{h}^p$ is the submatrix (column vector) of J with column index p and row indices corresponding to T. Note that

$$h_i^p = J_{pi} \text{ for all } i \in \mathcal{N}(p) \tag{2.25}$$

$$h_i^p = 0 \text{ for all } i \notin \mathcal{N}(p), \tag{2.26}$$

[8]Note that the scores in (2.21) are adjusted at each iteration to reflect that nodes already in the FVS are removed from the graph (together with edges associated with them, as well as nodes and edges removed in the tree-cleanup phase) are removed from the graph used in the next stage of the selection process.

and thus $\mathbf{h}^p$ can be constructed locally with default value zero. In this step, the messages from node i to its neighbor j include $k+2$ values: $\Delta J_{i\to j}$, $\Delta h_{i\to j}$ for standard LBP and $\{\Delta h^p_{i\to j}\}_{p=1,2,\ldots,k}$ for computing the feedback gains. The standard LBP messages yield for each node i in T its "partial variance" $\Sigma^{\mathcal{T}}_{ii}$ (if the feedback nodes form a full FVS, then $\Sigma^{\mathcal{T}}_{ii} = (J_T^{-1})_{ii}$) and its "partial mean" $\mu^{\mathcal{T}}_i$ (as long as the messages converge, we have $\mu^{\mathcal{T}}_i = (J_T^{-1}\mathbf{h}_T)_i$). Note that these results are not the true variances and means since this step does not involve the contributions of the feedback nodes. At the same time, LBP using the auxiliary potential vectors $\{\mathbf{h}^1, \mathbf{h}^2, \ldots, \mathbf{h}^k\}$ yield a set of "feedback gain" $\{g^p_i\}_{p=1,2,\ldots,k}$ (similar to the mean computation, we have $g^p_i = (J_T^{-1}\mathbf{h}^p)_i$ if the messages converge). Figure 2.4a illustrates this procedure.

Step 3: After the messages in Step 2 converge, the feedback nodes collect the feedback gains from their neighbors and obtain a size-k subgraph with $\widehat{J}_{\mathcal{F}}$ and $\widehat{\mathbf{h}}_{\mathcal{F}}$ given by

$$(\widehat{J}_F)_{pq} = J_{pq} - \sum_{j\in\mathcal{N}(p)\cap T} J_{pj} g^q_j, \quad \forall p, q \in F \tag{2.27}$$

$$(\widehat{\mathbf{h}}_F)_p = h_p - \sum_{j\in\mathcal{N}(p)\cap T} J_{pj}\mu^{\mathcal{T}}_j, \quad \forall p \in F. \tag{2.28}$$

Then we solve a small inference problem involving only the feedback nodes and obtain the mean vector in $\boldsymbol{\mu}_F$ and the full covariance matrix Σ_F at the feedback nodes using

$$\Sigma_F = \widehat{J}_F^{-1} \tag{2.29}$$

$$\boldsymbol{\mu}_F = \widehat{J}_F^{-1}\widehat{\mathbf{h}}_F. \tag{2.30}$$

Figure 2.4b gives an illustration for this step.

Step 4: After the feedback nodes compute their own variances and means, their inference results are used to correct the partial variances $\Sigma^{\mathcal{T}}_{ii}$ and "partial means" $\mu^{\mathcal{T}}_i$ computed in Step 2.

The partial variances are corrected by adding correction terms using

$$\Sigma_{ii} = \Sigma^{\mathcal{T}}_{ii} + \sum_{p,q\in F} g^p_i \Sigma_{pq} g^q_i, \qquad \forall i \in T. \tag{2.31}$$

The partial means are corrected by running a second round of LBP with revised potential vector $\widetilde{\mathbf{h}}_T$ and the same information matrix J_T. The revised potential vector

is computed as follows:

$$\tilde{h}_i = h_i - \sum_{j \in \mathcal{N}(i) \cap F} J_{ij}(\boldsymbol{\mu}_F)_j, \ \forall i \in T. \tag{2.32}$$

Since this revision only uses local values, it can be viewed as passing messages from the feedback nodes to their neighbors (c.f. Figure 2.4c). Then a second round of LBP is performed on the subgraph $\mathcal{T}$ with model parameters J_T and $\widetilde{\mathbf{h}}_T$. After convergence, the final means are obtained, such that if $\mathcal{T}$ is a tree, this message-passing algorithm provides the true means, namely,

$$\mu_i = (J_T^{-1}\widetilde{\mathbf{h}}_T)_i, \ \forall i \in T. \tag{2.33}$$

An illustration of this step is shown in Figure 2.4d.

The complete message update equations (except for the selection of the feedback nodes) of FMP is summarized in Algorithm 2.2.2. We also provide some theoretical results in the following propositions and theorems, whose proofs can be found in [16].

Algorithm 2.2.1 Selection of the Feedback Nodes

Input: information matrix J and the maximum size k of the pseudo-FVS

Output: a pseudo-FVS F

1. Let $F = \emptyset$ and normalize J to have unit diagonal.
2. Repeat until $|F| = k$ or the remaining graph is empty.
   (a) Clean up the current graph by eliminating all the tree branches.
   (b) Update the scores $p(i) = \sum_{j \in \mathcal{N}(i)} |J_{ij}|$ on the remaining graph
   (c) Put the node with the largest score into F and remove it from the current graph.

Theorem 2.2.5 : *The FMP algorithm described in Algorithm 2.2.2 results in the exact means and exact variances for all nodes if F is an FVS.*

Algorithm 2.2.2 Feedback Message Passing Algorithm

Input: information matrix J, potential vector $\mathbf{h}$ and (pseudo-) feedback vertex set F of size k

Output: mean μ_i and variance Σ_{ii} for every node i

1. Construct k extra potential vectors: $\forall p \in F$, $\mathbf{h}^p = J_{T,p}$, each corresponding to one feedback node.

2. Perform LBP on $\mathcal{T}$ with J_T, $\mathbf{h}_T$ to obtain $\Sigma_{ii}^{\mathcal{T}} = (J_T^{-1})_{ii}$ and $\mu_i^{\mathcal{T}} = (J_T^{-1}\mathbf{h}_T)_i$ for each $i \in T$. With the k extra potential vectors, calculate the feedback gains $g_i^1 = (J_T^{-1}\mathbf{h}^1)_i, g_i^2 = (J_T^{-1}\mathbf{h}^2)_i, \ldots, g_i^k = (J_T^{-1}\mathbf{h}^k)_i$ for $i \in T$ by LBP.

3. obtain a size-k subgraph with $\widehat{J}_F$ and $\widehat{\mathbf{h}}_F$ given by

$$(\widehat{J}_F)_{pq} = J_{pq} - \sum_{j \in \mathcal{N}(p) \cap T} J_{pj} g_j^q, \ \forall p, q \in F, \tag{2.34}$$

$$(\widehat{\mathbf{h}}_F)_p = h_p - \sum_{j \in \mathcal{N}(p) \cap T} J_{pj} \mu_j^{\mathcal{T}}, \ \forall p \in F, \tag{2.35}$$

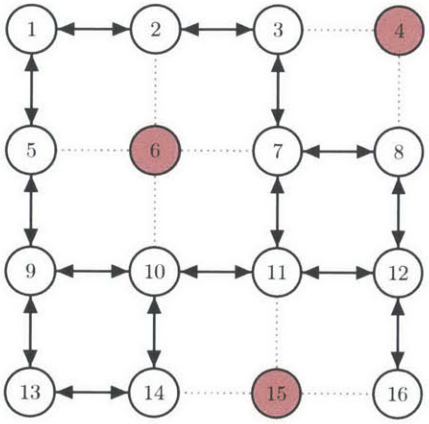
   and solve the inference problem on the small graph by $\Sigma_F = \widehat{J}_F^{-1}$ and $\boldsymbol{\mu}_F = \widehat{J}_F^{-1}\widehat{\mathbf{h}}_F$.

4. Revise the potential vector on $\mathcal{T}$ using

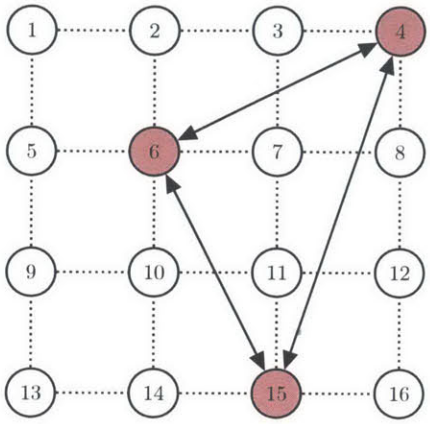
$$\tilde{h}_i = h_i - \sum_{j \in \mathcal{N}(i) \cap F} J_{ij}(\boldsymbol{\mu}_F)_j, \ \forall i \in T.$$

5. Another round of BP with the revised potential vector $\widetilde{\mathbf{h}}_T$ gives the exact means for nodes on $\mathcal{T}$.
   Add correction terms to obtain the exact variances for nodes in T:

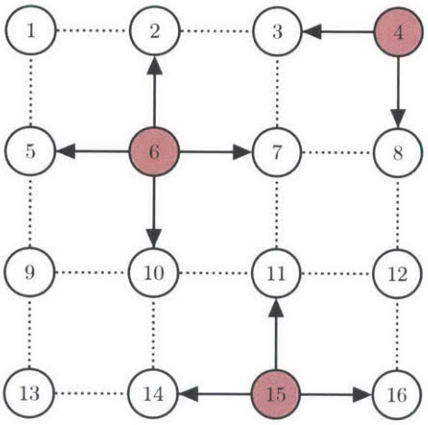
$$\Sigma_{ii} = \Sigma_{ii}^{\mathcal{T}} + \sum_{p \in \mathcal{F}} \sum_{q \in \mathcal{F}} g_i^p (\Sigma_{\mathcal{F}})_{pq} g_i^q, \ \forall i \in T.$$



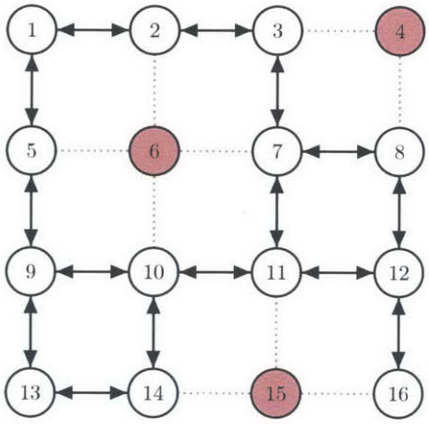
(a) LBP on the subgraph excluding the feedback nodes



(b) Solving a small inference problem among the feedback nodes



(c) Feedback nodes send feedback messages back to their neighbors



(d) Another round of LBP among the non-feedback nodes gives the final results

Figure 2.4: Illustration for the FMP algorithm. Shaded nodes (4, 6, and 15) are selected feedback nodes.

Theorem 2.2.6 : *Consider a Gaussian graphical model with parameters J and $\mathbf{h}$. If FMP converges with a pseudo-FVS F, it gives the correct means for all nodes and the correct variances on the pseudo-FVS. The variance of node i in T calculated by this algorithm equals the sum of all the backtracking walks of node i within T plus all the self-return walks of node i that visit F, so that the only walks missed in the computation of the variance at node i are the non-backtracking walks within T.*

Proposition 2.2.7 : *Consider a Gaussian graphical model with graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and model parameters J and $\mathbf{h}$. If the model is walk-summable, then FMP converges for any pseudo-FVS $F \subset \mathcal{V}$.*

Proposition 2.2.8 : *Consider a walk-summable Gaussian graphical model with n nodes. Assume the information matrix J is normalized to have unit diagonal. Let ϵ_{FMP} denote the error of FMP and $\widehat{\Sigma}_{ii}^{\text{FMP}}$ denote the estimated variance of node i. Then*

$$\epsilon_{\text{FMP}} = \frac{1}{n} \sum_{i \in \mathcal{V}} |\widehat{\Sigma}_{ii}^{\text{FMP}} - \Sigma_{ii}| \leq \frac{n-k}{n} \frac{\tilde{\rho}^{\tilde{g}}}{1-\tilde{\rho}},$$

where k is the number of feedback nodes, $\tilde{\rho}$ is the spectral radius corresponding to the subgraph T, and $\tilde{g}$ denotes the girth of T, i.e., the length of the shortest cycle in T. In particular, when $k = 0$, i.e., LBP is used on the entire graph, we have

$$\epsilon_{\text{LBP}} = \frac{1}{n} \sum_{i \in \mathcal{V}} |\Sigma_{ii}^{\text{LBP}} - \Sigma_{ii}| \leq \frac{\rho^g}{1-\rho},$$

where the notation is similarly defined.

■ 2.3 Common Sampling Algorithms

In this section, we summarize some commonly used sampling algorithms including using the Cholesky decomposition, forward sampling on trees (and beyond), and Gibbs sampling (with its variants).

Sampling Using the Cholesky Decomposition The Cholesky decomposition gives a lower triangular matrix L such that $J = LL^T$. Let $\mathbf{z}$ be an n-dimensional random vector whose entries are drawn *i.i.d.* from the standard Gaussian distribution $\mathcal{N}(0,1)$. An exact sample $\mathbf{x}$ can be obtained by computing $\mathbf{x} = (L^T)^{-1}\left(\mathbf{z} + L^{-1}\mathbf{h}\right)$. If such a

decomposition is available and if L is sparse, sampling is fast even for very large models. However, for a general sparse J, the computation of L has cubic complexity, while fill in L can be quadratic in the size of the model. For very large models, the Cholesky decomposition is computationally prohibitive.[9]

Forward Sampling for Tree-Structured Models For a tree-structured GGM, an exact sample can be generated in linear time (with respect to the number of nodes) by first computing the variances and means for all nodes and covariances for the edges using BP, and then sampling the variables one by one following a root-to-leaf order where the root node can be an arbitrary node [23].

Forward Sampling for Models with Small Feedback Vertex Sets There are other tractable graphical models that one can consider, including models with small FVSs. In this case, one can compute the means and covariances using the FMP algorithm that scales quadratically in the size of the FVS and linearly in the overall size of the graph and can then produce samples by first sampling the nodes in the FVS (perhaps using the Cholesky decomposition, with complexity cubic in the size of the FVS) and then performing forward tree sampling on the rest.

Basic Gibbs Sampling The basic Gibbs sampler generates new samples, one variable at a time, by conditioning on the most recent values of its neighbors. In particular, in each iteration, a sample for all n variables is drawn by performing

$$x_i^{(t+1)} \sim \mathcal{N}(\frac{1}{J_{ii}} \left(h_i - \sum_{j<i,\ j\in\mathcal{N}(i)} J_{ji} x_j^{(t+1)} - \sum_{j>i,\ j\in\mathcal{N}(i)} J_{ji} x_j^{(t)} \right), J_{ii}^{-1}) \text{ for } i = 1, 2, \ldots . n.$$

The Gibbs sampler always converges when $J \succ 0$; however, the convergence can be very slow for many GGMs, including many tree-structured models. More details on Gibbs sampling can be found in [24].

Variants of Gibbs Sampling There have been many variants of the Gibbs sampler using the ideas of reordering, coloring, blocking, and collapsing. For example, in the blocked Gibbs sampler the set of nodes is partitioned into several disjoint subsets and each subset is treated as a single variable. One approach is to use graph coloring, in which variables

[9] Sparse Cholesky decomposition can be employed to reduce the computational complexity. However, even for sparse graphs, the number of fills in the worst case is still $\mathcal{O}(n^2)$ and the total computational complexity is $\mathcal{O}(n^3)$ in general [31].

are colored so that adjacent nodes have different colors, and then each Gibbs block is the set of nodes in one color [57]. In [29] the authors have proposed a blocking strategy where each block induces a tree-structured subgraph.

2.4 Learning Graphical Models

Learning graphical models refers to the procedure of recovering the graph structure as well as model parameters of an unknown model given observations. In this section, we first give a brief introduction to some useful notions in information theory that will be use in our problem formulation or proofs. Next we introduce the maximum likelihood criterion for structure and parameter learning and its equivalent formulation as an optimization problem. Finally, we summarize the Chow-Liu algorithm, which has been proposed for efficiently learning models in the family of trees.

2.4.1 Information Quantities

In the following we review some important information quantities with brief descriptions.

The *entropy* of a probabilistic distribution is defined as

$$H_{p_{\mathbf{x}}}(\mathbf{x}) \triangleq -\int_{\mathbf{x}} p_{\mathbf{x}}(\mathbf{x}) \log p_{\mathbf{x}}(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{2.36}$$

The *conditional entropy* is the expected entropy of the conditional distribution, i.e.,

$$H_{p_{\mathbf{x},\mathbf{y}}}(\mathbf{x}|\mathbf{y}) \triangleq -\int_{\mathbf{x},\mathbf{y}} p_{\mathbf{x}\mathbf{y}}(\mathbf{x},\mathbf{y}) \log p_{\mathbf{x}|\mathbf{y}}(\mathbf{x}|\mathbf{y}) \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{y}. \tag{2.37}$$

The *mutual information* of two variables or two sets of variables is a nonnegative measure of the variables' (or sets of variables') mutual dependence:

$$I_{p_{\mathbf{x},\mathbf{y}}}(\mathbf{x};\mathbf{y}) \triangleq \int_{\mathbf{x},\mathbf{y}} p_{\mathbf{x}\mathbf{y}}(\mathbf{x},\mathbf{y}) \log \frac{p_{\mathbf{x}}(\mathbf{x}) p_{\mathbf{y}}(\mathbf{y})}{p_{\mathbf{x}\mathbf{y}}(\mathbf{x},\mathbf{y})} \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{y}. \tag{2.38}$$

The mutual information between two sets of random variables that are jointly Gaussian is

$$I(\mathbf{x};\mathbf{y}) = \frac{1}{2} \log \frac{\det \Sigma_{\mathbf{x}} \det \Sigma_{\mathbf{y}}}{\det \Sigma}, \tag{2.39}$$

where $\Sigma = \begin{bmatrix} \Sigma_{\mathbf{x}} & \Sigma_{\mathbf{x}\mathbf{y}} \\ \Sigma_{\mathbf{y}\mathbf{x}} & \Sigma_{\mathbf{y}} \end{bmatrix}$ is the covariance matrix. In particular, the mutual informa-

tion between two scalar jointly Gaussian variables is $I(x;y) = -\frac{1}{2}\log(1-\rho^2)$, where ρ is the correlation coefficient. The *conditional mutual information* is useful to express the mutual information of two random variables (or two sets of random variables) conditioned on a third. It is defined as follows

$$I_{p_{\mathbf{x},\mathbf{y},\mathbf{z}}}(\mathbf{x};\mathbf{y}|\mathbf{z}) \triangleq \int_{\mathbf{x},\mathbf{y},\mathbf{z}} p_{\mathbf{xyz}}(\mathbf{x},\mathbf{y},\mathbf{z}) \log \frac{p_{\mathbf{xy}|\mathbf{z}}(\mathbf{x},\mathbf{y}|\mathbf{z})}{p_{\mathbf{x}|\mathbf{z}}(\mathbf{x}|\mathbf{z})p_{\mathbf{y}|\mathbf{z}}(\mathbf{y}|\mathbf{z})} \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{y}\mathrm{d}\mathbf{z}. \tag{2.40}$$

The *Kullback-Leibler divergence* or *K-L divergence* is a non-symmetric nonnegative measure of the difference between two distributions:

$$D_{\mathrm{KL}}(p_{\mathbf{x}}||q_{\mathbf{x}}) \triangleq \int_{\mathbf{x}} p_{\mathbf{x}}(\mathbf{x}) \log \frac{p_{\mathbf{x}}(\mathbf{x})}{q_{\mathbf{x}}(\mathbf{x})}. \tag{2.41}$$

The K-L divergence is always nonnegative. It is zero if and only if the two distributions are the same (almost everywhere). The *conditional K-L divergence* between two conditional distributions $p_{\mathbf{x}|\mathbf{y}}(\mathbf{x}|\mathbf{y})$ and $q_{\mathbf{x}|\mathbf{y}}(\mathbf{x}|\mathbf{y})$ under distribution $p_{\mathbf{y}}(\mathbf{y})$ is the expected K-L divergence defined as

$$D_{\mathrm{KL}}(p_{\mathbf{x}|\mathbf{y}}||q_{\mathbf{x}|\mathbf{y}}|p_{\mathbf{y}}) \triangleq \mathbb{E}_{p_{\mathbf{y}}}\left[D_{\mathrm{KL}}(p_{\mathbf{x}|\mathbf{y}=y}||q_{\mathbf{x}|\mathbf{y}=y})|\mathbf{y}=y\right] \tag{2.42}$$

$$= D(p_{\mathbf{x}|\mathbf{y}}p(\mathbf{y})||q_{\mathbf{x}|\mathbf{y}}p_{\mathbf{y}}). \tag{2.43}$$

When there is no confusion, we often omit the subscripts in the distributions, e.g., $I_{p_{\mathbf{x},\mathbf{y}}}(\mathbf{x};\mathbf{y})$ written as $I_p(\mathbf{x};\mathbf{y})$. With a slight abuse of notation, we also use $p(\mathbf{x}_A)$ to denote the marginal distribution of $\mathbf{x}_A$ under the joint distribution $p(\mathbf{x})$, and similarly $p(\mathbf{x}_A|\mathbf{x}_B)$ to denote the conditional distribution of $\mathbf{x}_A$ given $\mathbf{x}_B$ under the joint distribution $p(\mathbf{x})$.

2.4.2 Maximum Likelihood Estimation

Learning graphical models refers to recovering the underlying graph structures and model parameters from observations, where the models are often known or assumed to be in a family of models. The *maximum likelihood* (ML) criterion is to select the model such that the observed data has the maximum likelihood. The estimated model using the ML criterion is called the ML estimate. In the following, we define the ML criterion and introduce its equivalent formulation.

Given samples $\{\mathbf{x}^i\}_{i=1}^s$ independently generated from an unknown distribution q in

the family $\mathcal{Q}$, the ML estimate is defined as

$$q_{\mathrm{ML}} = \arg\max_{q\in\mathcal{Q}} \prod_{i=1}^{s} q(\mathbf{x}^i) \tag{2.44}$$

$$= \arg\max_{q\in\mathcal{Q}} \sum_{i=1}^{s} \log q(\mathbf{x}^i). \tag{2.45}$$

It has been shown that computing the ML estimate is equivalent to minimizing the K-L divergence between the empirical distribution and the distributions in the family. The following proposition 2.4.1 states this equivalence. The proof of this proposition can be found in standard texts such as in [58].

Proposition 2.4.1 : *Given independently generated samples* $\{\mathbf{x}^i\}_{i=1}^s$, *the ML estimate* $q_{ML} = \arg\max_{q\in\mathcal{Q}} \sum_{i=1}^{s} \log q(\mathbf{x}^i)$ *can be computed using*

$$q_{ML} = \arg\min_{q\in\mathcal{Q}} D_{KL}(\hat{p}||q), \tag{2.46}$$

where $\hat{p}$ *is the empirical distribution of the samples.*

For Gaussian distributions, the empirical distribution can be written as

$$\hat{p}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \hat{\boldsymbol{\mu}}, \hat{\Sigma}), \tag{2.47}$$

where the empirical mean

$$\hat{\boldsymbol{\mu}} = \frac{1}{s} \sum_{i=1}^{s} \mathbf{x}^i \tag{2.48}$$

and the empirical covariance matrix

$$\hat{\Sigma} = \frac{1}{s} \sum_{i=1}^{s} \mathbf{x}^i \left(\mathbf{x}^i\right)^T - \hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^T. \tag{2.49}$$

For more general models, the expectation-maximization (EM) algorithm is often used to iteratively find the ML estimate of the model parameters. The general steps of the EM algorithm can be found in [59].

2.4.3 The Chow-Liu Algorithm

For the family of tree-structured models, the ML estimate can be computed exactly using the Chow-Liu algorithm, where the graph structure is obtained by computing the maximum spanning tree (MST) with the weight of each edge equal to the empirical mutual information (the mutual information between the two nodes of the edge computed under the empirical distribution) and then the model parameters are computed using information projection [36]. In Algorithm 2.4.1, we summarize the Chow-Liu algorithm specialized for GGMs. The input is the empirical covariance matrix $\hat{\Sigma}$ and the outputs are Σ_{CL}, the estimated covariance matrix that has a tree-structured inverse, and $\mathcal{E}_{\mathrm{CL}}$, the set of edges in the learned model. The computational complexity of Algorithm 2.4.1 is $\mathcal{O}(n^2 \log n)$, where n is the number of nodes.

Algorithm 2.4.1 The Chow-Liu Algorithm for GGMs

Input: the empirical covariance matrix $\hat{\Sigma}$
Output: Σ_{CL} and $\mathcal{E}_{\mathrm{CL}}$

1. Compute the correlation coefficients $\rho_{ij} = \frac{\hat{\Sigma}_{ij}}{\sqrt{\hat{\Sigma}_{ii}\hat{\Sigma}_{jj}}}$ for all $i,\ j \in \mathcal{V}$.
2. Find an MST (maximum weight spanning tree) of the complete graph with weights $|\rho_{ij}|$ for edge (i, j). The edge set of the tree is denoted as $\mathcal{E}_T$.
3. The entries in Σ_{CL} are computed as follows
   (a) For all $i \in \mathcal{V}$, $(\Sigma_{\mathrm{CL}})_{ii} = \hat{\Sigma}_{ii}$;
   (b) for $(i, j) \in \mathcal{E}_T$, $(\Sigma_{\mathrm{CL}})_{ij} = \hat{\Sigma}_{ij}$;
   (c) for $(i, j) \notin \mathcal{E}_T$, $(\Sigma_{\mathrm{CL}})_{ij} = \sqrt{\Sigma_{ii}\Sigma_{jj}} \prod_{(l,k)\in \mathrm{Path}(i,j)} \rho_{lk}$, where $\mathrm{Path}(i, j)$ is the set of edges on the unique path between i and j in the spanning tree.

Chapter 3

Recursive Feedback Message Passing for Distributed Inference

3.1 Introduction

In Section 2.2, we have described the FMP algorithm proposed in [16]. FMP uses the standard LBP message-passing protocol among the nodes that are not in the FVS and uses a special protocol for nodes in the FVS. The FMP algorithm gives the exact means and variances for all nodes with a total computational complexity that is quadratic in the size of the FVS and linear in the total number of nodes. When the size of the FVS is large, a pseudo-FVS is used instead of a full FVS to obtain approximate inference results. By performing two rounds of standard LBP among the non-feedback nodes and solving a small inference problem among the feedback nodes[1], FMP improves the convergence and accuracy significantly compared with running LBP on the entire graph. In addition, choosing the size of the pseudo-FVS enables us to make the trade-off between efficiency and accuracy explicit.

The overall message-passing protocol of FMP is indeed distributed among the non-feedback nodes since the messages among them are updated using only local parameters or incoming messages from neighbors; however, centralized communication (i.e., propagating information between nodes without connecting edges) among the feedback nodes is still required when solving the smaller inference problem among these nodes. Moreover, the set of feedback nodes (either forming an FVS or a pseudo-FVS) are selected in a centralized manner prior to running FMP (c.f. Algorithm 2.2.1 in Section 2.2).

[1] As mentioned in Section 2.2, nodes in the FVS or pseudo-FVS are called feedback nodes.

Hence, we refer to FMP as a *hybrid* algorithm. One can ask some natural questions: Is it possible to select the feedback nodes in a purely distributed manner? Can we further eliminate the centralized communication among the feedback nodes in FMP without losing the improvements on convergence and accuracy?

In this chapter, we propose and analyze the *recursive FMP* algorithm, which is a purely distributed extension of FMP where all nodes use the same distributed message-passing protocol in the entire procedure. In recursive FMP, an inference problem on the entire graph is recursively (but in a distributed manner) reduced to those on smaller and smaller subgraphs until the final inference problem can be solved efficiently by an exact or approximate message-passing algorithm. In this algorithm, all messages are passed between nodes with connecting edges. Furthermore, the *election*[2] of the feedback nodes is integrated into the distributed protocol so that each node uses incoming messages to determine whether it itself is a feedback node. In this recursive approach, centralized communication among feedback nodes in FMP is reduced to *message forwarding*[3] from the feedback nodes. Such a purely distributed algorithm is of great importance because in many scenarios, for example wireless sensor networks, it is easy to implement the same distributed protocol on all nodes while centralized computation is often expensive or impractical. Moreover, this algorithm now shares with LBP the characteristic that each node receives messages and performs computations using exactly the same protocol. Throughout this chapter, we use the same notation for the model parameters as in Section 2.1.3. In particular, we assume that the information matrix J is normalized to have unit diagonal.[4] In addition, without loss of generality, we assume that the underlying graphs are connected.[5]

The remainder of this chapter is organized as follows. First in Section 3.2, we describe the recursive FMP algorithm in three separate stages. Then in Section 3.3, we summarize the recursive FMP algorithm as a single integrated protocol without the separation of stages. Next we present and prove our theoretical results using walk-sum

[2]When an algorithm is distributed, the word "election" is used in place of "selection" to emphasize the distributed nature.

[3]Message passing is also called message forwarding if messages are passed without being modified.

[4]The information matrix J is normalized using $J \leftarrow D^{-\frac{1}{2}} J D^{-\frac{1}{2}}$, where D is a diagonal matrix having the same diagonal as J.

[5]When the underlying graph of a graphical model is not connected, then the random variables in different connected components are independent. Hence, the inference problem on the entire graph can be solved by considering inference problems on individual connected components.

analysis in Section 3.4. Finally in Section 3.5, we demonstrate the performance of the algorithm using simulated models on grids as well as real data for estimating sea surface height anomaly (SSHA).

3.2 Recursive FMP Described by Stages

In this section, we describe the message-passing protocol used in recursive FMP in separate stages. In practice, all nodes use the same integrated protocol (while they may execute different message-update rules at a particular time depending on their internal status). However, for clarity, we present the protocol in three separate stages: 1) election of feedback nodes; 2) initial estimation; and 3) recursive correction. For each stage, we explain the motivation and illustrate the protocol with examples.

Overview

It is useful to understand that the FMP algorithm described in Section 2.2.3 can be interpreted as, first organizing the nodes into two sets (feedback and non-feedback), then performing Gaussian elimination of the non-feedback nodes (or an approximation to it using LBP if a pseudo-FVS is used), then solving the reduced problem on the set of feedback nodes, followed by back-substitution (accomplished via the second wave of LBP). At a coarse level, one can think of our distributed algorithm as continuing to perform Gaussian elimination to solve the problem on the non-feedback nodes rather than performing this in a centralized fashion, where these nodes need to determine on the fly which ones will begin Gaussian elimination and back-substitution and in what order. Our fully integrated algorithm in Section 3.3 combines all of these steps together, so that each node knows, from a combination of its own internal memory and the messages that it receives, what role it is playing at each step of the algorithm.

In the following, we first contrast our distributed algorithm with the FMP algorithm described in Section 2.2.3, which can be directly interpreted as having distinct stages. Then we describe our algorithm in several stages as well (although as we discuss, even in this staged version, several of these stages actually may run together). In doing so, we will also need to be much more careful in describing the protocol information that accompanies messages in each as well as the quantities stored during each stage at each node. Ultimately, in Section 3.3, we describe an integrated algorithm without

explicit stages, while in Section 3.4 we present theoretical results on the correctness of our algorithm.

To begin, we briefly re-examine the hybrid FMP algorithm in Section 2.2.3 (distributed among the non-feedback nodes while centralized among the feedback nodes both in the selection and in message passing). First, there is one parameter that needs to be specified *a priori*, namely k, the maximum number of feedback nodes that are to be included. This algorithm can be thought of as having several stages:

1. Identify feedback nodes, either for a full FVS or for a pseudo-FVS. As we discuss in Section 2.2.3, the greedy algorithm (Algorithm 2.2.1) we suggest involves computing "priority scores" for each node, choosing the node with the highest score, recomputing scores, and continuing. For this algorithm, once the set of feedback nodes have been computed:

   (a) The scores are thrown away and have no further use.

   (b) All nodes are aware of which nodes are feedback nodes and which are not.

2. Perform a first round of LBP among the non-feedback nodes. As described in Section 2.2.3, this is done with a set of auxiliary "mean" computations, corresponding to the feedback gains to be used by the feedback nodes. Explicitly, this means that if there are k feedback nodes, all non-feedback nodes must compute and then send $k+2$ messages, the "2" corresponding to the LBP messages using the potential vector and "J" matrix for the non-feedback part of the graph (c.f. 2.10–(2.12)) as well as a message for each of the k feedback nodes (which correspond to computations analogous to the computation of means on that part of the graph but with the non-zero potential of each initiated with non-zero values at the immediate neighbors of each feedback node in the remaining graph). These imply the following:

   (a) Each message sent has enough protocol information to identify it—i.e., is it one of the two messages corresponding to LBP on the non-feedback graph or is it a message associated with the computation of the feedback gain of a particular feedback node.

   (b) Each node stores and then updates its values of these $k+2$ quantities. Note that in this stage of the computation, the feedback nodes are *inactive*, i.e.,

they do not contribute to the message passing after they provide the initial potentials for the computation of feedback gains. All of the non-feedback nodes are *active*, as they participate in the message-passing. As all nodes are aware of the set of feedback nodes, each node knows who is active and who is not, including which of its neighbors is active.

3. Perform exact inference on the set of feedback nodes. In general this requires dense communication among the feedback nodes (which are typically not neighbors of each other), but results in the mean and full covariance on the vector of variables at the set of feedback nodes.

4. Add correction terms to the partial variances obtained in Step 2, where the correction terms are computed from the inference results in Step 3 and the feedback gains in Step 2. This requires all of the feedback nodes communicating to all of the nodes in T, either by direct communication or by message forwarding. The final estimates of the means are computed by performing a second round of LBP among the feedback nodes with a modified potential vector, where each entry of the potential vector is modified by adding correction terms corresponding to its neighboring feedback nodes.

As we have discussed in Section 2.2.3, if k is large enough to allow for a complete FVS to be used, the algorithm produces exact means and variances. If k is not large enough and if the algorithm converges (which our choice of scoring and greedy choice of feedback nodes aims to ensure), the means are correct and the variances, while incorrect, collect more walks than LBP on the entire graph.

Finally, we note that in describing the algorithm in Section 2.2.3, we did not emphasize the "status" of nodes during the steps of the algorithm, but it is worth doing so now, as this is a much more important issue with the distributed algorithm. Specifically, in the hybrid algorithm as the process of choosing feedback nodes begins, all nodes are undecided, a status we denote by "U". As feedback nodes are chosen, they switch to a status which we denote by "F", although we also call these nodes "inactive" as they do not participate in the LBP operations. The remaining nodes (after all feedback nodes are chosen) take on status "T", which we also refer to as "active." In this hybrid algorithm each node's status remains unchanged throughout the algorithm. In contrast, as we will see, in our purely distributed algorithm, inactive nodes can become active (and switch status state to "T") as they join the LBP/Gaussian elimination stage.

In contrast, the "staged" version of our distributed algorithm completely avoids the non-local communication in Step 3 of the hybrid algorithm and, in addition, allows nodes to operate with knowledge gained only through local communication. In its general form, each node needs to have three parameters specified, whose purpose is made clear in the following sections. Also, in its general form, each node may have a different and dynamically evolving set of nodes that it includes in its list of feedback nodes. These three parameters are:

- An *effective diameter* d_i for each node i, which can be interpreted as a default estimate of the network diameter. This is used in our algorithm for "electing" feedback nodes (Stage I), where, in essence, each node does not look beyond a certain distance from itself. As we will see, the election of feedback nodes has a two-layered set of iterations, where the effective diameters control the "inner" iterations in this stage.

- A number of outer iterations l_i in which each node i participates in the election of feedback nodes.

- A capacity K_i, which is the maximum number of inactive nodes that each node i is allowed to keep track of.

For simplicity we will describe our algorithm assuming that these three parameters are constant across nodes. As mentioned previously, the introduction of this staged algorithm is included for expository reasons, and the integrated single-protocol algorithm in Section 3.3 combines all of these stages. In particular, as we proceed with our "staged" version, we will systematically describe how components of the message protocols and especially the memory at individual nodes are added to accommodate the needs of successive stages (again, the integrated algorithm has all of this memory structure from the start).

The "stages" of our algorithm are the following:

Stage I A first stage of electing feedback nodes, i.e., the nodes that have status F and are initially inactive, combining the so-called leader election algorithm [60] and the algorithm for eliminating tree branches. As this stage proceeds, each node keeps track of the node it has heard from that has the highest score and then identifies itself as a feedback node if it receives messages indicating that it itself is the highest scoring node seen in a circuit of diameter d and it is at least distance d from the end node of

any tree branch. At the end of this stage, each node either (a) knows that it is not a feedback node; or (b) knows that it is a feedback node and also knows and stores its own score. At this point, this is the only memory that each node has. While there is other information that could be gleaned by each node concerning other nodes we do not include that in stored memory at this point, as that knowledge becomes exposed as the subsequent stages proceed. So, in contrast to the hybrid algorithm, at this point each node has stored only local information about its own status (feedback or not) but, for each feedback node its priority score that was thrown away in the hybrid algorithm is maintained. This algorithm has several important properties: For example, if the effective diameter is sufficiently large, if a sufficient number of outer iterations is allowed, and if the remaining graph of active nodes stays connected, then the nodes identified as feedback nodes in this algorithm will equal those in the centralized selection algorithm.

Stage II The second stage of the distributed algorithm is roughly equivalent to Step 2 of the hybrid algorithm, and involves message passing among active nodes. The protocol and memory for the base set of two messages corresponding to LBP on the active nodes are exactly the same as in the hybrid algorithm, but each active node only begins to augment the number of messages it sends and quantities it stores (in particular to compute feedback gains) as it receives incoming messages, which carry information about the existence of inactive nodes. The number of messages each node i sends may be as large as $K_i + 2$ (plus extra information bits including the inactive node indices and priority scores).

Stage III The third stage of recursive correction can be viewed as counterpart of Step 3 and Step 4 of the hybrid algorithm combined. However, the most important difference is that here no centralized communication is needed, i.e., no messages between two inactive nodes without direct links are passed and there is no solving of the inference problem on the set of inactive nodes in a centralized manner. When some local conditions are satisfied, an inactive node is triggered to convert to an active node, where this conversion consists of the following steps: (a) the inactive node computes some local values that store the current estimate of its mean and variance (which are similar to those stored at current active nodes); (b) sends these values (in the form of a *correction message*) to neighboring active nodes for forwarding; and (c) becomes an active node and follows the protocols for active nodes from now on. At the same time, an active node in this stage only corrects its local values upon receiving correction messages and forwards those

messages to neighboring active nodes.

3.2.1 Stage I: Election of Feedback Nodes

In the first stage, the set of feedback nodes are elected in a distributed manner. The nodes that are elected are called feedback nodes because they play a similar role as the feedback nodes used in standard FMP (Algorithm 2.2.2). Specifically, the set of "feedback nodes" refers to the *fixed* set of nodes that are elected at the end of Stage I, while the set of "inactive" nodes refers to the set of nodes that do not participate in message passing, which is initially simply the set of feedback nodes but may change at each iteration as inactive nodes become active. Similarly, the set of "non-feedback" nodes[6] refers to the *fixed* set of nodes that are not elected at the end of Stage I, while the term "active nodes" refers to the set of nodes that participate in message passing, which is initially the set of non-feedback nodes but will increase as inactive nodes switch status. We use S_i to denote the status of node i, where the status can be U (undecided), F (inactive) or T (active). Initially, all nodes have status U, but as our overall algorithm proceeds, nodes change status from U to either F or T and in subsequent stages nodes with status F may change to T. We use U to denote the set of nodes with status U and $\mathcal{N}_{\mathrm{U}}(i)$ to denote the set of i's neighbors with status U, i.e., $\mathcal{N}_{\mathrm{U}}(i) = \{j | j \in \mathcal{N}(i) \text{ and } S_i = U\}$. The symbols F, T, $\mathcal{N}_{\mathrm{F}}(i)$, and $\mathcal{N}_{\mathrm{T}}(i)$ are similarly defined. We note that sets such as $\mathcal{N}_{\mathrm{U}}(i)$ will change from iteration to iteration. In addition, we use $\mathcal{E}_T$ to denote the set of edges among the active nodes, i.e., $\mathcal{E}_T = \{(i,j) | (i,j) \in \mathcal{E} \text{ and } i,j \in T\}$.

The message-passing protocol in the first stage is motivated by the greedy pseudo-FVS selection algorithm (Algorithm 2.2.1). In the greedy algorithm, each node i has the priority score $p_i = \sum_{j \in \mathcal{N}(i)} |J_{ij}|$ which arises from the theoretical results on convergence [16]. The node with the highest priority score that also breaks some cycles is selected as the first feedback node. Then the algorithm continues to select the cycle-breaking node with the highest priority score[7] among the remaining nodes until a certain number of feedback nodes are selected.[8]

[6]These non-feedback nodes are counterparts of the nodes in the pseudo-tree in standard FMP [16]. The subgraph induced by the non-feedback nodes may still have cycles if a pseudo-FVS is used.

[7]The priority scores are now updated to exclude the contribution of the already selected feedback node.

[8]As discussed in [16], the number of feedback nodes to select is determined by how much computational resource is available.

We first describe the well-established "Leader Election Algorithm" as well as two extensions that (a) allow for the election of more than one leader; and (b) that take into account that we do not wish to include as leaders nodes that are on tree branches. These then lead to our Stage I Algorithm 3.2.4, which elects feedback nodes that are given inactive status F. We first describe the well-established "Leader Election Algorithm" as well as two extensions that (a) allow for the election of more than one leader; and (b) that take into account that we don't wish to include as leaders nodes that are on tree branches. These then lead to our Stage I Algorithm 3.9. Throughout the following part of this chapter, without loss of generality, we assume that the priority scores are distinct, i.e., there are no ties.[9]

Leader Election Algorithm The leader election algorithm is a purely distributed algorithm that elects the node with the highest priority score regardless of the graph structure (i.e., whether it breaks cycles) [60]. In this algorithm, each node i stores two scalar values locally: MaxScore(i) and MaxId(i) representing the maximum priority score that node i has "seen" so far and the corresponding node index. Every node has status U at the beginning of the algorithm and will eventually change its status to F if it is elected as the leader. At each iteration, each node i sends the current values of MaxScore(i) and MaxId(i) to its neighbors and updates the values of MaxScore(i) and MaxId(i) based on incoming messages. In the end, a node proclaims itself has a leader if it itself has the highest priority score it has seen. Figure 3.1 provides an illustrating example of the message-passing protocol. Currently, as the figure shows, node 2 thinks it itself has the highest priority score 0.3 (since MaxScore(2) = 0.3 and MaxScore(2) = 2), but among its neighbors, node 3 knows the highest priority score (which is 0.5 of node 8 as MaxScore(3)=0.5 and MaxId(3) = 8). Hence, the stored values at node 2 will change to MaxScore(2) = 0.5 and MaxId(2) = 8 at the end of this iteration. We summarize the leader election algorithm in Algorithm 3.2.1. It has been shown that if the number of iterations d is equal to or greater than the diameter of the graph, then after running Algorithm 3.2.1, the node with the highest priority will be the (only) node elected [60].

Algorithm 3.2.1 can be extended to elect the nodes with the top l priority scores. This extension can be done by repeating Algorithm 3.2.1 for l times (which also requires resetting the locally stored values and excluding the already elected nodes from message passing). We summarize the extended leader election algorithm in Algorithm 3.2.2.

[9]If there are ties of priorities scores, we can define an arbitrary tiebreaker, e.g., using node indices.

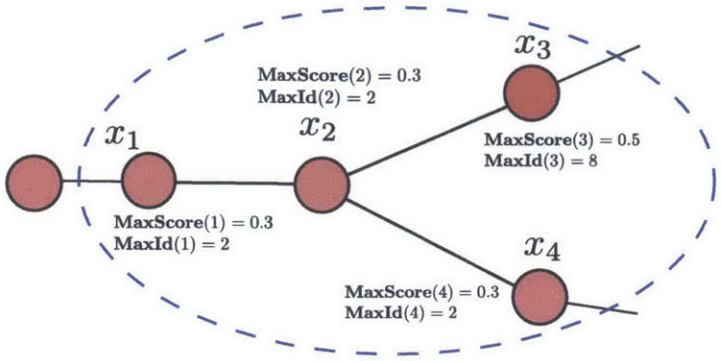


Figure 3.1: Illustrating example of the leader election algorithm. MaxScore(i) and MaxId(i) represent the maximum priority score node i has "seen" so far and the corresponding node index. These local values are shown for nodes 1, 2, 3 and 4 (corresponding to the random variables x_1, x_2, x_3, and x_4).

Algorithm 3.2.1 Message Protocol for Leader Election with General Scores

---

Input: priority score p_i for each $i \in \mathcal{V}$
Output: status S_i for each $i \in \mathcal{V}$
At each node $i \in \mathcal{V}$, simultaneously,

1. Initialization: $S_i \leftarrow$ U, $\text{MaxScore}^{(0)}(i) = p_i$, and $\text{MaxId}^{(0)}(i) = i$.
2. For $t = 1, 2, \ldots, d$:
   - Update local values using:

$$\text{MaxScore}^{(t)}(i) = \max_{j \in \mathcal{N}(i) \cup \{i\}} \text{MaxScore}^{(t-1)}(j)$$
$$\text{MaxId}^{(t)}(i) = j^*,$$

   where $j^* = \arg\max_{j \in \mathcal{N}(i) \cup \{i\}} \text{MaxScore}^{(t-1)}(j)$.
3. $S_i \leftarrow F$ if $i = \text{MaxId}(i)$.

Here we have assumed that the priority scores of the remaining nodes do not change after some leaders are elected. However, if the scores do change (as they will in our Stage I Algorithm 3.2.4), the algorithm can be easily modified to reset the local values accordingly in Step 2 (a) of Algorithm 3.2.2.

Algorithm 3.2.2 Extended Leader Election: Electing the Nodes with the Top-l Priority Scores with General Scoring Function

Input: priority score p_i for each $i \in \mathcal{V}$
Output: S_i for each $i \in \mathcal{V}$
Initialization: status $S_i \leftarrow$ U for each $i \in \mathcal{V}$

- Outer Iterations: repeat l times
  At each node with status U,

  1. $\text{MaxScore}^{(0)}(i) = p_i$ and $\text{MaxId}^{(0)}(i) = i$.
  2. Inner Iterations: for $t = 1,\ 2,\ \ldots, d$:
     - Update local values using:

$$\text{MaxScore}^{(t)}(i) = \max_{j \in \mathcal{N}_{\text{U}}(i) \cup \{i\}} \text{MaxScore}^{(t-1)}(j)$$
$$\text{MaxId}^{(t)}(i) = j^*,$$

     where $j^* = \arg\max_{j \in \mathcal{N}(i) \cup \{i\}} \text{MaxScore}^{(t-1)}(j)$.
  3. $S_i \leftarrow$ F if $i = \text{MaxId}(i)$.

  At each node i with status F,

1. $\text{MaxScore}^{(0)}(i) = -\text{Min}$ and $\text{MaxId}^{(0)}(i) = i$.
2. Inner Iterations: for $t = 1,\ 2,\ \ldots, d$:

$$\text{MaxScore}^{(t)}(i) = \max_{j \in \mathcal{N}(i)} \text{MaxScore}^{(t-1)}(j)$$
$$\text{MaxId}^{(t)}(i) = j^*,$$

where $j^* = \arg\max_{j \in \mathcal{N}(i)} \text{MaxScore}^{(t-1)}(j)$.

Distributed Elimination of Tree Branches We now describe a distributed algorithm that eliminates the tree branches. The high-level idea of this algorithm is that if we remove the degree-0 or degree-1 nodes one by one (where new degree-1 nodes may appear as nodes are removed), then the resulting graph will not have any tree branches. In this algorithm, all nodes are initialized with status U. At each iteration, at node i with status U, if $|\mathcal{N}_{\text{U}}(i)|$, the size of $\mathcal{N}_{\text{U}}(i)$, is less than or equal to one, then node i changes its

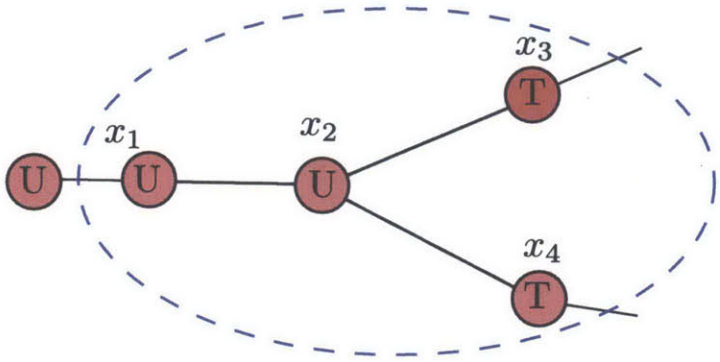


Figure 3.2: Elimination of tree branches. The current status of nodes 1, 2, 3, and 4 (corresponding to the random variables x_1, x_2, x_3, and x_4) are shown. The status of node 2 will change to T at the end of this iteration according to Algorithm 3.2.3.

status to T. Once a node has status T, it remains the same. Figure 3.2 provides an illustrating example: At the current iteration, node 2 has status U while only one of its neighbors has status U. Hence, the status of node 2 will change to T at the end of this iteration. We summarize this distributed protocol in Algorithm 3.2.3.

The following Lemma 3.2.1 states the correctness of Algorithm 3.2.3. The proof of Lemma 3.2.1 is provided in Appendix 3.6.

Algorithm 3.2.3 Elimination of Tree Branches

Input: the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$
Output: status S_i for each $i \in \mathcal{V}$
Initialization: $S_i \leftarrow$ U, for each $i \in \mathcal{V}$
At each node i with status U, simultaneously

- For $t = 1, 2, \ldots, d$:
  $S_i \leftarrow$ T if $|\mathcal{N}_{\mathrm{U}}(i)| \leq 1$.

At each node i with status T,
No action.

Lemma 3.2.1 : *After running Algorithm 3.2.3, all nodes that are not on the tree branches have status U regardless of d. If d is greater than the diameter of the graph, then every node i that is on some tree branch has status T.*

Distributed Protocol for Stage I Equipped with distributed algorithms for both leader election and eliminating tree branches, we can of course simply alternate between these two algorithms to elect the feedback nodes in a distributed manner. To further improve the efficiency, we propose a message-passing protocol that combines Algorithm 3.2.2 and Algorithm 3.2.3 by conducting the elimination of the tree branches and the election of the leader simultaneously. In addition, this algorithm also includes the re-computation of node scores as the sets $\mathcal{N}_{\mathrm{U}}(i)$ change from iteration to iteration.

As mentioned previously, at the beginning of the first stage, each node has status U, meaning that it is undecided whether it is a feedback node or a tree node. During message passing, its status may change to F (denoting the feedback nodes) or T (denoting the tree nodes). In this stage, the status of a node will remain the same during this stage once it changes from U to F or from U to T. Any nodes with status U at the end of the stage will change their status to T. We summarize this distributed protocol in Algorithm 3.2.4.

In practice, different nodes may use different effective diameters d_i instead of a uniform l. Similarly, each node i may use different l_i instead of a single l (in Algorithm 3.2.4). We will discuss more on these parameters in Section 3.3, where we present the integrated distributed protocol without the separation of stages.

Note that at each outer iteration Algorithm 3.2.4 may not always elect a feedback node. Figure 3.9 shows an illustrating example. Assume that node 2 has the highest priority score 0.8 among all nodes and that the effective diameter $d = 5$. At the end of the first outer iteration, each node i will have $\mathrm{MaxScore}(i) = 0.8$ and $\mathrm{MaxId}(i) = 2$, but node 2 is not elected as as a feedback node since its status has changed to T. However, for every two outer iterations, at least one feedback node will be elected. will be elected so that the total number of elected feedback nodes is between $\lfloor \frac{l}{2} \rfloor$ and l. We defer the statement of this theoretical result and its proof to Section 3.4.

■ 3.2.2 Stage II: Initial Estimation

At the beginning of second stage, each node knows its own status, but in general it does not know the status of other nodes.[10] Every node keeps a list of inactive nodes (called a *priority list* and denoted as $\mathcal{L}_i$ at node i) with the corresponding priority scores. As we mentioned before, initially the set of the inactive nodes is the set of elected feedback

[10]Although in theory more information about the statuses of other nodes can be inferred from the messages, here we do not collect the extra information.

Algorithm 3.2.4 Message Protocol for Stage I: Election of Feedback Nodes

Input: graphical model with information matrix J

Output: status S_i for each $i \in \mathcal{V}$

At each node i with status U, simultaneously

1. Outer iterations: repeat for l times

   (a) Compute $p_i = \sum_{j \in \mathcal{N}_{\mathrm{U}}(i)} |J_{ij}|$. Set

   $$\mathrm{MaxScore}(i) \leftarrow p_i \text{ and } \mathrm{MaxId}(i) \leftarrow i.$$

   (b) Inner iterations: for $t = 1, 2, \ldots, d_i$:

      i. If $|\mathcal{N}_{\mathrm{U}}(i)| \leq 1$, then $S_i \leftarrow \mathrm{T}$.

      ii. Update local values using:

      $$\mathrm{MaxScore}^{(t)}(i) = \max_{j \in \mathcal{N}_{\mathrm{U}}(i) \cup \{i\}} \mathrm{MaxScore}^{(t-1)}(j)$$
      $$\mathrm{MaxId}^{(t)}(i) = j^*,$$

      where $j^* = \arg\max_{j \in \mathcal{N}_{\mathrm{U}}(i) \cup \{i\}} \mathrm{MaxScore}^{(t-1)}(j)$.

      iii. $S_i \leftarrow \mathrm{F}$ if $i = \mathrm{MaxId}(i)$.

   (c) $S_i \leftarrow \mathrm{F}$ if $i = \mathrm{MaxId}(i)$.

2. If $S_i = \mathrm{U}$, then $S_i \leftarrow \mathrm{T}$

At each node i with status T or F,
No action.

nodes and the set of the active nodes is the set of the non-feedback nodes. The nodes in a priority list are sorted by their priority scores in descending order. Each list $\mathcal{L}_i$ has maximum capacity K_i and can only keep the nodes with the top-K_i priority scores if the capacity is exceeded. In the following, for simplicity of notation, we use the notation $\lceil \mathcal{L}_i \rceil^{K_i}$ to denote the truncation corresponding to keeping only the nodes with the top-K_i priority scores in list $\mathcal{L}_i$. At the beginning of Stage II, at an active node i, we initiate $\mathcal{L}_i$ by including all neighboring inactive nodes, i.e.,

$$\mathcal{L}_i^{(0)} = \lceil \mathcal{N}_{\mathrm{F}}(i) \rceil^{K_i} \tag{3.1}$$

as well as their corresponding scores. At an inactive node q, the list $\mathcal{L}_q^{(0)}$ is initiated by including q itself and all other neighboring inactive nodes, i.e.,

$$\mathcal{L}_q^{(0)} = \lceil \{q\} \cup \mathcal{N}_{\mathrm{F}}(q) \rceil^{K_q} \tag{3.2}$$

as well as their corresponding scores. Figure 3.3 gives an example of initializing the priority lists: node 2 has status F and among its neighbors only node 1 has status F, so its initial priority list $\mathcal{L}_2^{(0)} = \{2, 1\}$ with corresponding scores 0.3 and 0.2. Node 4 has status T and among its neighbors only node 2 has status F, so its initial priority list is $\mathcal{L}_4^{(0)} = \{2\}$ with score 0.3.

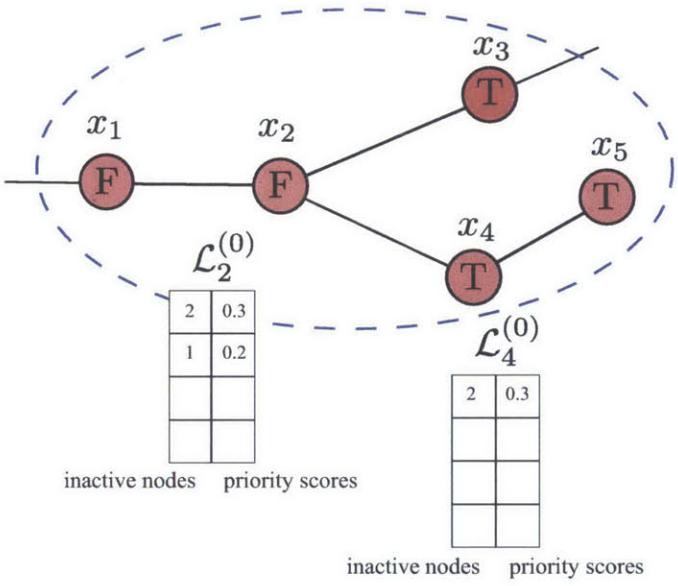


Figure 3.3: Priority lists at the start of Stage II

In the second stage, partial estimates of the means and variances for the active nodes, as well as the "feedback gains"[11] are computed by passing messages only among the active nodes. The partial estimates are estimates of variances and means only on the active subgraph without considering any edges involving the inactive nodes. Hence, the protocol for computing the partial estimates is simply standard LBP. The feedback

[11]The "feedback gains" here directly correspond to the feedback gains in the standard FMP algorithm (c.f. Section 2.2.3 and [16]).

gains are intermediate results characterizing the effect of the inactive nodes on the exact variances and means of the active nodes. As shown in [16] and in Section 2.2.3, the messages for computing the feedback gains are essentially LBP messages with carefully designed new potential vectors, each corresponding to an inactive node. The major difference from the protocol in standard FMP (Algorithm 2.2.2) is that here each active node initially does not know which nodes are the inactive nodes and can only pass feedback gain messages after they receive such information.[12] In this stage, the inactive nodes do not participate in message passing, but merely monitor the priority lists at their neighbors (except under some conditions to be explained, an inactive node may convert to an active node and then follow the message protocol for active nodes). In the following, for clarity, we use the superscript $^{(t)}$ to denote the values (e.g., messages or locally stored values) at iteration t.[13]

Message Protocol at an active node First, we describe the message-update rules used at an active node. The messages being passed include standard LBP messages and messages for computing the feedback gains.

(a) **Standard LBP Messages**

Each active node i sends LBP messages $\Delta J_{i\to j}$ and $\Delta h_{i\to j}$ to each of its active neighbor j, which corresponds to computing the partial variances and the partial means respectively. Σ_{ii} and μ_i are locally stored values representing the current estimates of the variance and mean of node i respectively.

The messages are updated using

$$\Delta J_{i\to j}^{(t)} = -J_{ji}\hat{J}_{i\backslash j}^{(t-1)}J_{ij} \tag{3.3}$$

$$\Delta h_{i\to j}^{(t)} = -J_{ji}\hat{J}_{i\backslash j}^{(t-1)}\hat{h}_{i\backslash j}^{(t-1)}, \tag{3.4}$$

where

$$\hat{J}_{i\backslash j}^{(t-1)} = J_{ii} + \sum_{u\in\mathcal{N}_{\mathrm{T}}(i)\backslash\{j\}} \Delta J_{u\to i}^{(t-1)} \tag{3.5}$$

[12] As will be explained later, the priority list at an active node will expand or change upon receiving feedback gain messages.

[13] Note that the values with different superscripts use the same storage space, i.e., they overwrite previous values.

$$\hat{h}_{i\backslash j}^{(t-1)} = h_i + \sum_{u \in \mathcal{N}_{\mathrm{T}}(i)\backslash\{j\}} \Delta h_{u\to i}^{(t-1)} \tag{3.6}$$

and all messages have initial value zero. The local values Σ_{ii} and μ_i are updated using

$$\Sigma_{ii}^{(t)} = \left(J_{ii} + \sum_{j \in \mathcal{N}_{\mathrm{T}}(i)} \Delta J_{j\to i}^{(t-1)} \right)^{-1} \tag{3.7}$$

$$\mu_i^{(t)} = \Sigma_{ii}^{(t)} \left(h_i + \sum_{j \in \mathcal{N}_{\mathrm{T}}(i)} \Delta h_{j\to i}^{(t-1)} \right). \tag{3.8}$$

Note that while we have not been explicit about this previously, each of these messages has protocol bits indicating both the identity of the sending node as well as the fact that it is active.

(b) **Messages for Feedback Gains**

At each active node i, feedback messages $\Delta G_{i\to j} \triangleq \{\Delta g_{i\to j}^q\}_{q\in\mathcal{L}_i}$ are sent to each active neighbor j. Each individual message $\Delta g_{i\to j}^q$ can be viewed as an LBP message for the mean (similar to $\Delta h_{i\to j}$) but using an auxiliary potential vector $\mathbf{h}^q$ corresponding to the feedback node q. The entries of this auxiliary potential vector are

$$(\mathbf{h}^q)_i = \begin{cases} 0 & \forall i \notin \mathcal{N}(q) \\ J_{iq} & \forall i \in \mathcal{N}(q). \end{cases} \tag{3.9}$$

Hence, $\mathbf{h}^q$ can be constructed locally by using the default value zero for nodes that are not neighbors of node q and use local parameters J_{iq} for the nodes that are. Note that each of these messages has protocol bits indicating the corresponding inactive node as well as its score (and also which active node is sending the message).

Note that at an active node i, the list $\mathcal{L}_i$ may expand or change when node i receives messages corresponding to other inactive nodes that are not its neighbors.[14]

At each iteration at an active node i, before sending out messages, the priority list $\mathcal{L}_i$ is updated using

$$\mathcal{L}_i^{(t)} = \left\lceil \mathcal{L}_i^{(t-1)} \cup \left(\cup_{j\in\mathcal{N}_{\mathrm{T}}(i)} \mathcal{L}_j^{(t-1)} \right) \right\rceil^{K_i}. \tag{3.10}$$

[14]Some active nodes may not be neighbors of any inactive node, so they initially do not pass feedback gain messages until they receive them.

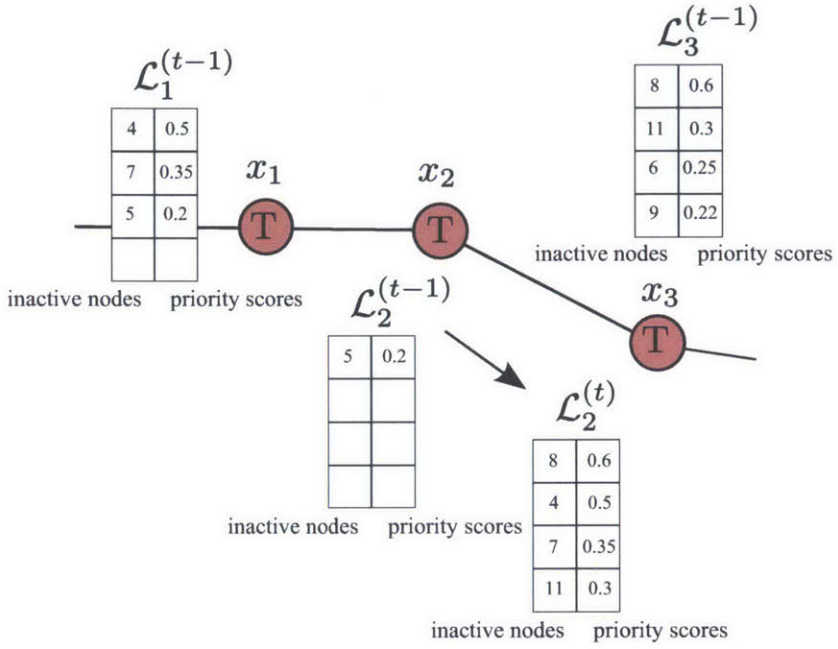


Figure 3.4: Updating priority list at an active node

Figure 3.4 provides an illustrating example. At iteration $t-1$, the priority lists of nodes 1, 2 and 3 are each sorted by the priority scores. At iteration t, the priority list at node 2 takes the union and is then truncated to satisfy the maximum capacity 4.

Then for each $q \in \mathcal{L}_i^{(t)}$, node i sends messages to each active neighbor j using

$$\left(\Delta g_{i\to j}^q\right)^{(t)} = -J_{ji}\left(\hat{J}_{i\backslash j}^{(t-1)}\right)^{-1}\left(\hat{g}_{i\backslash j}^q\right)^{(t-1)}, \tag{3.11}$$

where

$$\left(\hat{g}_{i\backslash j}^q\right)^{(t-1)} = (\mathbf{h}^q)_i + \sum_{u\in\mathcal{N}_{\mathrm{T}}(i)\backslash\{j\}} \left(\Delta g_{u\to i}^q\right)^{(t-1)} \tag{3.12}$$

and $\left(\Delta g_{u\to i}^q\right)^{(t-1)} = 0$ for $q \notin \mathcal{L}_u^{(t-1)}$, i.e., the incoming messages have default value zero.

In addition to the messages, feedback gains $G_i = \{g_i^q\}_{q\in\mathcal{L}_i}$ are locally stored, where each g_i^q is updated using

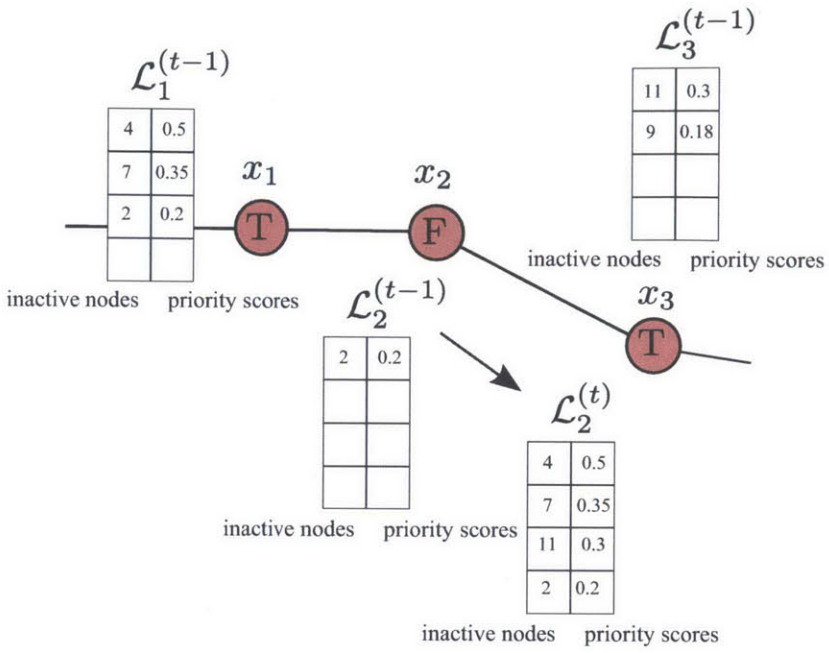


Figure 3.5: Updating priority list at an inactive node

$$(g_i^q)^{(t)} = \Sigma_{ii}^{(t)} \left((\mathbf{h}^q)_i + \sum_{j \in \mathcal{N}_{\mathrm{T}}(i)} \left(\Delta g_{j \to i}^q \right)^{(t-1)} \right) \tag{3.13}$$

and $\left(\Delta g_{j \to i}^q \right)^{(t-1)} = 0$ for $q \notin \mathcal{L}_j^{(t-1)}$.

Actions at an inactive node Each inactive node monitors the priority lists of its active neighbors. At an inactive node q, its priority list $\mathcal{L}_q$ is updated using

$$\mathcal{L}_q^{(t)} = \left\lceil \mathcal{L}_q^{(t-1)} \cup \left(\cup_{j \in \mathcal{N}_{\mathrm{T}}(q)} \mathcal{L}_j^{(t-1)} \right) \right\rceil^{K_q}. \tag{3.14}$$

Figure 3.5 provides an illustrating example where the priority list of the inactive node 2 expands after taking the union and is truncated to keep 4 nodes.

After the priority list $\mathcal{L}_q$ is updated, if it does not include q itself, i.e., $q \notin \mathcal{L}_i$, then node q converts to an active node:

$$S_q \leftarrow \mathrm{T}. \tag{3.15}$$

Algorithm 3.2.5 Message Protocol for Stage II: Initial Estimation

At each active node i, simultaneously

Initialization: $\mathcal{L}_i^{(0)} = \lceil \mathcal{N}_{\mathrm{F}}(i) \rceil^{K_i}$

At each iteration t until convergence or timeout,

1. Send LBP messages to each $j \in \mathcal{N}_{\mathrm{T}}(i)$:

$$\Delta J_{i\to j}^{(t)} = -J_{ji}\hat{J}_{i\backslash j}^{(t-1)}J_{ij}$$
$$\Delta h_{i\to j}^{(t)} = -J_{ji}\hat{J}_{i\backslash j}^{(t-1)}\hat{h}_{i\backslash j}^{(t-1)},$$

   where $\hat{J}_{i\backslash j}^{(t-1)} = J_{ii} + \sum_{u\in\mathcal{N}_{\mathrm{T}}(i)\backslash\{j\}} \Delta J_{u\to i}^{(t-1)}$ and $\hat{h}_{i\backslash j}^{(t-1)} = h_i + \Sigma_{u\in\mathcal{N}_{\mathrm{T}}(i)\backslash\{j\}}\Delta h_{u\to i}^{(t-1)}$.

2. Update priority list: $\mathcal{L}_i^{(t)} = \left\lceil \mathcal{L}_i^{(t-1)} \cup \left(\cup_{j\in\mathcal{N}_{\mathrm{T}}(i)} \mathcal{L}_j^{(t-1)} \right) \right\rceil^{K_i}$.

3. For all $q \in \mathcal{L}_i^{(t)}$ and all $j \in \mathcal{N}_{\mathrm{T}}(i)$, send messages

$$\left(\Delta g_{i\to j}^{q}\right)^{(t)} = -J_{ji}\left(\hat{J}_{i\backslash j}^{(t-1)}\right)^{-1}\left(\hat{g}_{i\backslash j}^{q}\right)^{(t-1)},$$

   where $\left(\hat{g}_{i\backslash j}^{q}\right)^{(t-1)} = (\mathbf{h}^q)_i + \sum_{u\in\mathcal{N}_{\mathrm{T}}(i)\backslash\{j\}} \left(\Delta g_{u\to i}^{q}\right)^{(t-1)}$ and $\left(\Delta g_{u\to i}^{q}\right)^{(t-1)} = 0$ if $q \notin \mathcal{L}_u^{(t-1)}$.

4. Update the local values by

$$\Sigma_{ii}^{(t)} = \left(J_{ii} + \sum_{j\in\mathcal{N}_{\mathrm{T}}(i)} \Delta J_{j\to i}^{(t)} \right)^{-1}, \quad \mu_i^{(t)} = \Sigma_{ii}^{(t)} \left(h_i + \sum_{j\in\mathcal{N}_{\mathrm{T}}(i)} \Delta h_{j\to i}^{(t)} \right)$$

$$(g_i^q)^{(t)} = \Sigma_{ii}^{(t)} \left((\mathbf{h}^q)_i + \sum_{j\in\mathcal{N}_{\mathrm{T}}(i)} \left(\Delta g_{j\to i}^{q}\right)^{(t-1)} \right) \text{ for } q \in \mathcal{L}_i^{(t)},$$

   where $\left(\Delta g_{j\to i}^{q}\right)^{(t-1)} = 0$ for $q \notin \mathcal{L}_j^{(t-1)}$

At each inactive node q, simultaneously

Initialization: $\mathcal{L}_q^{(0)} = \lceil \{q\} \cup \mathcal{N}_{\mathrm{F}}(q) \rceil^{K_q}$

At each iteration t until entering Stage III,

1. Update priority list $\mathcal{L}_q^{(t)} = \left\lceil \mathcal{L}_q^{(t-1)} \cup \left(\cup_{u\in\mathcal{N}_{\mathrm{T}}(q)} \mathcal{L}_u^{(t-1)} \right) \right\rceil^{K_q}$.

2. If $q \notin \mathcal{L}_q^{(t)}$, then $S_q \leftarrow T$.

Such a conversion happens when an inactive node receives too many feedback messages about other inactive nodes with higher scores so that it is removed from its own priority list. We can view it as a mistakenly elected inactive node returning to the active status when later proved under-qualified.

We summarize the whole message-passing protocol used in the second stage in Algorithm 3.2.5. The iterations (as indicated by t) stop when the messages have converged[15] or the maximum number of iterations allowed has reached (called *timeout*), which is usually set so that LBP have converged.

3.2.3 Stage III: Recursive Correction

In the third stage, each of the still remaining inactive nodes "wakes up" and converts to an active node when certain local conditions are satisfied. Before converting to an active node, the node initiates a new kind of messages (called *correction messages*) which are then passed among the active nodes to correct the partial estimates obtained in the second stage. This roughly corresponds to Step 3 of the standard FMP algorithm, although here it is done node-by-node as nodes wake up. At the same time, each of the active nodes makes corrections to their locally stored values and then relays the correction messages to neighboring active nodes without modification. In the following, we describe the message-passing protocol at the inactive nodes and at the active nodes respectively.

At an inactive node q: An inactive node q wakes up when both of the following conditions are satisfied.

1. Node q itself has the lowest priority score in $\mathcal{L}_q$.
2. The locally stored values (e.g., partial means, partial variances, and feedback gains) at all of its active neighbors have converged.[16]

Figure 3.6 provides an illustrating example of the local conditions: inactive node 2 has the lowest priority score in its own priority list. If the local values at its neighbors have converged, node 2 will initiate the correction messages and then change its status to T.

[15]There are various ways to quantify convergence. We can use for example the relative changes of the values between two recent iterations

[16]We let the convergence information be passed to the neighboring inactive nodes so that they know.

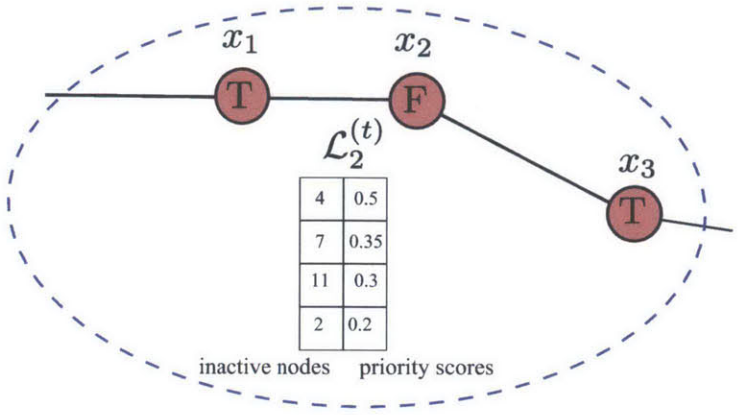


Figure 3.6: An inactive node waking up

After the conditions are satisfied, node q will do the following:

1. Delete itself from its own priority list: $\mathcal{L}_q^{(t)} = \mathcal{L}_q^{(t-1)} \backslash \{q\}$.

2. Compute the current estimates of the variance $\Sigma_{qq}^{(t)}$, the mean $\mu_q^{(t)}$, and the feedback gains $G_q^{(t)} = \{(g_q^p)^{(t)}\}_{p \in \mathcal{L}_q^{(t)}}$ using

$$\Sigma_{qq}^{(t)} = \left(J_{qq} - \sum_{j \in \mathcal{N}_{\mathrm{T}}(q)} J_{qj} \left(g_j^q \right)^{(t-1)} \right)^{-1} \tag{3.16}$$

$$\mu_q^{(t)} = \Sigma_{qq}^{(t)} \left(h_q - \sum_{j \in \mathcal{N}_{\mathrm{T}}(q)} J_{qj} \left(\mu_j \right)^{(t-1)} \right) \tag{3.17}$$

$$\left(g_q^p \right)^{(t)} = \Sigma_{qq}^{(t)} \left(\left(\mathbf{h}^p \right)_q - \sum_{j \in \mathcal{N}_{\mathrm{T}}(q)} J_{qj} \left(g_j^p \right)^{(t-1)} \right) \text{ for } p \in \mathcal{L}_q^{(t)}, \tag{3.18}$$

   where $\left(g_j^p \right)^{(t-1)} = 0$ if $p \notin \mathcal{L}_j^{(t)}$.

3. Forward the correction message $C_q = \{q, \Sigma_{qq}, \mu_q, G_q, \mathcal{L}_q\}$ to its active neighbors.[17]

4. Change status to T.

[17] Here we intentionally do not use superscripts to emphasize that the correction messages are sent once after all neighboring active nodes have convergence, and these messages are not sent again by the waking node.

At an active node i: When an active node i receives the correction message $C_q = \{q, \Sigma_{qq}, \mu_q, G_q, \mathcal{L}_q\}$ from some neighbor, and if that correction corresponds to an inactive node on its list, it corrects its local values accordingly and then relays the correction messages to other active neighbors. The following steps describe the protocol.

1. If $q \notin \mathcal{L}_i^{(t)}$, take no action.
2. If $q \in \mathcal{L}_i^{(t)}$, then
   (a) Update its priority list by

$$\mathcal{L}_i^{(t)} = \left[\mathcal{L}_q \cup \mathcal{L}_i^{(t-1)} \backslash \{q\}\right]^{K_i}. \tag{3.19}$$

   (b) Update the local values by

$$\Sigma_{ii}^{(t)} = \Sigma_{ii}^{(t-1)} + \left((g_i^q)^{(t-1)}\right)^2 \Sigma_{qq} \tag{3.20}$$

$$\mu_i^{(t)} = \mu_i^{(t-1)} - (g_i^q)^{(t-1)} \mu_q \tag{3.21}$$

$$(g_i^p)^{(t)} = (g_i^p)^{(t-1)} - (g_i^q)^{(t-1)} g_q^p \text{ for } p \in \mathcal{L}_i^{(t)}. \tag{3.22}$$

   (c) Forward the correction message C_q to its active neighbors.

At the end of the third stage, all nodes are active and all local values are updated to account for the inference results for the entire graph. We obtain the variance i and mean μ_i for all $i \in \mathcal{V}$.

In Algorithm 3.2.6, we summarize the whole protocol used in this stage. We provide an illustrating example of running recursive FMP on a 4-by-4 grid in Figure 3.7. The theoretical results on the correctness of the whole recursive FMP algorithm are postponed to Section 3.4. Those results show that this algorithm gives the same result as the FMP algorithm in Section 2.2.3 under certain conditions. However, we emphasize that our distributed algorithm allows much more flexibility, as different parts of the graph may use different subsets of feedback nodes and that nodes can dynamically change the set of feedback nodes to which they will pay attention. Our integrated protocol in the next section will make this clear, and we demonstrate the performance of this distributed algorithm for a very large graph in Section 3.5.

Algorithm 3.2.6 Message Protocol for Stage III: Recursive Correction

At an inactive node q:

At each iteration t, when the following two conditions are satisfied: 1) q itself has the lowest priority score in $\mathcal{L}_q$; and 2) local values at its active neighbors have converged, then

1. $\mathcal{L}_q^{(t)} = \mathcal{L}_q^{(t-1)} \backslash \{q\}$.

2. Compute the current estimates of the variance, the mean, and the feedback gains at q using

$$\Sigma_{qq}^{(t)} = \left(J_{qq} - \sum_{j \in \mathcal{N}_{\mathrm{T}}(q)} J_{qj} \left(g_j^q \right)^{(t-1)} \right)^{-1}$$

$$\mu_q^{(t)} = \Sigma_{qq}^{(t)} \left(h_q - \sum_{j \in \mathcal{N}_{\mathrm{T}}(q)} J_{qj} \mu_j^{(t-1)} \right)$$

$$\left(g_q^p \right)^{(t)} = \Sigma_{qq}^{(t)} \left((\mathbf{h}^p)_q - \sum_{j \in \mathcal{N}_{\mathrm{T}}(q)} J_{qj} \left(g_j^p \right)^{(t-1)} \right) \text{ for } p \in \mathcal{L}_q^{(t)},$$

   where $\left(g_j^p \right)^{(t-1)} = 0$ if $p \notin (\mathcal{L}_j)^{(t-1)}$.

3. Send correction message C_q including Σ_{qq}, μ_q, and g_q^p, $\forall p \in \mathcal{L}_q$ to active neighbors.

4. $S_q \leftarrow \mathrm{T}$

At an active node i:

When correction messages corresponding to node q are received, then

1. Update the priority list by $\mathcal{L}_i^{(t)} = \left| \mathcal{L}_q \cup \mathcal{L}_i^{(t-1)} \backslash \{q\} \right|^{K_i}$.

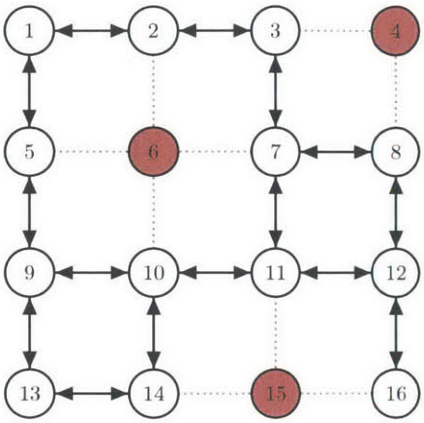
2. Update the local values by

$$\Sigma_{ii}^{(t)} = \Sigma_{ii}^{(t-1)} + \left((g_i^q)^{(t-1)} \right)^2 \Sigma_{qq}$$

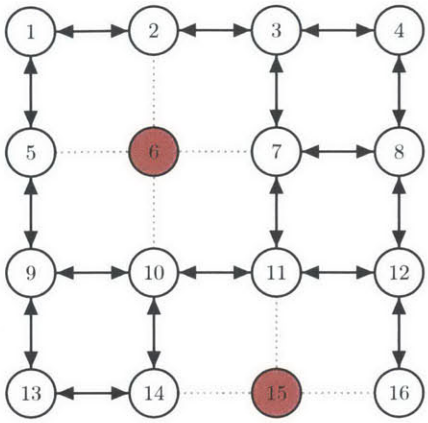
$$\mu_i^{(t)} = \mu_i^{(t-1)} - (g_i^q)^{(t-1)} \mu_q$$

$$(g_i^p)^{(t)} = (g_i^p)^{(t-1)} - (g_i^q)^{(t-1)} g_q^p, \text{ for } p \in \mathcal{L}_i^{(t)}.$$

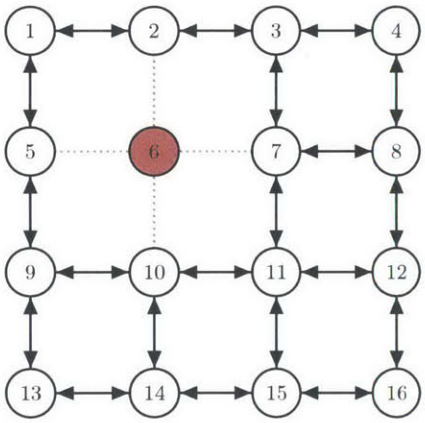
3. Forward the correction message C_q to other active neighbors.



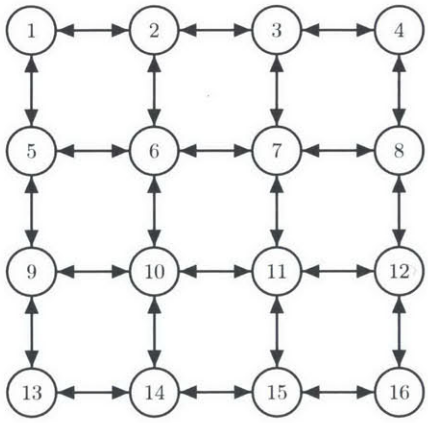
(a) Stage II: nodes 4, 6, and 15 have been elected as inactive nodes. Messages are passed among other nodes.



(b) Stage III: node 4 has become active again. Correction messages from node 4 are being forwarded to other active nodes.



(c) Stage III: node 15 has become active again. Correction messages from node 15 are being forwarded to other active nodes.



(d) Stage III: all nodes are active nodes now. Final correction messages from node 6 are being forwarded.

Figure 3.7: Stage II (a) and Stage III (b–d) of recursive FMP. Shaded nodes represent elected inactive nodes. Solid lines with arrows denote the edges where messages are being passed.

3.3 Recursive FMP: Integrated Message-Passing Protocol

In Section 3.2, we have described the recursive FMP algorithm in three separate stages (election of feedback nodes, initial estimation, and recursive correction), where we assume that all nodes move to the next stage at the same time. However, this requires a global communication protocol to signal when *all* the nodes are ready to move to the next stage. Instead, we can avoid this completely in an integrated protocol in which some nodes may move to the next stage earlier than others.

Specifically, in Stage I, once a node converts to status T or F, it does not have to wait and can in fact immediately move to Stage II. This "early" stage transition not only removes the need to synchronize the stages of all nodes, but also starts LBP message passing earlier so that we can reach some local convergence faster than if we let the nodes with status T just wait and stay in Stage I. Similarly in Stage II, some inactive nodes can wake up before LBP on the entire active graph converges, which is beneficial when different regions on the graph have different convergence rates or the active subgraph has several connected components. Furthermore, when the algorithm parameters d_i, l_i and K_i are different at different nodes $i \in \mathcal{V}$, an integrated protocol is not only more efficient (by reducing the waiting time) but also is more natural to think about.

In Figure 3.8 we show the flowchart of our integrated protocol without the global separation of stages. We use the following color and shape scheme to indicate components with different categories of functionality. The two yellow rectangles with rounded corners (A1 and C9) are the start unit and end unit of the algorithm at each node respectively. The diamond-shaped components in light blue (A4, A7, A8, A10, B5, B10, B11, C5, and C8) are decision units where the next steps depend on whether the testing conditions are satisfied. The yellow rectangles with angular corners (A5, B3, and C7) are where messages are passed to or from neighbors. The rectangles in red with angular corners (B1, B6, and C3) are where node status changes. The green rectangular components (A6, B4, B9, and C4) denote "clock sync", which are used to indicate how the operations at different nodes are synchronized. Only for the purpose of understanding the synchronization, we can interprete the flowchart as whenever the "clock sync" is passed through, we increment one time unit while other operations are completed "instantaneously". Of course in practice, we only require that all operations between two "clock sync" are completed within one time unit. The components A1–A10 can be viewed as procedures in Stage I; the components B1–B11 can be viewed as procedures

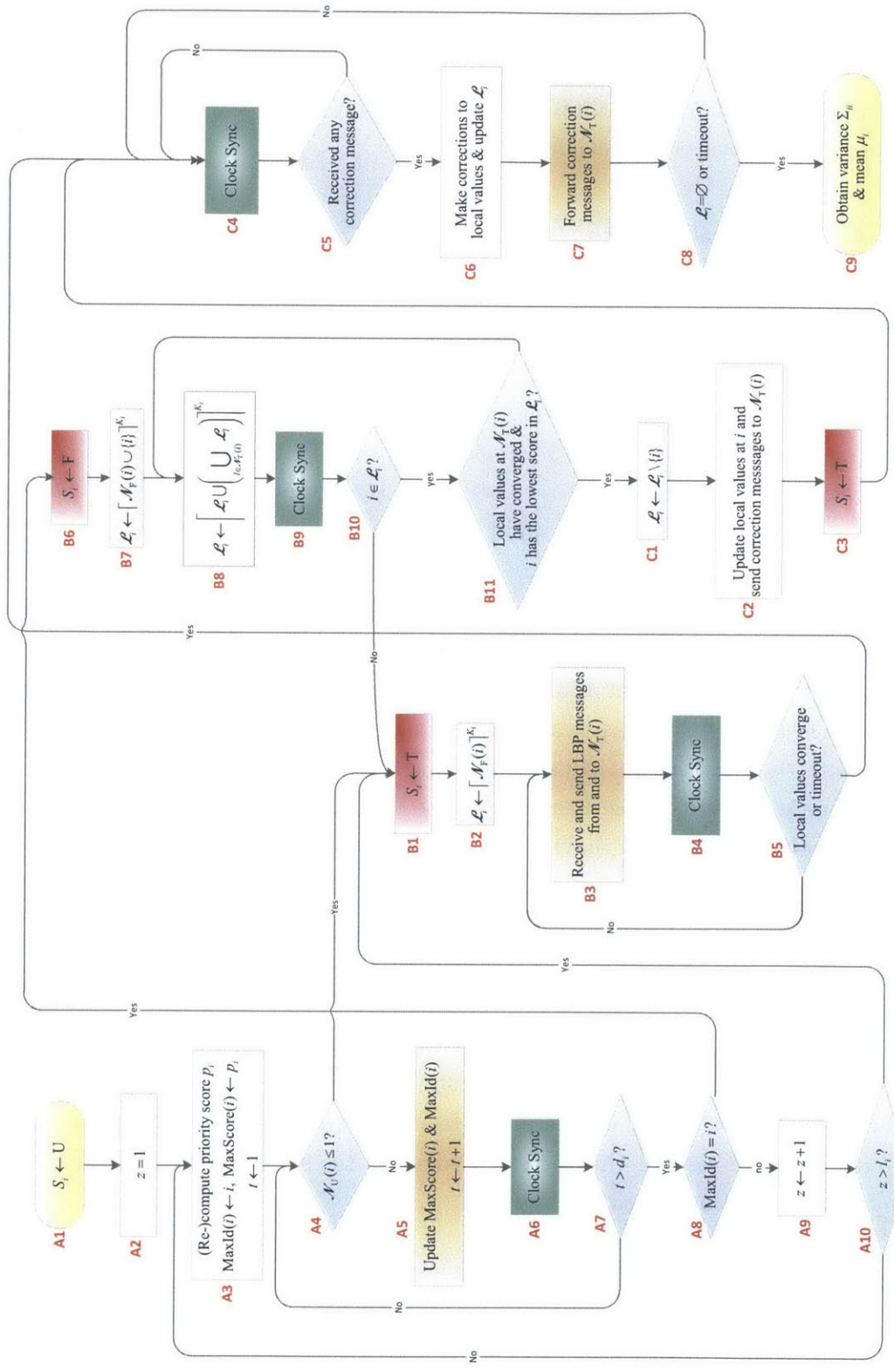


Figure 3.8: Recursive FMP as an integrated protocol

in Stage II; and the components C1–C9 can be viewed as procedures in Stage III.

Finally, we note that at each node i the memory cost is $\mathcal{O}(K_i)$ and the communication cost per time unit is $\mathcal{O}(K_i)$ for each outgoing link from i and $\mathcal{O}(K_j)$ for each incoming link from node i's neighbor j.

3.4 Theoretical Results

In this section, we present and prove some theoretical results on the convergence and accuracy of recursive FMP. The theoretical results are easier to state and comprehend under the framework with separate stages in Section 3.2. However, our conclusions still apply in the integrated framework in Section 3.3. First in Proposition 3.4.2, we state the conditions under which recursive FMP elects the same feedback nodes as the centralized greedy algorithm (Algorithm 2.2.1). Then we present Proposition 3.4.5 which states that if the elected feedback nodes form a full FVS, then under milder conditions the inference results obtained by recursive FMP are *exact*. Next in Proposition 3.4.8, we provide a new walk-sum interpretation of the correction messages, which further leads to a walk-sum decomposition of the final inference results. Finally, we conclude that even when the set of the elected FVS is not a full FVS, under some conditions the inference results by recursive FMP are still consistent with running the hybrid FMP algorithm (Algorithm 2.2.2) with the same set of feedback nodes.

While these results make clear the precise connection to the much more centralized FMP algorithm developed previously and reviewed in Section 2.2.3, it is crucial to note that for large graphs, the conditions in the following results—e.g., conditions on the effective diameters and capacities of nodes—may not hold, and, in fact, we may not want them to hold. In that very important sense, the recursive FMP algorithm is fundamentally a richer and, as we demonstrate via examples in Section 3.5, an extremely effective algorithm, truly generalizing LBP and applicable to very large graphs.

For a fixed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and any set $A \subset \mathcal{V}$, we use $\mathcal{G}_A$ to denote the subgraph induced by the set A and we use $d(\mathcal{G}_A)$ to denote the diameter of the graph $\mathcal{G}_A$. We reiterate that throughout this section, F denotes the set of elected feedback nodes at the end of Stage I and T equals $\mathcal{V} \backslash F$.

Assumption 3.4.1 : *1. All nodes use the same $l_i = l$.*

2. $\mathcal{G}_T$ is connected.

3. *For every* $i \in \mathcal{V}$, $d_i \geq d(\mathcal{G}_T) + |F|$.

4. *There are no ties in priority scores or ties are broken by a fixed rule.*

Proposition 3.4.2 : *Under Assumption 3.4.1, there exists an integer* k_0 *between* $\lfloor \frac{l}{2} \rfloor$ *and* l *such that the set of feedback nodes elected by recursive FMP is the same set obtained by running the centralized selection algorithm (Algorithm 2.2.1) with* k_0 *iterations.*

Proof. The set T can be partitioned into two sets as

$$T = T_a \cup T_b, \tag{3.23}$$

where T_a denotes the set of nodes marked as status T when they were on some tree branches (c.f. Step 1(b)i in Algorithm 3.2.4) and T_b denotes the set of nodes converted to status T from U at the end of the algorithm (c.f. Step 2 in Algorithm 3.2.4). In order to distinguish the node status at different outer iterations, we use $U^{(m)}$ to denote the set of nodes with status U at the end of the m-th outer iteration (i.e., right after running Step 1(c) in Algorithm 3.2.4) for $m = 0, 1, 2, \ldots, l$. In particular,

$$U^{(l)} = T_b \tag{3.24}$$

$$U^{(0)} = \mathcal{V}. \tag{3.25}$$

Furthermore, since nodes with status T or F never convert back to have status U again, we have

$$\mathcal{V} = U^{(0)} \supset U^{(1)} \supset U^{(2)} \supset \cdots \supset U^{(l)} = T_b. \tag{3.26}$$

First, we prove that the subgraph $\mathcal{G}_{U^{(l)}}$ is connected. Let

$$T_a = T^{(1)} \cup T^{(2)} \cup \cdots \cup T^{(l)}, \tag{3.27}$$

where $T^{(m)}$ denotes the set of nodes converted to status T in the m-th outer iteration, i.e., all nodes in $T^{(m)}$ is on some tree branch of $\mathcal{G}_{U^{(m-1)}}$. Then we have

$$U^{(0)} \supset U^{(1)} \cup T^{(1)} \supset U^{(2)} \cup T^{(2)} \cup T^{(1)} \supset \cdots \supset U^{(m)} \cup \left(\cup_{s=1}^{m} T^{(s)} \right) \cdots \supset U^{(l)} \cup \left(\cup_{s=1}^{l} T^{(s)} \right) \tag{3.28}$$

In particular, for $m = 0, 1 \ldots, l-1$, we have

$$U^{(m)} \cup \left(\cup_{s=1}^{m} T^{(s)}\right) \supset U^{(l)} \cup \left(\cup_{s=1}^{l} T^{(s)}\right) \tag{3.29}$$

and thus

$$U^{(m)} \supset U^{(l)} \cup \left(\cup_{s=m+1}^{l} T^{(s)}\right). \tag{3.30}$$

According to Lemma 3.4.3 to follow, for $m = 0, 1, \ldots l-1$, we have that nodes in $T^{(m+1)}$ are on tree branches of $\mathcal{G}_{U^{(l)} \cup \left(\cup_{s=m+1}^{l} T^{(s)}\right)}$ because nodes in $T^{(m+1)}$ are on tree branches of $\mathcal{G}_{U^{(m)}}$ by definition and $U^{(m)} \supset U^{(l)} \cup \left(\cup_{s=m+1}^{l} T^{(s)}\right) \supset T^{(m+1)}$. Hence, we can clean $T^{(1)}$, $T^{(2)}$,..., $T^{(l)}$ in order from the connected graph $\mathcal{G}_T$, or $\mathcal{G}_{U^{(l)} \cup \left(\cup_{s=1}^{l} T^{(s)}\right)}$ (where the clean-up of $T^{(s)}$ is from $\mathcal{G}_{U^{(l)} \cup \left(\cup_{s=m+1}^{l} T^{(s)}\right)}$). Hence, all nodes in $T_a = \cup_{s=1}^{l} T^{(s)}$ are on tree branches of $\mathcal{G}_T$. In addition, we have that $\mathcal{G}_{U^{(l)} \cup \left(\cup_{s=m+1}^{l} T^{(s)}\right)}$ is connected for each $m = 0, 1, \ldots, l$ because the elimination of tree branches does not break a connected graph (c.f. Lemma 3.4.3 to follow). In particular, $\mathcal{G}_{U^{(l)}}$ is connected.

Second, we prove that $\mathcal{G}_{U^{(m)}}$ is connected for all $m = 0, 1, \ldots, l$. We use $F^{(m)}$ to denote the set of feedback nodes elected in the m-th outer iteration (c.f. Step 1(c) of Algorithm 3.2.4) for $m = 1, 2, \ldots l$. Then we have $U^{(m-1)} = U^{(m)} \cup T^{(m)} \cup F^{(m)}$. By Step 1(c) in Algorithm 3.2.4, $\mathcal{G}_{F^{(m)}}$ cannot have any edges because otherwise different nodes in $F^{(m)}$ cannot be elected in the same outer iteration since we assume no ties in priority scores or fixed tie-breaker. Hence, any node in $F^{(m)}$ is connected to at least two nodes in $U^{(m)} \cup T^{(m)}$ because otherwise they will have status T by Step 1(b)i in Algorithm 3.2.4. In particular, $F^{(l)}$ is connected to $U^{(l)} \cup T^{(l)}$. Since we have proved that $\mathcal{G}_{U^{(l)} \cup T^{(l)}}$ is connected, we have that the subgraph $\mathcal{G}_{U^{(l-1)}}$ is also connected from $U^{(l)} = U^{(l)} \cup T^{(l)} \cup F^{(l)}$. By repeating this process, we have that $U^{(m)}$ is connected for each $m = 0, 1, \ldots, l$.

Third, we prove that $d_i \geq d(\mathcal{G}_{U^{(m)}})$ for all $m = 0, 1, \ldots, l$ and all $i \in \mathcal{V}$. We have that

$$U^{(m)} = U^{(l)} \cup \left(\cup_{s=m+1}^{(l)} T^{(s)}\right) \cup \left(\cup_{s=m+1}^{(l)} F^{(s)}\right) \tag{3.31}$$

by definition. Since $\mathcal{G}_{U^{(m)}}$ is connected and $\mathcal{G}_{U^{(l)} \cup \left(\cup_{s=m+1}^{(l)} T^{(s)}\right)}$ is connected, we have that

$$d(\mathcal{G}_{U^{(m)}}) \leq d(\mathcal{G}_{U^{(l)} \cup \left(\cup_{s=m+1}^{(l)} T^{(s)}\right)}) + |\cup_{s=m+1}^{(l)} F^{(s)}|. \tag{3.32}$$

Since $T^{(1)}$, $T^{(2)}$, ..., $T^{(m)}$ are tree branches of $T = U^{(l)} \cup \left(\cup_{s=1}^{(l)} T^{(s)}\right)$, we have that

$$d(\mathcal{G}_{U^{(l)} \cup \left(\cup_{s=m+1}^{(l)} T^{(s)}\right)}) \leq d(\mathcal{G}_T) \tag{3.33}$$

because the elimination of tree branches does not increase the diameter (c.f. Lemma 3.4.3).

Hence, for all $m = 0, 1, \ldots, l$,

$$d(\mathcal{G}_{U^{(m)}}) \leq d(\mathcal{G}_T) + |\cup_{s=m+1}^{(l)} F^{(s)}| \tag{3.34}$$

$$\leq d(\mathcal{G}_T) + |F|. \tag{3.35}$$

Since by Assumption 3.4.1 $d_i \geq d(\mathcal{G}_T) + |F|$ for all $i \in \mathcal{V}$, we have that $d_i \geq d(\mathcal{G}_{U^{(m)}})$ for all $i \in \mathcal{V}$ and all $m = 0, 1, \ldots, l$.

Fourth, we prove that when $\mathcal{G}_{U^{(0)}}$ has cycles the first feedback node is elected in at most two outer iterations and it is the same node as chosen in the centralized algorithm. In the first outer iteration of Algorithm 3.2.4, since $d_i \geq d(\mathcal{G}_{U^{(0)}})$, all nodes on the tree branches of $\mathcal{G}_{U^{(0)}}$ are marked as status T. Let i^* be the node having the highest priority score in $U^{(0)}$. There are two cases:

1) If $i^* \notin T^{(1)}$, i.e., node i^* is not on any tree branch of $\mathcal{G}_{U^{(0)}}$, then i^* (and only i^*) will be the first elected feedback node, which is exactly the same node as selected in the centralized algorithm by performing graph cleaning following by choosing the node with the highest score.

2) If $i^* \in T^{(1)}$, i.e., node i^* is on some tree branch of $\mathcal{G}_{U^{(0)}}$, then no node will be elected in the first outer iteration because for each node $i \in U^{(0)} \backslash T^{(1)}$, we have $\text{Maxid}(i) = i^*$ since the elimination of i^* occurs after or at the same time as p_{i^*} is passed to neighbors. In this case, no feedback node is elected, i.e., $|F^{(1)}| = 0$ and thus $U^{(1)}$ does not have any tree branches (since previous tree branches $T^{(1)}$ are removed and no new tree branches are created when $F^{(1)} = \emptyset$.) Hence, the second outer iteration is equivalent to electing the node with the highest score on $\mathcal{G}_{U^{(1)}}$, which is again the same node as chosen using the centralized algorithm.

We can repeat the same analysis for the following outer iterations and conclude that every additional feedback node (before the remaining graph with status U becomes cycle-free) is elected in at most two outer iterations, and the sequence of elected nodes is the same as using the centralized selection algorithm. Therefore, there exists an integer

k_0 between $\lfloor \frac{l}{2} \rfloor$ and l such that the set of elected feedback nodes are the same as the set obtained by running the centralized selection algorithm (Algorithm 2.2.1) with k_0 iterations.

□

Lemma 3.4.3 : *1. If a node i is on a tree branch of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is a subgraph of $\mathcal{G}$ such that $i \in \mathcal{V}'$, then node i is also on a tree branch of the subgraph $\mathcal{G}'$.*

2. The elimination of a tree branch does not break a connected graph.

3. The elimination of a tree branch does not increase the graph diameter.

4. If both $\mathcal{G}_A$ and $\mathcal{G}_{A \cup B}$ are connected, then $d(\mathcal{G}_{A \cup B}) \leq d(\mathcal{G}_A) + |B|$.

The proof of Lemma 3.4.3 is provided in Appendix 3.6. We give a concrete example of the case where one feedback node is elected in two consecutive outer iterations in Figure 3.9.

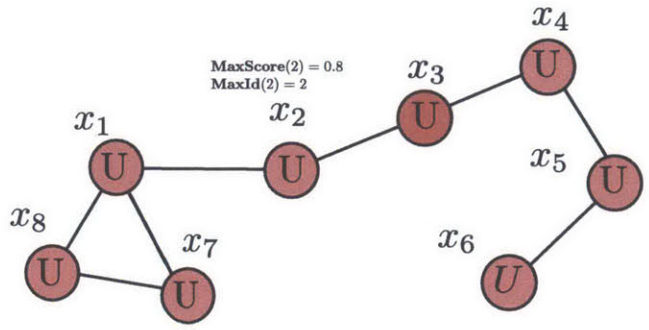
The following Proposition 3.4.5 states the exactness of recursive FMP when the set of feedback nodes F is a full FVS.

Assumption 3.4.4 : *1. F is a full FVS.*

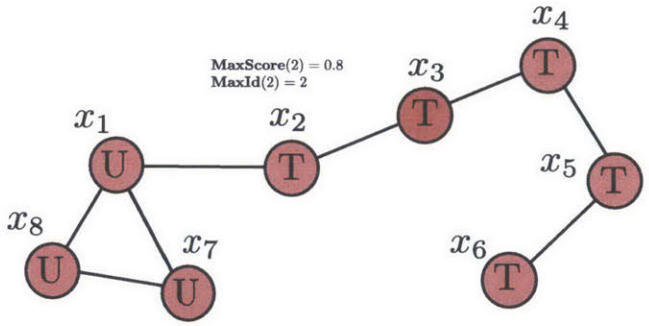
2. $|F| \leq K_i$, $\forall i \in \mathcal{V}$.

3. The number of maximum allowed iterations for LBP in Stage II, i.e., the timeout, is at least $\max_{C \in \mathcal{C}} d(\mathcal{G}_C)$, where $\mathcal{C}$ is the set of all connected components of $\mathcal{G}_T$.

Now we introduce some additional notation that is used in the following part of this section. In this proof, we use the following notation for submatrices. For any sets of nodes A and B, we use J_{AB} to denote the submatrix of the information matrix J taking the rows in A and columns in B and $\mathbf{h}_A$ to denote the subvector with entries in set A. As an abbreviation, we use J_A for J_{AA}. In addition, we use F_m to denote the set of feedback nodes with the top-$(|F| - m)$ priority scores for $m = 0, 1, 2, \ldots, |F|$ (i.e., the set obtained by removing the nodes with the lowest m priority scores from F) and let $T_m = \mathcal{V} \backslash F_m$. In particular, $F = F_0$ and $T = T_0 = \mathcal{V} \backslash F$. For all $i \in \mathcal{V}$, let $\mathcal{L}_i$ denote the priority list at node i at the end of Stage II (i.e., after running LBP).



(a) Before an outer iteration



(b) After an outer iteration

Figure 3.9: An example of electing the feedback nodes. (a) Assume that node 2 has the highest priority score 0.8 and that $d = 5$. (b) At the end of the outer iteration, each node i stores $\text{MaxScore}(i) = 0.8$ and $\text{MaxId}(i) = 2$, but node 2 is not elected as as a feedback node since it has status T.

Proposition 3.4.5 : *Under Assumption 3.4.4, recursive FMP converges and computes the exact means and exact variances for all nodes.*

Proof. In this proof, we first analyze the local mean and variance estimates at the end of Stage II regardless of whether $\mathcal{G}_T$ is connected. Next, we analyze the local estimates after receiving correction messages, where we need to look at several cases.

Local Values at the End of Stage II Note that the priority lists at all the active nodes are initially empty and the lists increase when the information about more and more feedback nodes is received with the LBP messages. Eventually, nodes in the same connected components of $\mathcal{G}_T$ have the same priority lists, which are subsets of F_0. Since

$K_i \geq |F_0|$ for all $i \in \mathcal{V}$, no feedback node will become active before the end of Stage II (which may happen as in (3.15) when the maximum capacity is exceeded). Hence, in Stage II, no node changes its status. Hence, the LBP messages are only passed among each connected component of $\mathcal{G}_{T_0}$.

First, we give closed-form expressions for the local values (i.e., μ_i, Σ_{ii}, g_i^q, $\forall q \in \mathcal{L}_i$) at each active node at the end of Stage II. To distinguish the local values at different time steps, we use the following notation to represent the local values at each $i \in T_0$ at the end of Stage II:

1. Vector $\boldsymbol{\mu}^{\mathcal{T}_0}$ for the means, where each entry $\mu_i^{\mathcal{T}_0}$ equals the local mean estimate "μ_i";
2. The vector $\mathrm{Diag}(\Sigma^{\mathcal{T}_0})$ taking the diagonal of $\Sigma^{\mathcal{T}_0}$, where each diagonal entry $\Sigma_{ii}^{\mathcal{T}_0}$ equals the local variance estimate "Σ_{ii}";
3. Vectors $\mathbf{g}^{\mathcal{T}_0,q}$ for all $q \in F$, where each entry $g_i^{\mathcal{T}_0,q}$ equals the local value "g_i^q" for $q \in \mathcal{L}_i$ and $g_i^{\mathcal{T}_0,q} = 0$ for $q \notin \mathcal{L}_i$.

Because $F_0 = F$ is a full FVS, $\mathcal{G}_{T_0}$ is a tree-structured graph and thus LBP on $\mathcal{G}_{T_0}$ converges within $\max_{C \in \mathcal{C}} d(\mathcal{G}_C)$ iterations and gives exact inference results on $\mathcal{G}_{T_0}$, i.e.,

$$\boldsymbol{\mu}^{\mathcal{T}_0} = J_{T_0}^{-1}\mathbf{h}_{T_0} \tag{3.36}$$

$$\Sigma_{ii}^{\mathcal{T}_0} = (J_{T_0}^{-1})_{ii} \text{ for all } i \in T_0. \tag{3.37}$$

Equations 3.36 and 3.37 are true even when $\mathcal{G}_{T_0}$ is not connected because in this case LBP on $\mathcal{G}_{T_0}$ is equivalent to LBP on each connected component of $\mathcal{G}_{T_0}$.

In addition, from the definition of the auxiliary potential vector $\mathbf{h}^q$ (c.f. (3.9)) and our convention of using default value zero for $g_i^{\mathcal{T}_0,q}$ when $i \notin \mathcal{L}_i$, regardless of whether $\mathcal{G}_{T_0}$ is connected and whether $\mathcal{L}_i$ are the same for all i, we always have

$$\mathbf{g}^{\mathcal{T}_0,q} = J_{T_0}^{-1} J_{T_0,q}, \text{ for all } q \in F_0. \tag{3.38}$$

Updated Local Values After Receiving Correction Messages Corresponding to Each Feedback Node As mentioned previously, it is possible that $\mathcal{L}_i$ are different at different node i depending on the graph topology. We first study the case where it is guaranteed to have $\mathcal{L}_i = F_0$ for all $i \in \mathcal{V}$ and show that the local mean and variance estimates are eventually the exact inference results. Then we analyze the case where it is possible to

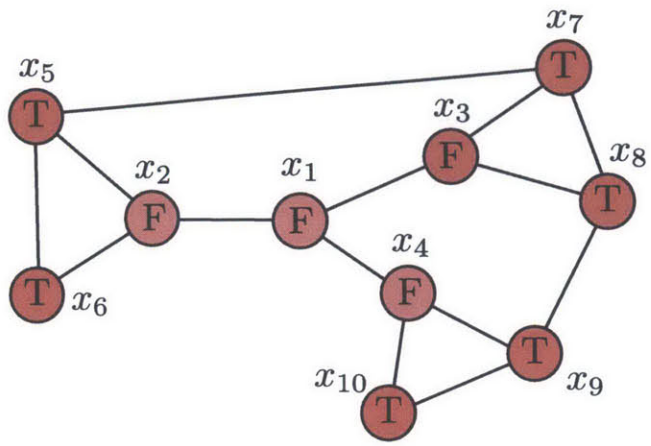


Figure 3.10: An example where $\mathcal{G}_T$ is connected but $\mathcal{L}_i \subsetneq F$

have different priority lists at the end of Stage II and show that the inference results are still exact. Note that even when $\mathcal{G}_T$ is connected, it is still possible to have $\mathcal{L}_i \subsetneq F_0$, which occurs when some feedback node q has no active neighbors. For example, in Figure 3.10, $\mathcal{G}_T$ is connected but all neighbors of node 1 have status F and thus $1 \notin \mathcal{L}_i$ for any active node i. The scenario in Figure 3.10 is possible when the feedback nodes 1, 2, 3, and 4 are elected in order when they have the top-4 priority scores.

Case 1: $\mathcal{G}_{T_0}$ is connected and each node $i \in F_0$ is connected to at least one node in T_0
From the local update equations (3.10) and (3.14), we can see that the information about all the feedback nodes is passed to all nodes with the LBP messages at the end of Stage II and thus $\mathcal{L}_i = F_0$ for all $i \in \mathcal{V}$.

Note that it is possible to have multiple different correction messages being forwarded at the same time. Now we establish that the correction messages are initiated in ascending order of their corresponding priority scores and that for any node $i \in \mathcal{V}$, the nodes in $\mathcal{L}_i$ are removed in ascending order of the corresponding priority scores regardless of which feedback node is closer to i on the graph. This is because a feedback node q wakes up (and initiates its correction message) only when all nodes in $\mathcal{L}_q$ with lower priority scores are removed, which are only triggered by the corresponding correction messages. Hence, those correction messages are initiated and have reached q's active neighbors before node q wakes up. So those correction messages would reach any other nodes before q's correction messages do.

In the following, for $m = 1, 2, \ldots, |F_0|$, we use the vector $\boldsymbol{\mu}^{\mathcal{T}_m}$ to denote the local

means at nodes in T_m after receiving m correction messages (i.e., those corresponding to the feedback nodes with lowest-m priority scores). Note that the entries in $\boldsymbol{\mu}^{\mathcal{T}_m}$ are not necessarily local means at the same time, but are defined as the local values right after receiving the same number of correction messages. Similarly, we use Diag $(\Sigma^{\mathcal{T}_m})$ and $\mathbf{g}^{\mathcal{T}_m,q}$ to denote the local variances and local feedback gains after receiving m correction messages.

Let q be the feedback node with the lowest priority score. We can re-write the update equations (3.16) and (3.17) (which compute the local values at q when node q wakes up) as

$$\Sigma_{qq}^{\mathcal{T}_1} = \left(J_{qq} - J_{T_0,q}\mathbf{g}_{T_0}^{q}\right)^{-1} \tag{3.39}$$

$$\mu_q^{\mathcal{T}_1} = \Sigma_{qq}^{\mathcal{T}_1}\left(h_q - J_{T_0,q}\boldsymbol{\mu}^{\mathcal{T}_0}\right), \tag{3.40}$$

where $\boldsymbol{\mu}^{\mathcal{T}_0} = J_{T_0}^{-1}\mathbf{h}_{T_0}$ and $\mathbf{g}^{\mathcal{T}_0,q} = J_{T_0}^{-1}J_{T_0,q}$, for all $q \in F_0$ computed exactly by LBP in Stage II.

According to Lemma 3.4.6 to follow, these update equations exactly give

$$\Sigma_{qq}^{\mathcal{T}_1} = \left(J_{T_1}^{-1}\right)_{qq} \tag{3.41}$$

$$\mu_q^{\mathcal{T}_1} = J_{T_1}^{-1}\mathbf{h}_{T_1}. \tag{3.42}$$

The update equations (3.20) and (3.65) at the active nodes (which make corrections to the local values after receiving correction messages corresponding to node q) can be re-written as

$$\text{Diag}\left(\Sigma_{T_0}^{\mathcal{T}_1}\right) = \text{Diag}\left(\Sigma^{\mathcal{T}_0} + \mathbf{g}^{\mathcal{T}_0,q}\Sigma_{qq}^{\mathcal{T}_1}\mathbf{g}^{\mathcal{T}_0,q}\right) \tag{3.43}$$

$$\boldsymbol{\mu}_{T_0}^{\mathcal{T}_1} = \boldsymbol{\mu}^{\mathcal{T}_0} - \mu_q^{\mathcal{T}_1}\mathbf{g}^{\mathcal{T}_0,q}. \tag{3.44}$$

According to Lemma 3.4.6, these update equations exactly give

$$\text{Diag}\left(\Sigma_{T_0}^{\mathcal{T}_1}\right) = \text{Diag}\left(\left(J_{T_1}^{-1}\right)_{T_0}\right) \tag{3.45}$$

$$\boldsymbol{\mu}_{T_0}^{\mathcal{T}_1} = \left(J_{T_1}^{-1}\mathbf{h}_{T_1}\right)_{T_0}. \tag{3.46}$$

Combining equations (3.41), (3.42), (3.45) and (3.46), we have that

$$\boldsymbol{\mu}^{\mathcal{T}_1} = J_{T_1}^{-1}\mathbf{h}_{T_1} \tag{3.47}$$

$$\text{Diag}(\Sigma^{\mathcal{T}_1}) = \text{Diag}(J_{T_1}^{-1}). \tag{3.48}$$

The newly computed feedback gains at node q using equation (3.18) (when node q wakes up) can be re-written as

$$g_{qq}^{\mathcal{T}_1,p} = \Sigma_{qq}^{\mathcal{T}_1}\left(J_{pq} - J'_{T_0,q}\mathbf{g}^{\mathcal{T}_0,p}\right), \forall p \in \mathcal{L}_q\backslash\{q\}. \tag{3.49}$$

For each $p \in \mathcal{L}_q\backslash\{q\}$, viewing $\mathbf{g}^{\mathcal{T}_1,p}$ as $\boldsymbol{\mu}$ in Lemma 3.4.6 and $J_{T_1,p}$ as $\mathbf{h}$ in Lemma 3.4.6, we have that

$$g_{qq}^{\mathcal{T}_1,p} = \left(J_{T_1}^{-1}J_{T_1,p}\right)_{qq}. \tag{3.50}$$

The feedback gains at nodes in T_0 are then updated using (3.22), which can be re-written as

$$\mathbf{g}_{T_0}^{\mathcal{T}_1,p} = \mathbf{g}^{\mathcal{T}_0,p} - g_{qq}^{\mathcal{T}_1,p}\mathbf{g}^{\mathcal{T}_0,q}. \tag{3.51}$$

From (3.38), LBP in Stage II gives

$$\mathbf{g}^{\mathcal{T}_0,p} = J_{T_0}^{-1}J_{T_0,p}, \forall p \in \mathcal{L}_q\backslash\{q\} \tag{3.52}$$

$$\mathbf{g}^{\mathcal{T}_0,q} = J_{T_0}^{-1}J_{T_0,q}. \tag{3.53}$$

Substituting (3.52) and (3.53) into (3.51), we can obtain

$$\mathbf{g}_{T_0}^{\mathcal{T}_1,p} = J_{T_0}^{-1}J_{T_0,p} - g_{qq}^{\mathcal{T}_1,p}J_{T_0}^{-1}J_{T_0,q} \tag{3.54}$$

$$= J_{T_0}^{-1}\left(J_{T_0,p} - \left(g_{qq}^{\mathcal{T}_1,p}\right)J_{T_0,q}\right). \tag{3.55}$$

By viewing $J_{T_1,p}$ here as $\mathbf{h}$ in Lemma 3.4.6, and $\mathbf{g}^{\mathcal{T}_1,p}$ as $\boldsymbol{\mu}$ in Lemma 3.4.6, we have that

$$\mathbf{g}_{T_0}^{\mathcal{T}_1,p} = \left(J_{T_1}^{-1} J_{T_1,p}\right)_{T_0}. \tag{3.56}$$

Combining (3.51) and (3.56), we can obtain

$$\mathbf{g}^{\mathcal{T}_1,p} = J_{T_1}^{-1} J_{T_1,p} \text{ for all } p \in \mathcal{L}_q \backslash \{q\}. \tag{3.57}$$

Therefore, we have that

$$\boldsymbol{\mu}^{\mathcal{T}_1} = J_{T_1}^{-1} \mathbf{h}_{T_1} \tag{3.58}$$

$$\text{Diag}(\Sigma^{\mathcal{T}_1}) = \text{Diag}(J_{T_1}^{-1}) \tag{3.59}$$

$$\mathbf{g}^{\mathcal{T}_1,p} = J_{T_1}^{-1} J_{T_1,p} \text{ for all } p \in F \backslash \{q\}, \tag{3.60}$$

where we have the same mathematical structure as in (3.36), (3.37) and (3.38). Also, $\mathcal{G}_{T_1}$ is still connected and all nodes in $F_1 = F\backslash\{q\}$ are connected to at least one node in T_1 (because $F_1 \subset F_0$ and $T_1 \supset T_0$). Hence, we can repeat the same process for $m = 2, 3, \ldots, |F|$, all in Case 1 and eventually obtain the exact inference results

$$\boldsymbol{\mu}^{\mathcal{T}_{|F|}} = J^{-1}\mathbf{h} \tag{3.61}$$

$$\text{Diag}(\Sigma^{\mathcal{T}_{|F|}}) = \text{Diag}(J^{-1}). \tag{3.62}$$

Case 2: the active subgraph $\mathcal{G}_{T_0}$ is not connected, or $\mathcal{G}_{T_0}$ is connected but some node in F_0 is not connected to any node in T_0 In this case, it is possible that $\mathcal{L}_i \subsetneq F$ for some i. At each connected component of $\mathcal{G}_{T_0}$, the correction messages are still initiated in ascending order of the corresponding priority scores (of nodes in $\mathcal{L}_i$ while possibly skipping some feedback nodes in F).

We use C_i to denote all the nodes in the same connected component as node i. In order for the same analysis in Case 1 to apply, we only need to show that the "imaginary" correction messages corresponding to a node $q \notin \mathcal{L}_i$ (i.e., imagining q is put in $\mathcal{L}_i$ by an oracle) equal zero. Since the maximum allowed number of iterations is at least $\max_{C\in\mathcal{C}} d(\mathcal{G}_C)$, the only case where $q \in F$ but $q \notin \mathcal{L}_i$ is when node q has no edge connected to any node in C_i, i.e., $J_{C_i,q} = \mathbf{0}$. Thus, from (3.38), we have that $\mathbf{g}^{\mathcal{T}_0,q} = \mathbf{0}$. Hence, the local values at node i do not change after adding these "imaginary" correction messages corresponding to $q \notin \mathcal{L}_i$ from the update equations (3.20), (3.21) and (3.22).

We can repeat the same analysis whenever a feedback node is skipped and conclude that the same steps of Case 1 still apply.

Therefore, we have completed the proof of Proposition 3.4.5.

□

Lemma 3.4.6 : *Let* $J = \begin{bmatrix} J_{11} & J_M' \\ J_M & J_T \end{bmatrix}$ *be an invertible matrix and* $\mathbf{h} = \begin{bmatrix} h_1 \\ \mathbf{h}_T \end{bmatrix}$, *where* J_{11} *and* h_1 *are scalars,* J_M *is an* n*-by-*n *matrix, and* J_M *and* $\mathbf{h}_T$ *are* n*-dimensional vectors. We denote* $\Sigma = J^{-1} = \begin{bmatrix} \Sigma_{11} & \Sigma_M' \\ \Sigma_M & \Sigma_T \end{bmatrix}$, $\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \boldsymbol{\mu}_T \end{bmatrix} = J^{-1}\mathbf{h}$, $\mathbf{g} = J_T^{-1}J_M$, $\boldsymbol{\mu}^{\mathcal{T}} = J_T^{-1}\mathbf{h}_T$, *and* $\Sigma^{\mathcal{T}} = J_T^{-1}$. *We have that*

$$\Sigma_{11} = \left(J_{11} - J_M'\mathbf{g}\right)^{-1} \tag{3.63}$$

$$\mu_1 = \Sigma_{11}\left(h_1 - J_M'\boldsymbol{\mu}^{\mathcal{T}}\right) \tag{3.64}$$

and

$$\boldsymbol{\mu}_T = J_T^{-1}\left(\mathbf{h}_T - \mu_1 J_M\right) \tag{3.65}$$

$$\Sigma_T = \Sigma^{\mathcal{T}} + \mathbf{g}_1\Sigma_{11}\mathbf{g}_1'. \tag{3.66}$$

The proof of Lemma 3.4.6 is provided in Appendix 3.6.

Assumption 3.4.7 : *1.* $|F| \leq K_i$ *for all* $i \in \mathcal{V}$;

2. J_T, *the model on the subgraph* $\mathcal{G}_T$ *is walk-summable;*

3. *The maximum number of iterations is sufficiently large to allow LBP in Stage II to converge.*

Now we define some new notation for the reminder of this section. For $i \in T$, we use $\mathcal{L}_i^{(0)} = \{i_1, i_2, \ldots, i_{|\mathcal{L}_i^{(0)}|}\}$ to denote the priority list at i at the end of Stage II (i.e., after LBP), where the elements are in the ascending order of the corresponding priority scores. We use $\mathcal{L}_i^{(m)}$ to denote the set obtained by removing the nodes with the lowest-m priority scores from $\mathcal{L}_i^{(0)}$. For $q \in F$, we use $\mathcal{L}_q^{(0)}$ to denote its priority list at node q *just*

after it becomes an active node and similarly use $\mathcal{L}_q^{(m)}$ to denote the set obtained by removing the nodes with the lowest-m priority scores from $\mathcal{L}_q^{(0)}$. In particular, $\mathcal{L}_i^{(0)} = \mathcal{L}_i$ if $i \in T$ and $\mathcal{L}_i^{(0)} \subsetneq \mathcal{L}_i$ if $i \in F$ (since $i \in \mathcal{L}_i$ but $i \notin \mathcal{L}_i^{(0)}$).

The following Proposition 3.4.8 gives new walk-sum interpretations of some intermediate results and correction terms in the recursive FMP algorithm.

Proposition 3.4.8 : *Under Assumption 3.4.7 we have the following conclusions regardless of whether F is a full FVS or a pseudo FVS:*

1. *Recursive FMP converges.*
2. *At the end of Stage II, the local variance estimate at a node $i \in T$ equals the sum of all backtracking self-return walks of i within $\mathcal{G}_{\mathcal{V}\backslash\mathcal{L}_i^{(0)}}$.*
3. *Just after a node $q \in F$ coverts to an active node, the computed local variance equals the sum of all self-return walks of q within the subgraph $\mathcal{G}_{\mathcal{V}\backslash\mathcal{L}_q^{(0)}}$.*
4. *At any active node i (including those converted from an inactive node), after receiving a correction message corresponding to node q, the variance correction term in (3.20) equals the sum of all self-return walks of i within $\mathcal{G}_{\mathcal{V}\backslash\mathcal{L}_i^{(m^*)}}$ that visit node q at least once, where m^* is the node such that $\{q\} = \mathcal{L}^{(m^*-1)}\backslash\mathcal{L}^{(m^*)}$, i.e., $\mathcal{L}_i^{(m^*)}$ is the priority list at node i right after the local correction.*

Proof. In this proof, when not otherwise stated, the notation follows that in the proof of Proposition 3.4.5.

First we prove Conclusion 1: Since the model on the subgraph $\mathcal{G}_T$, i.e., J_T, is walk-summable, according to Proposition 2.2.3, LBP on $\mathcal{G}_T$ converges and gives correct means (for arbitrary potential vectors including the auxiliary ones for computing the feedback gains) for that subgraph, i.e., after convergence we have that

$$\boldsymbol{\mu}^{\mathcal{T}_0} = J_{T_0}^{-1}\mathbf{h}_{T_0} \tag{3.67}$$

$$\mathbf{g}^{\mathcal{T}_0,p} = J_{T_0}^{-1}J_{T_0,p}\forall p \in F. \tag{3.68}$$

Note that by the convention as in the proof of Proposition 3.4.5, $g_i^{\mathcal{T}_0,p} = 0$ when $p \notin \mathcal{L}_i^{(0)}$. Also, by the same analysis as in the proof of Proposition 3.4.5, no feedback node will become active before the end of Stage II (which may happen in (3.15) when the

maximum capacity is exceeded). Since LBP in Stage II is the only iterative procedure in the recursive FMP, we have that recursive FMP converges under Assumption 3.4.7.

From Proposition 2.2.4, we also have that at the end of Stage II, the local variance estimate at a node $i \in T$ equals the sum of all backtracking self-return walks of i within $\mathcal{G}_{\mathcal{V}\setminus\mathcal{L}_i^{(0)}}$.

To prove Conclusions 3 and 4, we consider two cases.

Case 1: $\mathcal{G}_{T_0}$ is connected and each node $i \in F_0$ is connected to at least one node in T_0 Using the same analysis as in the proof of Proposition 3.4.5, we have that $\mathcal{L}_i = F$ for all $i \in \mathcal{V}$ and the feedback nodes are removed from $\mathcal{L}_i$ in ascending order of their priority scores.

When the first feedback node q wakes up, the equation to compute the local variance (i.e., (3.16)) is the same as Step 3 of the hybrid FMP constrained on $\mathcal{G}_{T_1}$ with a single feedback node q (c.f. (2.27) and (2.29)). Hence, $\Sigma_{qq}^{\mathcal{T}_1}$ is the exact inference result within $\mathcal{G}_{T_1}$ according to Theorem 2.2.6. From (2.18), we have that the newly computed local variance $\Sigma_{qq}^{\mathcal{T}_1}$ equals the sum of all the self-return walks from q to q within $\mathcal{G}_{T_1} = \mathcal{G}_{\mathcal{V}\setminus\mathcal{L}_q^{(0)}}$.

After an active node i receives the correction message corresponding to q, the variance correction term in the update equation (3.20) is the same as the correction term in the hybrid FMP algorithm (c.f. (2.31)) constrained to $\mathcal{G}_{T_1}$ with a single feedback node q. According to Theorem 2.2.6, the updated local variance equals the sum of all the backtracking walks of node i within $\mathcal{G}_{T_0}$ plus all the self-return walks of node i within $\mathcal{G}_{T_1}$ that visit node q. From Conclusion 1, we have that the correction term equals the sum of all self-return walks of i within $\mathcal{G}_{T_1} = \mathcal{G}_{\mathcal{V}\setminus\mathcal{L}_i^{(0)}}$ that visit node q at least once.

Since $\mathbf{g}^{\mathcal{T}_0,p} = J_{T_0}^{-1} J_{T_0,p}, \forall p \in F$ from (3.68) and the update equations for the feedback gains (c.f. (3.18) and (3.22)) only involve quantities that are computed exactly by LBP (means computed from various potential vector and $\Sigma_{qq}^{\mathcal{T}_1}$), we have that $\mathbf{g}^{\mathcal{T}_1,p} = J_{T_1}^{-1} J_{T_1,p} \forall p \in F_1$.

Similar to the proof of Proposition 3.4.5, $\mathcal{G}_{T_1}$ is still connected and all nodes in F_1 are connected to at least one node in T_1. Hence, we can repeat the same process all in Case 1 and eventually obtain Conclusion 3 and 4.

Case 2: the active subgraph $\mathcal{G}_{T_0}$ is not connected, or $\mathcal{G}_{T_0}$ is connected but some node in F_0 is not connected to any node in T_0 Again, we can follow the same analysis as in the proof of Proposition 2.2.3 to show that the "imaginary" correction messages corresponding to a node $q \notin \mathcal{L}_i$ are all zero. In order for Conclusion 3 and 4 to stand,

we only need to show that the corresponding "imaginary" walk-sums are also zero. This is because no walk from node i can visit a feedback node q when $q \notin \mathcal{L}_i$ (which can only happen when q is not connected to C_i, the connected component of $\mathcal{G}_{T_0}$ containing node i) and thus the corresponding walk-sum is zero.

Therefore, we have proved the four conclusions in Proposition 3.4.8.

□

Theorem 3.4.9 : *Under Assumption 3.4.7, recursive FMP gives the same inference results as the hybrid FMP algorithm (Algorithm 2.2.2) with the same F, i.e., it gives the correct means for all nodes and the correct variances for nodes in F and the calculated variance of node i in T equals the sum of all the backtracking walks of node i within T plus all the self-return walks of node i that visit F.*

Proof. For the means, the same analysis as in the proof of Proposition 3.4.5 still applies. This is because after convergence, LBP in Stage II gives the correct mean and feedback gain estimates and all the update equations for the means only involve the values that are computed exactly. Hence, the computed means are exact for all nodes in the whole graph after convergence.

In the rest of this proof, we use $\phi(i \overset{\mathcal{G}_A}{\to} i)$ to denote the self-return walk-sum for node i within the subgraph $\mathcal{G}_A$ and use $\phi_q(i \overset{\mathcal{G}_A}{\to} i)$ to denote the self-return walk-sum for node i within $\mathcal{G}_A$ that visit node q at least once. In addition, $\phi(i \overset{\text{bk, } \mathcal{G}_A}{\longrightarrow} i)$ denotes the sum of all the backtracking self-return walks within $\mathcal{G}_A$ and $\phi_q(i \overset{\text{bk, } \mathcal{G}_A}{\longrightarrow} i)$ denotes the sum of all the backtracking self-return walks within $\mathcal{G}_A$ that visit node q at least once.

Now we prove the statements for the variances. For any node $q \in F$, when it first becomes an active node, the newly computed local variance estimate equals

$$\phi(q \xrightarrow{\mathcal{G}_{\mathcal{V}\setminus\mathcal{L}_q^{(0)}}} q) \tag{3.69}$$

according to Proposition 3.4.8. After adding all the correction terms for the variance, the final variance estimate Σ_{qq}^* satisfies

$$\Sigma_{qq}^* = \phi(q \xrightarrow{\mathcal{G}_{\mathcal{V}\setminus\mathcal{L}_q^{(0)}}} q) + \sum_{s=1}^{|\mathcal{L}_q^{(0)}|} \phi_{q_s}(q \xrightarrow{\mathcal{G}_{\mathcal{V}\setminus\mathcal{L}_q^{(s)}}} q), \tag{3.70}$$

where $\{q_s\} = \mathcal{L}_q^{(s-1)} \backslash \mathcal{L}_q^{(s)}$. The right hand side of (3.70) is exactly a decomposition of all self-return walks from of q in the entire graph. Hence, the final variance estimate Σ_{qq}^* is the exact variance on the entire graph according to (2.18).

For a node $i \in T$, at the end of Stage II, according to Proposition 3.4.8, the local variance estimate obtained at the end of Stage II equals

$$\phi(i \xrightarrow{\text{bk, } \mathcal{G}_{\mathcal{V} \backslash \mathcal{L}_i^{(0)}}} i). \tag{3.71}$$

Hence, according to Proposition 3.4.8, the final variance estimate Σ_{ii}^* satisfies

$$\Sigma_{ii}^* = \phi(i \xrightarrow{\text{bk, } \mathcal{G}_{\mathcal{V} \backslash \mathcal{L}_i^{(0)}}} i) + \sum_{s=1}^{|\mathcal{L}_i^{(0)}|} \phi_{i_s}(i \xrightarrow{\mathcal{G}_{\mathcal{V} \backslash \mathcal{L}_i^{(s)}}} i), \tag{3.72}$$

where $\{i_s\} = \mathcal{L}_i^{(s-1)} \backslash \mathcal{L}_i^{(s)}$.

Using the same analysis as in the proofs of Proposition 3.4.8, when $\mathcal{L}_i^{(0)} \subsetneq \mathcal{L}_i \subsetneq F$ the feedback nodes in $F \backslash \mathcal{L}_i$ do not contribute to the walk-sums. Hence, (3.72) can be re-written as

$$\Sigma_{ii}^* = \phi(i \xrightarrow{\text{bk, } \mathcal{G}_T} i) + \sum_{s=1}^{|F|} \phi_{f_s}(i \xrightarrow{\mathcal{G}_{T_s}} i), \tag{3.73}$$

where $\{f_s\} = F_{s-1} \backslash F_s$. The term $\sum_{s=1}^{|F|} \phi_{f_s}(i \xrightarrow{\mathcal{G}_{T_s}} i)$ is exactly a decomposition of the sum of all walks within the whole graph $\mathcal{G}$ that visit F.

Therefore, the calculated variance of node i in T equals the sum of all the back-tracking walks of node i within T plus all the self-return walks of node i that visit F. We have thus proved Theorem 3.4.9.

□

3.5 Experimental Results

In this section, we demonstrate the performance of recursive FMP using simulated models on grids of various sizes as well as a large-scale GGM for estimating sea surface height anomaly (SSHA).

Simulated Grids of Various Sizes

In this motivating experiment, we consider GGMs defined on grids of size $s \times s$, where $s = 4,\ 6,\ 8, \ldots, 18,\ 20$. For each size, we simulate 50 models with randomly generated parameters. Specifically, the information matrix J and potential vector $\mathbf{h}$ are randomly generated as follows: the entries of $\mathbf{h}$ are generated *i.i.d.* from a uniform distribution $U[-1, 1]$; the sparsity pattern of J is determined by the graph structure and the non-zero entries of J are also generated *i.i.d.* from $U[-1, 1]$ with a multiple of the identity matrix added to ensure $J \succ 0$. We solve the inference problems (i.e., computing the variances and means) using LBP and recursive FMP with various parameter settings. The parameters in recursive FMP include d_i, the effective diameter; l_i, the number of outer iterations in Stage I, and K_i, the maximum capacity of the priority list. In this experiment, we use uniform parameters for all nodes, i.e., $d_i = d$, $l_i = l$, and $K_i = K$ for all $i \in \mathcal{V}$. In addition, we let $l = K$ for all algorithm settings and let the maximum number of iterations (timeout) be twice the effective diameter. Since the model sizes are moderate, we can compute the exact solution for comparison. In particular, for each algorithm setting and each grid size, we compute the average error of variances, i.e., $\frac{1}{n}\sum_{i\in\mathcal{V}} |\Sigma_{ii} - \Sigma_{ii}^{\text{approx}}|$, where Σ_{ii} denotes the exact variance of node i and $\Sigma_{ii}^{\text{approx}}$ denote the approximate variance of node i computed by certain algorithm. Furthermore, for each size, the final results are averaged over the 50 sets of randomly constructed model parameters.

Our experimental results are presented in Figure 3.11, where the horizontal axis represents the size of the grids and the vertical axis represents the logarithm of the average error of variances. As shown in the plots, LBP has the worst performance among all algorithm settings for all grid sizes. If we run recursive FMP with fixed $K = 2$ and fixed $d = 3$ for all sizes, we obtain significant improvement in average error over LBP for all sizes. If we let K grow logarithmically with respect to the total number of nodes (i.e., $K = \lceil \log(s^2) \rceil$) while using fixed effective diameter $d = 3$, then we obtain further improvement on average error, but the error reduction is not very significant compared with using fixed $K = 2$, indicating that the increased list capacity is not fully utilized due to the fixed d. Finally, when we keep the relationship between K and s as $K = \lceil \log(s^2) \rceil$ while also make d grow linearly with respect to s (in particular, $d = 2s$),[18] then we can obtain further significant improvement compared with using $K = \lceil \log(s^2) \rceil$

[18]For grids of size $s \times s$, the diameter of the graph grows linearly with respect to s.

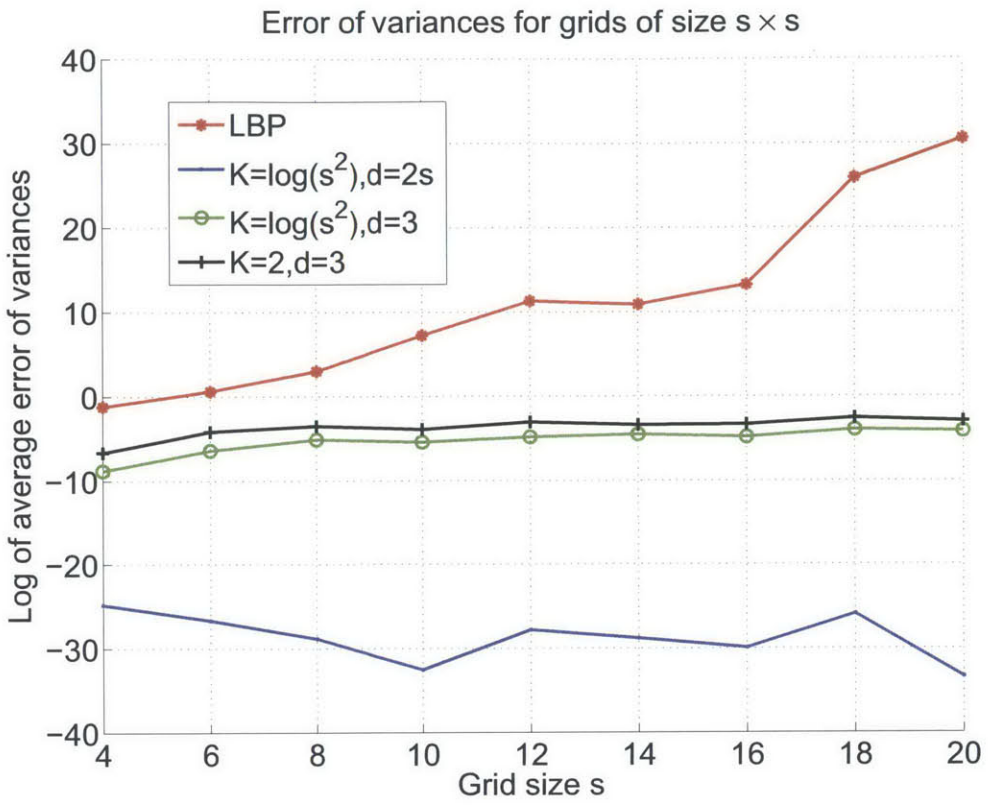


Figure 3.11: Recursive FMP with different parameters performed on grids of various sizes

and $d = 3$.[19] Note that the size of a full FVS for grids of size $s \times s$ is $\mathcal{O}(s^2)$, which grows much faster with s than either of our choices of K. Compared with running the hybrid FMP algorithm with a pseudo-FVS of size K, here in the recursive FMP algorithm, different nodes have different local lists of feedback nodes, which provide stronger local influence as they break more cycles in local regions. Our experimental results here provide insight on the trade-off between memory/communication capacity and inference accuracy. From Figure 3.11, we can also see that our algorithm is more effective for larger graphs since the error of LBP grow much faster with the grid size.

[19]Note computations in Stage II scale linearly with d.

Sea Surface Height Anomaly Data

In this experiment, we use sea surface height anomaly data, which is measured relative to seasonal, space-variant mean-sea level (the dataset is publicly available at http://podaac.jpl.nasa.gov/dataset/). The raw data is preprocessed to have measurements at 915×1080 different locations with latitudes between $\pm 82°$ and a full $360°$ of longitude.

We construct a grid of 988,200 nodes and connect the eastmost and westmost nodes at the same latitudes because they are geographical neighbors. We then remove the nodes that have invalid measurements (most of which correspond to land areas) and obtain the final graph structure shown in Figure 3.12a. Using this underlying structure, we build a GGM using the thin-membrane model [1]. We run recursive FMP with uniform $d_i = 200$ and $K_i = l_i = 15$. Our distributed algorithm converges with the specified parameters in a few seconds and the final estimates are plotted in Figure 3.12b.

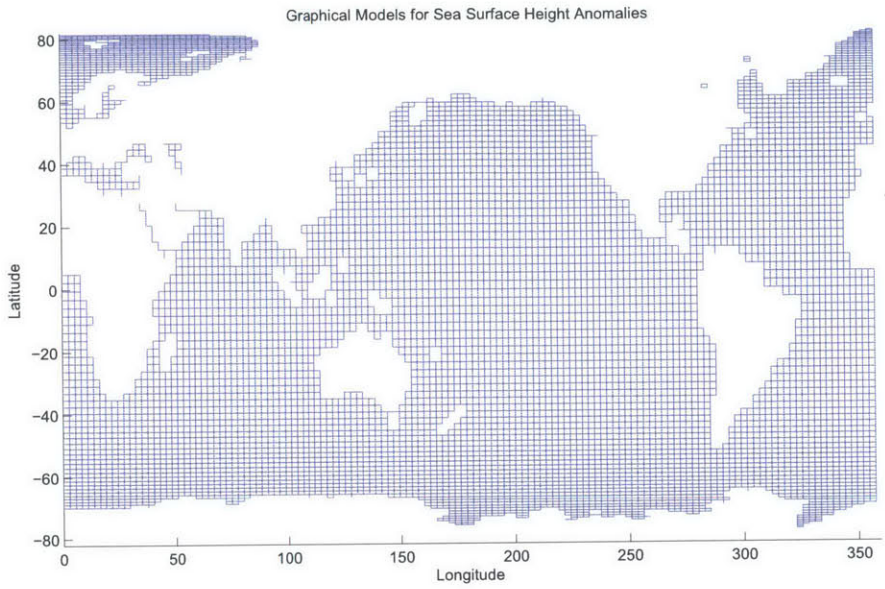
3.6 Appendix for Chapter 3

Proof of Lemma 3.2.1

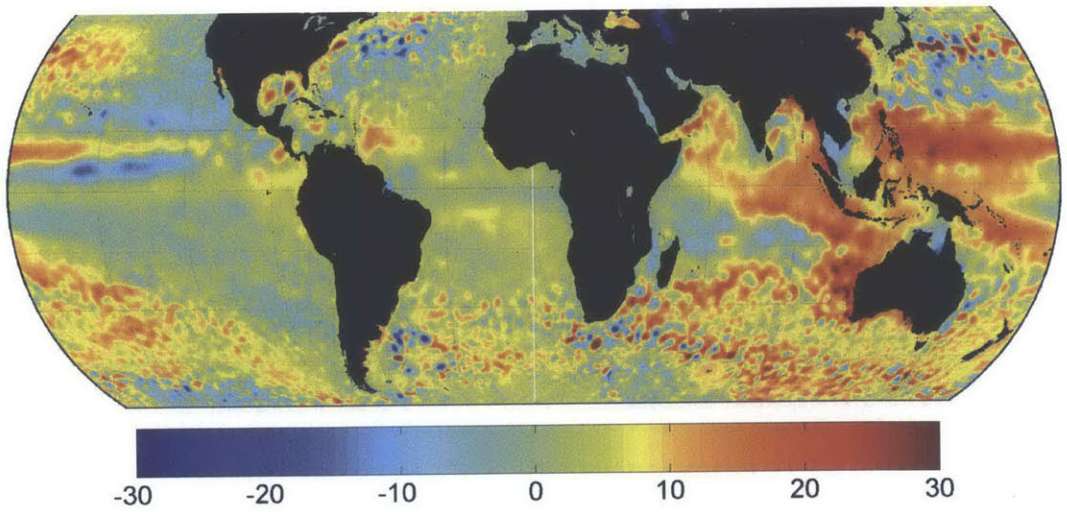
Lemma 3.2.1 : *After running Algorithm 3.2.3, all nodes that are not on the tree branches have status U regardless of d. If d is greater than the diameter of the graph, then every node i that is on some tree branch has status T.*

Proof. Consider the change of status to T as node elimination. Then $\mathcal{N}_{\mathrm{U}}(i)$ equals the set of node i's neighbors that are still on the graph. So, any node that changes status to T is on a tree branch by definition of graph cleaning. Hence, after running Algorithm 3.2.3, all nodes that are not on the tree branches have status U regardless of d.

In the first iteration of Algorithm 3.2.3, all leaf nodes (nodes that have degree zero or one) are eliminated. In the second iteration, every node on some tree branches that is within distance one to its nearest leaf node is eliminated. Repeating this analysis, we can show that every node on some tree branch is eliminated after d iterations if d is greater than its distance to its nearest leaf node, which is at most the diameter of the graph. Hence, after running Algorithm 3.2.3, if d is greater than the diameter of the graph, then every node i that is on some tree branch has status T. We have thus



(a) GGM for SSHA. For clarity, the grids are drawn coarser, and the edges connecting the eastmost and westmost nodes are not shown.



(b) Estimated SSHA with $d = 200$ and $l_i = K_i = 15$ for all i.

Figure 3.12: Estimating SSHA using recursive FMP

completed the proof.

□

Proof of Lemma 3.4.3

Lemma 3.4.3 : *1. If a node i is on a tree branch of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is a subgraph of $\mathcal{G}$ such that $i \in \mathcal{V}'$, then node i is also on a tree branch of the subgraph $\mathcal{G}'$.*

2. The elimination of a tree branch does not break a connected graph.

3. The elimination of a tree branch does not increase the graph diameter.

4. If both $\mathcal{G}_A$ and $\mathcal{G}_{A\cup B}$ are connected, then $d(\mathcal{G}_{A\cup B}) \leq d(\mathcal{G}_A) + |B|$.

1. Let $i_1, i_2, \ldots i_l$ be a sequence of nodes eliminated in the clean-up procedure on graph $\mathcal{G}$ where $i_l = i$. We follow the same sequence and remove those nodes from graph $\mathcal{G}'$where we skip all $i_s \notin \mathcal{V}'$ for $s = 1, 2, \ldots, l$. Since $\mathcal{E}' \subset \mathcal{E}$, we have that each node removal is a valid step of graph clean-up in $\mathcal{G}'$ as well. Hence, i is also on a tree branch of $\mathcal{G}'$.

2. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected graph and $\{i_1, i_2, \ldots i_l\}$ be any sequence of node removal in a graph clean-up procedure, which requires that the degree of i_1 in $\mathcal{G}$ must be one or zero. Since $\mathcal{G}$ is connected, the degree of i_1 in $\mathcal{G}$ is one. If $\mathcal{G}_{\mathcal{V}\setminus\{i_1\}}$ is disconnected, then i_1 is connected to only one of the connected components of $\mathcal{G}_{\mathcal{V}\setminus\{i_1\}}$ and thus $\mathcal{G}$ is disconnected, which is a contradiction. Hence, $\mathcal{G}_{\mathcal{V}\setminus\{i_1\}}$ must be a connected graph. We can repeat the same analysis for other nodes in the sequence in order and conclude that the elimination of a tree branch does not break a connected graph.

3. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph and $\{i_1, i_2, \ldots i_l\}$ be any sequence of node removal in a graph clean-up procedure. If $\mathcal{G}$ is disconnected, we have $d(\mathcal{G}) = \infty$ and thus the conclusion is trivially true. When $\mathcal{G}$ is connected, let i^* be the single node that is connected to i_1. Any path between two nodes different from i_1 does not pass through i_1 because otherwise i^* will be included twice in the sequence which is contradictory to the definition of path. Hence, after the removal of i_1, the distance between any pair of nodes different from i_1 remains the same (while in

the new graph, when computing the graph diameter, we do not need to consider the distance between i_1 and another node as i_1 has been removed). Hence, we have that the diameter of the graph does not increase after removing i_1. By repeating the same analysis for other nodes in the sequence in order, we conclude that the elimination of a tree branch does not increase the graph diameter.

4. For any graph $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$ and any two nodes $i, j \in \mathcal{V}^*$, we use $d_{\mathcal{G}^*}(i,j)$ to denote the distance between node i and node j within graph $\mathcal{G}^*$.
First, we prove the conclusion for the case where B has a single node $b_1 \notin A$. For any $i_1, i_2 \in A$, we have that $d_{\mathcal{G}_{A\cup B}}(i_1,i_2) \leq d_{\mathcal{G}_A}(i_1,i_2) \leq d(\mathcal{G}_A)$ because there are no fewer choices of paths between i_1 and i_2 on $\mathcal{G}_{A\cup B}$ than on $\mathcal{G}_A$. Since $\mathcal{G}_{A\cup B}$ is connected, we can choose $i^* \in A$ to be a node that is directly connected to b_1. So, for any $i_1 \in A$, $d_{\mathcal{G}_{A\cup B}}(i_1,b_1) \leq d_{\mathcal{G}_A}(i_1,i^*) + 1 \leq d(\mathcal{G}_A) + 1$. Hence, when $|B| = 1$, we have $d(\mathcal{G}_{A\cup B}) \leq d(\mathcal{G}_A) + |B|$.
Second, when $|B|$ has multiple nodes, since $\mathcal{G}_{A\cup B}$ is connected there is at least one node b_1 that is directly connected to A and thus by the previous case, $\mathcal{G}_{A\cup\{b_1\}}$ is connected and thus $d(\mathcal{G}_{A\cup\{b_1\}}) \leq d(\mathcal{G}_A) + 1$. Similarly, $d(\mathcal{G}_{A\cup\{b_1\}\cup\{b_2\}}) \leq d(\mathcal{G}_{A\cup\{b_1\}}) + 1 \leq d(\mathcal{G}_A) + 2$. We can repeat the same analysis can conclude that $d(\mathcal{G}_{A\cup B}) \leq d(\mathcal{G}_A) + |B|$.

We have completed the proof of Lemma 3.4.3.

Proof of Lemma 3.4.6

Lemma 3.4.6 : *Let* $J = \begin{bmatrix} J_{11} & J_M' \\ J_M & J_T \end{bmatrix}$ *be an invertible matrix and* $\mathbf{h} = \begin{bmatrix} h_1 \\ \mathbf{h}_T \end{bmatrix}$, *where* J_{11} *and* h_1 *are scalars,* J_M *is an* n*-by-*n *matrix, and* J_M *and* $\mathbf{h}_T$ *are* n*-dimensional vectors. We denote* $\Sigma = J^{-1} = \begin{bmatrix} \Sigma_{11} & \Sigma_M' \\ \Sigma_M & \Sigma_T \end{bmatrix}$, $\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \boldsymbol{\mu}_T \end{bmatrix} = J^{-1}\mathbf{h}$, $\mathbf{g} = J_T^{-1}J_M$, $\boldsymbol{\mu}^{\mathcal{T}} = J_T^{-1}\mathbf{h}_T$, *and* $\Sigma^{\mathcal{T}} = J_T^{-1}$. *We have that*

$$\Sigma_{11} = \left(J_{11} - J_M'\mathbf{g}\right)^{-1} \tag{3.74}$$

$$\mu_1 = \Sigma_{11}\left(h_1 - J_M'\boldsymbol{\mu}^{\mathcal{T}}\right) \tag{3.75}$$

and

$$\boldsymbol{\mu}_T = J_T^{-1}\left(\mathbf{h}_T - \mu_1 J_M\right) \tag{3.76}$$

$$\Sigma_T = \Sigma^{\mathcal{T}} + \mathbf{g}_1 \Sigma_{11} \mathbf{g}_1^{'}. \tag{3.77}$$

By taking the Schur complement of block J_T in J, we have

$$\Sigma_{11}^{-1} = J_{11} - J_M^{'} J_T^{-1} J_M. \tag{3.78}$$

Since $\mathbf{g} = J_T^{-1} J_M$, we get

$$\Sigma_{11} = \left(J_{11} - J_M^{'}\mathbf{g}\right)^{-1}. \tag{3.79}$$

Using the matrix inversion lemma, we have

$$\Sigma_T = J_T^{-1} + \left(J_T^{-1} J_M\right)\left(J_{11} - J_M^{'} J_T^{-1} J_M\right)^{-1}\left(J_T^{-1} J_M\right)^{'}. \tag{3.80}$$

Substituting $\mathbf{g} = J_T^{-1} J_M$, $J_T^{-1} = \Sigma^{\mathcal{T}}$ and (3.79) into 3.80, we have

$$\Sigma_T = \Sigma^{\mathcal{T}} + \mathbf{g}_1 \Sigma_{11} \mathbf{g}_1^{'}. \tag{3.81}$$

From the definition of Σ, we have

$$\begin{bmatrix} J_{11} & J_M^{'} \\ J_M & J_T \end{bmatrix} \begin{bmatrix} \Sigma_{11} & \Sigma_M^{'} \\ \Sigma_M & \Sigma_T \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}_{1\times n} \\ \mathbf{0}_{n\times 1} & I_{n\times n} \end{bmatrix}, \tag{3.82}$$

where $\mathbf{0}_{n\times 1}$ is the all-zero matrix of dimension $n\times 1$, $\mathbf{0}_{1\times n}$ is the all-zero matrix of dimension $1\times n$, and $I_{n\times n}$ is the identity matrix of dimension $n\times n$. By matrix multiplication, we have

$$J_M \Sigma_{11} + J_T \Sigma_M = 0 \tag{3.83}$$

and thus

$$\Sigma_M = -\left(J_T^{-1} J_M\right)\Sigma_{11}. \tag{3.84}$$

From the definition of $\boldsymbol{\mu}$, we have

$$\begin{bmatrix} \mu_1 \\ \boldsymbol{\mu}_T \end{bmatrix} = \begin{bmatrix} \Sigma_{11} & \Sigma_M^{'} \\ \Sigma_M & \Sigma_T \end{bmatrix} \begin{bmatrix} h_1 \\ \mathbf{h}_T \end{bmatrix} \tag{3.85}$$

and thus

$$\mu_1 = \Sigma_{11} h_1 + \Sigma_M^{'} h_T. \tag{3.86}$$

Substituting (3.84) into (3.86), we have

$$\mu_1 = \Sigma_{11} h_1 - \Sigma_{11} J_M^{'} \left(J_T^{-1} h_T \right) \tag{3.87}$$

$$= \Sigma_{11} \left(h_1 - J_M^{'} \boldsymbol{\mu}^{\mathcal{T}} \right), \tag{3.88}$$

where $\boldsymbol{\mu}^{\mathcal{T}} = J_T^{-1} \mathbf{h}_T$.

Again by the definition of $\boldsymbol{\mu}$, we have

$$\begin{bmatrix} J_{11} & J_M^{'} \\ J_M & J_T \end{bmatrix} \begin{bmatrix} \mu_1 \\ \boldsymbol{\mu}_T \end{bmatrix} = \begin{bmatrix} h_1 \\ \mathbf{h}_T \end{bmatrix} \tag{3.89}$$

and thus

$$J_M \mu_1 + J_T \boldsymbol{\mu}_T = \mathbf{h}_T, \tag{3.90}$$

which gives

$$\boldsymbol{\mu}_T = J_T^{-1} \left(\mathbf{h}_T - \mu_1 J_M \right). \tag{3.91}$$

Therefore, we have completed the proof of Lemma 3.4.6.

Chapter 4

Sampling Gaussian Graphical Models Using Subgraph Perturbations

4.1 Introduction

In this chapter, we propose a general framework to convert iterative linear solvers based on graphical splittings to MCMC samplers by adding a random perturbation at each iteration. In particular, the algorithm can be thought of as a stochastic version of graph-based solvers and, in fact, is motivated by the use of embedded trees in [32, 33] for the computation of the mean of a GGMs. That approach corresponds to decomposing the graph of the model into a tractable graph[1], i.e., one for which sampling is easy (e.g., a tree), and a "cut" matrix capturing the edges removed to form the tractable subgraph. The subgraphs used can have any structure for which efficient inference algorithms exist: for example, tree-structured, low tree-width, or having a small FVS [16]. Much more importantly, in order to obtain a valid sampling algorithm, we must exercise some care, not needed or considered for the linear solvers in [32, 33], in constructing the graphical models corresponding to both the tractable subgraph and to the set of variables involved in the cut edges.

We give general conditions under which graph-based iterative linear solvers can be converted into samplers and we relate these conditions to the so-called P-regularity condition [34]. We then provide a simple construction that produces a splitting satisfying

[1]Here the subgraph is a spanning subgraph, i.e., one that includes all of the vertices and a subset of all edges.

those conditions. Once we have such a decomposition our algorithm proceeds at each iteration by generating a sample from the model on the subgraph and then randomly perturbing it based on the model corresponding to the cut edges. That perturbation obviously must admit tractable sampling itself and also must be shaped so that the resulting samples of the overall model are asymptotically exact. Our construction ensures both of these. As was demonstrated in [32, 33], using non-stationary splittings, i.e., different graphical decompositions in successive iterations, can lead to substantial gains in convergence speed. We extend our subgraph perturbation algorithm from stationary graphical splittings to non-stationary graphical splittings and give theoretical results for convergence guarantees. We propose an algorithm to select tractable subgraphs for stationary splittings and an adaptive method for selecting non-stationary splittings.

The authors of [61] have proposed a sampling framework that generalizes and accelerates the Gibbs sampler. Previous work in [62] has shown that the Gibbs sampler is a stochastic version of the Gauss-Seidel iteration for solving learning systems. The sampling algorithm in [61] adds additional noises corresponding to the first or second order Chebyshev coefficients to accelerate the Gibbs sampler. While the idea of converting a linear solver to a sampler is also discussed in [61], their work is different from ours because their algorithm does not consider graph structures in constructing the matrix splitting that is used (i.e., the sparsity pattern of the base matrix remains the same without considering any tractable subgraphs). Moreover, when multiple matrix splittings are used, the different splittings in [61] have differences only in the Chebyshev coefficients while in our work, different matrix splittings correspond to different graph structures.

Formally, the sampling problem considered in this chapter is to efficiently generate samples from a GGM with underlying distribution $\mathcal{N}^{-1}(\mathbf{h}, J)$ with given model parameters $\mathbf{h}$ and J . We consider iterative samplers that produce a sequence of samples $\mathbf{x}^{(t)}$ for $t = 1, 2, \ldots$. An iterative sampling algorithm is correct if the samples converge in distribution to the target distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ where $\boldsymbol{\mu} = J^{-1}\mathbf{h}$ and $\Sigma = J^{-1}$. If the process to generate this sequence is Gaussian, then the marginal distribution of each iteration is fully described by its mean $\boldsymbol{\mu}^{(t)}$ and covariance matrix $\Sigma^{(t)}$. In this case, the convergence of the sampler is equivalent to $\boldsymbol{\mu}^{(t)} \to \boldsymbol{\mu}$ and $\Sigma^{(t)} \to \Sigma$ as $t \to \infty$.

As we are especially interested in fast convergence to the target distribution, we need a clear notion of convergence rate. In the study of MCMC samplers, convergence rate is often measured by the total variation of the sample distribution from the target

distribution [24]. In this chapter, for convenience, we instead use the Euclidean norm (denoted by $||\cdot||$) of the difference of the means and the Frobenius norm (denoted by $||\cdot||_F$) of the difference of the covariance matrices to measure the deviation of the sample distribution from the target distribution. It can be shown that for non-degenerate Gaussian models, the convergence in total variation is equivalent to the convergence in the model parameters. In particular, we define convergence rate for the mean as

$$\tau_{\boldsymbol{\mu}} = -\ln \limsup_{t\to\infty} \frac{\|\boldsymbol{\mu}^{(t+1)} - \boldsymbol{\mu}\|}{\|\boldsymbol{\mu}^{(t)} - \boldsymbol{\mu}\|}, \tag{4.1}$$

and convergence rate for the covariance as

$$\tau_{\Sigma} = -\frac{1}{2} \ln \limsup_{t\to\infty} \frac{\|\Sigma^{(t+1)} - \Sigma\|_F}{\|\Sigma^{(t)} - \Sigma\|_F}. \tag{4.2}$$

4.2 Sampling by Subgraph Perturbations with Stationary Graphical Splittings

In this section, we introduce our subgraph perturbation sampling framework using stationary (fixed) splittings. First, we describe the general framework with an arbitrary graphical splitting followed by theoretical results on convergence. We then describe a local construction of the splitting that builds up the decomposition as a sum of rank-one terms corresponding to each of the edges removed from the tractable graph. The construction of this splitting is simple to perform at run time, leads to very efficient sampling of the perturbation term required in the sampling algorithm, and ensures convergence.

4.2.1 General Algorithm

Our sampling framework relies on a graph-based matrix splitting. Given the information matrix J with underlying graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, consider the splitting

$$J = J_T - K, \tag{4.3}$$

where J_T has sparsity corresponding to a tractable subgraph $\mathcal{G}_T = (\mathcal{V}, \mathcal{E}_T)$ with $\mathcal{E}_T \subset \mathcal{E}$, and K has sparsity corresponding to the graph with edge set $\mathcal{E}\backslash\mathcal{E}_T$ (See Figure 4.1). Throughout this chapter, we assume that the splittings we consider are all *graphical*

splittings, i.e., both J_T and K are symmetric matrices corresponding to undirected graphs.

In [32] this type of splitting (although crucially without one of the further assumptions we will make) was proposed for the computation of the mean (but not covariance or samples). In particular, when J_T is non-singular the mean is computed using the iterative equation

$$\boldsymbol{\mu}^{(t+1)} = J_T^{-1}\left(K\boldsymbol{\mu}^{(t)} + \mathbf{h}\right) \tag{4.4}$$

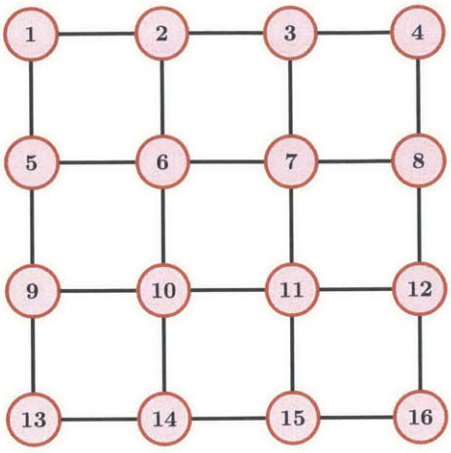
with the fixed-point solution $\boldsymbol{\mu} = J^{-1}\mathbf{h}$. Note that the sequence in (4.4) converges if and only if $\rho(J_T^{-1}K) < 1$. For this to be efficient, the computation on the right-hand side of (4.4) would need to be far simpler than solving for the mean directly, using the full matrix J. The approach in [32] took J_T to have tree structure (so that the computation in (4.4) has linear complexity), although in principle it is possible to choose J_T to have other graph structures (e.g., as in [63]) that lead to tractable computations. Moreover, while the approach is not limited to the following, the original idea in [32] and especially in [33] is simply to "cut" edges from the graph, so that J_T is obtained from J simply by zeroing out the elements corresponding to the cut edges, and $-K$ is the matrix whose only nonzero elements are the values corresponding to those cut edges.

The high-level idea of our sampling algorithm is to further inject noise into (4.4), so that the iterative linear solver becomes a stochastic process whose stationary distribution is the target distribution $\mathcal{N}^{-1}(\mathbf{h}, J)$. However, the simple idea of constructing K by copying the elements of J corresponding to cut edges may not be feasible for our sampling algorithm. Rather, we need to ensure that K is chosen so that $J_T + K \succ 0$. Assuming that we have a splitting that satisfies this condition, our iterative sampling algorithm is given by:

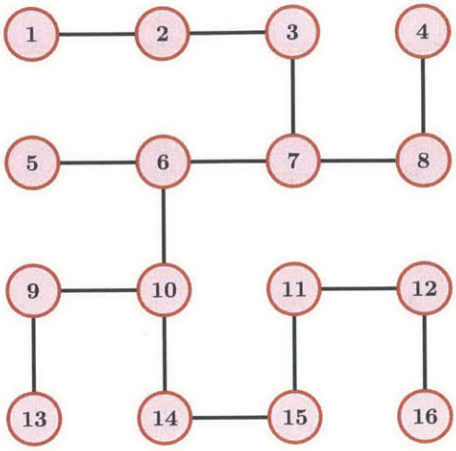
$$\mathbf{x}^{(t+1)} = J_T^{-1}\left(K\mathbf{x}^{(t)} + \mathbf{h} + \mathbf{e}^{(t+1)}\right), \tag{4.5}$$

where the perturbation $\mathbf{e}^{(t+1)}$ is a Gaussian random vector, independent of all other variables, with zero mean and covariance $J_T + K$. The general sampling framework is summarized in Algorithm 4.2.1.

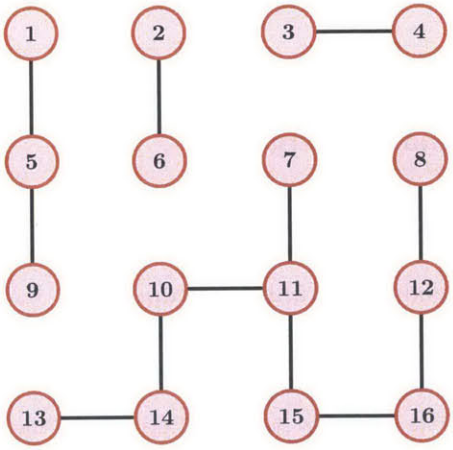
In the next subsection we provide theoretical results showing that convergence of the iteration in (4.4) for a graphical splitting is equivalent to the condition $J_T + K \succ 0$, which in turn implies that the sampling method in (4.5) is well-defined. We also show



(a) Graph structure for J



(b) Graph structure for J_T



(c) Graph structure for K

Figure 4.1: Decomposition of a grid: The grid shown in (a) can be decomposed into a spanning tree (b) and a graph consisting of the missing edges (c)

that in this case, the sample distribution indeed converges to the correct distribution. In the last part of this section, we provide a straightforward "local" edge-by-edge method for constructing such a splitting that also directly yields an efficient generation of the perturbation $\mathbf{e}^{(t+1)}$.

Algorithm 4.2.1 Sampling by Subgraph Perturbations with Stationary Splittings

Input: J, $\mathbf{h}$, and subgraph structure $\mathcal{T}$
Output: samples with the asymptotic distribution $\mathcal{N}^{-1}(\mathbf{h}, J)$

1. Form J_T and K.
2. Draw an initial sample $\mathbf{x}^{(0)}$ from a Gaussian distribution.
3. At each iteration:
   (a) Generate an independent sample $\mathbf{e}^{(t+1)}$ with zero mean and covariance matrix $J_T + K$.
   (b) Compute $\mathbf{x}^{(t+1)}$ using the equation

$$\mathbf{x}^{(t+1)} = J_T^{-1}\left(\mathbf{h} + K\mathbf{x}^{(t)} + \mathbf{e}^{(t+1)}\right).$$

4.2.2 Correctness and Convergence

In this subsection, we present theoretical results for the general subgraph perturbation framework. Proposition 4.2.1 and Theorem 4.2.3 establish the correctness of Algorithm 4.2.1 as well as a convergence guarantee. Proposition 4.2.4 and Corollary 4.2.5 give bounds on the convergence rate.

In general, a matrix splitting $A = M - N$ is called a *P-regular splitting* if M is non-singular and $M^T + N$ is positive definite [34]. The P-regularity condition has been proposed in the study of iterative linear solvers as a condition for convergence [34, 64]. In our graphical splitting $J = J_T - K$, since J_T is symmetric, the P-regular condition $J_T^T + K \succ 0$ is precisely the condition that the added noise term in our perturbation framework is valid, i.e., that it corresponds to a random variable with positive definite covariance. Therefore, our sampling framework provides a new interpretation of the P-regularity condition—for graphical splittings as in (4.3), convergence of iterative solvers as in (4.4) is equivalent to the noise in $\mathbf{e}^{(t+1)}$ being valid. It has been shown in [32] that the necessary and sufficient condition for the embedded tree algorithm to converge with any initial point is $\rho(J_T^{-1}K) < 1$. In Proposition 4.2.1 we prove that this condition is equivalent to the graphical splitting being P-regular, which further guarantees the validity of the added noise in (4.5).

Proposition 4.2.1 : *Assuming $J \succ 0$ and that $J = J_T - K$ is a graphical splitting, the condition $\rho(J_T^{-1}K) < 1$ is satisfied if and only if the splitting is P-regular, i.e., the added noise in Algorithm 4.2.1 has a valid covariance matrix $J_T + K \succ 0$.*

The proof of Proposition 4.2.1 is included in Appendix 4.6. The following Lemma 4.2.2 is used in the proof of Theorem 4.2.3, which is our main result in this section. The proof of Lemma 4.2.2 is also deferred to Appendix 4.6.

Lemma 4.2.2 : *Let A and B be square matrices. If 1) A is invertible; 2) $A + B$ is symmetric and invertible, then $\Sigma = (A + B)^{-1}$ is a solution of the equation $A\Sigma A^T = B\Sigma B^T + A^T - B$.*

The following Theorem 4.2.3 states that for graphical splittings, a convergent linear solver can be converted to a convergent sampler with the same convergence rate.

Theorem 4.2.3 : *For a valid GGM with information matrix $J \succ 0$, let $J = J_T - K$ be a graphical splitting. If the corresponding linear solver converges, i.e., $\rho(J_T^{-1}K) < 1$, then the sample distribution generated by Algorithm 4.2.1 is guaranteed to converge to the target distribution and the asymptotic convergence rates $\tau_{\boldsymbol{\mu}}$ for the mean and τ_Σ for the covariance are both equal to $-\ln \rho(J_T^{-1}K)$.*

Proof. From Proposition 4.2.1, we have that $J_T + K \succ 0$, i.e., the covariance matrix of the added noise is valid. It can be shown that with the initial sample distribution being Gaussian, the iterations in Algorithm 4.2.1 generate a sequence of Gaussian samples, with $\mathbf{x}^{(t)}$ having mean $\boldsymbol{\mu}^{(t)}$ and covariance matrix $\Sigma^{(t)}$. From Step 3(b) in Algorithm 4.2.1, we have

$$\begin{aligned} \boldsymbol{\mu}^{(t+1)} &= \mathbb{E}\left[\mathbf{x}^{(t+1)}\right] = \mathbb{E}\left[J_T^{-1}(\mathbf{h} + \mathbf{e}^{(t+1)} + K\mathbf{x}^{(t)})\right] && (4.6) \\ &= J_T^{-1}(\mathbf{h} + K\boldsymbol{\mu}^{(t)}). && (4.7) \end{aligned}$$

Since $\rho(J_T^{-1}K) < 1$, the mean $\boldsymbol{\mu}^{(t+1)}$ converges to the unique fixed-point $\hat{\boldsymbol{\mu}}$ satisfying

$$\hat{\boldsymbol{\mu}} = J_T^{-1}\left(\mathbf{h} + K\hat{\boldsymbol{\mu}}\right). \tag{4.8}$$

So $\hat{\boldsymbol{\mu}} = (J_T - K)^{-1}\mathbf{h} = J^{-1}\mathbf{h}$, and thus $\boldsymbol{\mu}^{(t)}$ converges to the exact mean $\boldsymbol{\mu} = J^{-1}\mathbf{h}$ with convergence rate $\tau_{\boldsymbol{\mu}} = -\ln \rho(J_T^{-1}K)$.

Now we consider the convergence of the covariance matrix. From Step 3(b) in Algorithm 4.2.1, we have

$$\Sigma^{(t+1)} = \mathrm{Cov}\left\{\mathbf{x}^{(t+1)}\right\} \tag{4.9}$$

$$= \mathrm{Cov}\left\{J_T^{-1}(\mathbf{h} + \mathbf{e}^{(t+1)} + K\mathbf{x}^{(t)})\right\} \tag{4.10}$$

$$= J_T^{-1}\left(J_T + K + K\Sigma^{(t)}K\right)J_T^{-1} \tag{4.11}$$

$$= (J_T^{-1}K)\Sigma^{(t)}(J_T^{-1}K)^T + J_T^{-1}(J_T + K)J_T^{-1}. \tag{4.12}$$

This equation can be rewritten in vector form as

$$\begin{aligned}\mathrm{vec}(\Sigma^{(t+1)}) &= \left[(J_T^{-1}K)\otimes(J_T^{-1}K)\right]\mathrm{vec}(\Sigma^{(t)}) \\ &\quad +\mathrm{vec}(J_T^{-1}(J_T + K)J_T^{-1}),\end{aligned} \tag{4.13}$$

where $\mathrm{vec}(\cdot)$ denotes the column vector obtained by stacking all the columns in its argument and $A \otimes B$ denotes the Kronecker product of matrices A and B, i.e.,

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}, \tag{4.14}$$

where A is an m-by-n matrix $[a_{ij}]_{m\times n}$. From [65], $\rho\left((J_T^{-1}K)\otimes(J_T^{-1}K)\right)$, the spectral radius of the matrix $(J_T^{-1}K)\otimes(J_T^{-1}K)$, is $\rho^2(J_T^{-1}K) < 1$. Hence the iterative equation (4.13) is guaranteed to converge to a unique fixed-point, denoted by $\mathrm{vec}(\hat{\Sigma})$, with asymptotic convergence rate $-\ln\rho^2(J_T^{-1}K)$ in the Euclidean norm. Hence equation (4.12) converges to a unique fixed-point matrix $\hat{\Sigma}$. By Lemma 4.2.2, the fixed-point solution $\hat{\Sigma} = (J_T - K)^{-1} = J^{-1}$ is exactly the target covariance matrix. Hence, the convergence rate $\tau_\Sigma = -\frac{1}{2}\ln\rho^2(J_T^{-1}K) = -\ln\rho(J_T^{-1}K)$ since $\forall A$, $||\mathrm{vec}(A)|| = ||A||_F$. This completes the proof of Theorem 4.2.3.

□

We have shown in Theorem 4.2.3 that the convergence rates for both the mean and the covariance are $-\ln\rho(J_T^{-1}K)$. Naturally, we want to choose a splitting with a small $\rho(J_T^{-1}K)$. This spectral radius is a highly nonlinear function of both J_T and K, and it is useful to have bounds that are simple (and monotonic) functions of K or J_T

alone. The following Proposition 4.2.4 is adapted from Theorem 3 in [32]. For a valid distribution with $J \succ 0$, the condition of $K \succeq 0$ in Proposition 4.2.4 is sufficient to ensure $J_T + K \succ 0$, which guarantees the convergence of Algorithm 4.2.1. In the next subsection, we provide a local implementation of Algorithm 4.2.1 where the condition $K \succeq 0$ is satisfied.

Proposition 4.2.4 : *Consider a graphical splitting $J = J_T - K$ with $J \succ 0$. If $K \succeq 0$, then*

$$\frac{\lambda_{\max}(K)}{\lambda_{\max}(K) + \lambda_{\max}(J)} \leq \rho(J_T^{-1}K) \leq \frac{\lambda_{\max}(K)}{\lambda_{\max}(K) + \lambda_{\min}(J)} < 1, \tag{4.15}$$

where $\lambda_{\max}(\cdot)$ and $\lambda_{\min}(\cdot)$ denote the maximum and the minimum eigenvalues respectively.

Proof. Use Theorem 2.2 in [66] and let $\mu = 1$.

□

A simpler (and looser) bound that is much easier to compute (and hence can be used in choosing K) is given in the following Corollary 4.2.5. We define the weight of the i-th row of K are $w_i(K) = \sum_j |K_{ij}|$ and let $w(K) = \max_i w_i(K)$.

Corollary 4.2.5 : *In the same setting as in Proposition 4.2.4, we have $\rho(J_T^{-1}K) \leq \frac{w(K)}{w(K)+\lambda_{\min}(J)}$.*

Proof. Since $K \succeq 0$, we have $\lambda_{\max}(K) = \rho(K)$. By Corollary 8.1.18 in [67], we have that $\rho(K) \leq \rho(\overline{K})$, where $\overline{K}$ takes the entry-wise absolute values of K. By Corollary 8.1.22 in [67], $\rho(\bar{K}) \leq \max_i \Sigma_j \bar{K}_{ij}$, so $\lambda_{\max}(K) = \rho(K) \leq \rho(\overline{K}) \leq w(K)$. This corollary thus follows from Proposition 4.2.4.

□

■ 4.2.3 Efficient Local Implementation

Given a graphical splitting $J = J_T - K$, Algorithm 4.2.1 requires generating noise vectors $\mathbf{e}^{(t)}$ with zero mean and covariance $J_T + K \succ 0$. Depending on the splitting, these random noise vectors may be difficult to generate. In this subsection, given a tractable subgraph $\mathcal{T}$, we provide a method to construct the splitting matrices J_T and K specifically so that the noise vectors $\mathbf{e}^{(t)}$ can be constructed efficiently and to guarantee convergence. Moreover, our construction is entirely local with respect to the graph. In

this subsection, we focus on the construction of the splitting with a given subgraph and postpone the selection of subgraphs to Section 4.4.

Let $\mathcal{E}_\mathcal{T}$ denote the set of edges in the subgraph $\mathcal{T}$. We construct K to be sparse with respect to the subgraph with edge set $\mathcal{E}\backslash\mathcal{E}_\mathcal{T}$ as follows. For each $(i,j) \in \mathcal{E}\backslash\mathcal{E}_\mathcal{T}$, let $K^{(i,j)} = \begin{bmatrix} |J_{ij}| & -J_{ij} \\ -J_{ij} & |J_{ij}| \end{bmatrix}$, and let $\left[K^{(i,j)}\right]_{n\times n}$ be the n-by-n matrix zero-padded from $K^{(i,j)}$, i.e., the principal submatrix corresponding to rows (and columns) i and j of $\left[K^{(i,j)}\right]_{n\times n}$ equals $K^{(i,j)}$ while other entries are zero. It can be easily verified that $\left[K^{(i,j)}\right]_{n\times n} \succeq 0$. We define K to be the sum of these rank-one matrices as

$$K = \sum_{(i,j)\in\mathcal{E}\backslash\mathcal{E}_\mathcal{T}} \left[K^{(i,j)}\right]_{n\times n}. \tag{4.16}$$

The matrix $J_\mathcal{T}$ is then obtained by

$$J_\mathcal{T} = J + K. \tag{4.17}$$

Note that $J_\mathcal{T}$ is sparse with respect to $\mathcal{T}$. Moreover, K is positive semi-definite and $J_\mathcal{T}$ is positive definite (since J is positive definite for a valid model).

At iteration $t+1$ of the algorithm, rather than generating the noise vector $\mathbf{e}^{(t+1)}$ directly, instead we generate a noise vector $\tilde{\mathbf{e}}^{(t+1)}$ to be Gaussian with zero mean and covariance K, then let $\mathbf{x}^{(t+1)}$ be a sample from the Gaussian distribution with information matrix $J_\mathcal{T}$ and potential vector $K\mathbf{x}^{(t)} + \mathbf{h} + \tilde{\mathbf{e}}^{(t+1)}$. Hence we have

$$\mathbf{x}^{(t+1)} = J_\mathcal{T}^{-1}(K\mathbf{x}^{(t)} + \mathbf{h} + \tilde{\mathbf{e}}^{(t+1)}) + \mathbf{n}^{(t+1)}, \tag{4.18}$$

where $\mathbf{n}^{(t+1)}$ is Gaussian with zero mean and covariance $J_\mathcal{T}^{-1}$. The above procedure is equivalent to Algorithm 4.2.1 since $\tilde{\mathbf{e}}^{(t+1)} + J_\mathcal{T}\mathbf{n}^{(t+1)}$ is equal in distribution to $\mathbf{e}^{(t+1)}$, whose covariance matrix is $J_\mathcal{T} + K$. Note that $\boldsymbol{n}^{(t+1)}$ can be generated efficiently thanks to the assumption that $J_\mathcal{T}$ is tractable (e.g., if it is tree-structured, the sample can be generated by forward sampling). Furthermore, the structure of K allows $\tilde{\mathbf{e}}^{(t+1)}$ to be computed efficiently and locally: For each $(i,j) \in \mathcal{E}\backslash\mathcal{E}_\mathcal{T}$, let $\tilde{\mathbf{e}}^{(i,j)}$ be a two-dimensional vector sampled from a zero-mean Gaussian distribution with covariance matrix $K^{(i,j)}$. Moreover, note that since each of the matrices $K^{(i,j)}$ is rank-one, we can generate each of the $\tilde{\mathbf{e}}^{(i,j)}$ using an independent scalar sample drawn from the standard Gaussian

distribution $\mathcal{N}(0,1)$ and then multiplying this by the vector $[1, \text{-sgn}(J_{ij})]^T\sqrt{|J_{ij}|}$.We then obtain $\tilde{\mathbf{e}}^{(t+1)}$ by computing

$$\tilde{\mathbf{e}}^{(t+1)} = \sum_{(i,j)\in\mathcal{E}\setminus\mathcal{E}_{\mathcal{T}}} \left[\tilde{\mathbf{e}}^{(i,j)}\right]_n, \tag{4.19}$$

where $\left[\tilde{\mathbf{e}}^{(i,j)}\right]_n$ is the n-dimensional vector zero-padded from $\tilde{\mathbf{e}}^{(i,j)}$, i.e., the i-th and j-th entries of $\left[\tilde{\mathbf{e}}^{(i,j)}\right]_n$ take the two entries of $\tilde{\mathbf{e}}^{(i,j)}$and all other entries of $\left[\tilde{\mathbf{e}}^{(i,j)}\right]_n$ are zero.

We have that $J_{\mathcal{T}} + K \succ 0$ from our construction, so this constitutes a P-regular graphical splitting. Hence according to Proposition 4.2.1 and Theorem 4.2.3, the sample distribution converges to the target distribution. This local implementation is summarized in Algorithm 4.2.2. The computational complexity of one iteration is $C_{\mathcal{T}}+\mathcal{O}(|\mathcal{E}_K|)$, where $C_{\mathcal{T}}$ is the complexity of drawing a sample from the tractable subgraph $\mathcal{T}$ and $|\mathcal{E}_K| = |\mathcal{E}| - |\mathcal{E}_{\mathcal{T}}|$ is the number of edges missing from $J_{\mathcal{T}}$.

Algorithm 4.2.2 Sampling by Subgraph Perturbations with Local Implementation

Input: J, $\mathbf{h}$, and $\mathcal{T}$
Output: samples with the asymptotic distribution $\mathcal{N}^{-1}(\mathbf{h}, J)$

1. Construct $J_{\mathcal{T}}$ and K using (4.16) and (4.17).
2. Draw an initial sample $\mathbf{x}^{(0)}$ from a Gaussian distribution.
3. At each iteration:
   (a) Generate an independent sample $\tilde{\mathbf{e}}^{(t+1)}$ using (4.19).
   (b) Generate a sample $\mathbf{x}^{(t+1)}$ from $\mathcal{N}^{-1}(\mathbf{h} + K\mathbf{x}^{(t)} + \tilde{\mathbf{e}}^{(t+1)}, J_{\mathcal{T}})$.

4.3 Sampling by Subgraph Perturbations with Non-Stationary Graphical Splittings

In the previous section, we have introduced the subgraph perturbation algorithm with stationary splittings. It is natural to extend Algorithm 4.2.1 to using multiple subgraphs for different iterations (i.e., $J = J_{\mathcal{T}_t} - K_t$ at iteration t), which we refer to as *non-stationary* graphical splittings. Using non-stationary graphical splittings for sampling

is related to using non-stationary graphical splittings for inference (i.e., for computation of the mean) [32, 33], but the additional constraint $J_{T_t} + K_t \succ 0$ is needed to ensure that the added noise at each iteration is valid. In this section, we first summarize our sampling algorithm using non-stationary graphical splittings in Algorithm 4.3.1 and then present theoretical results on convergence. The results in this section provide theoretical foundations for the adaptive selection of the splittings, which will be studied in Section 4.4.

Algorithm 4.3.1 Sampling by Subgraph Perturbations with Non-Stationary Splittings

Input: J, $\mathbf{h}$
Output: samples with the asymptotic distribution $\mathcal{N}^{-1}(\mathbf{h}, J)$

1. Draw an initial sample $\mathbf{x}^{(0)}$ from a Gaussian distribution.
2. At each iteration t for $t = 1,\ 2,\ 3, \ldots$
   (a) Form a graphical splitting $J = J_{T_t} - K_t$, where $J_{T_t} + K_t \succ 0$.
   (b) Generate an independent sample $\mathbf{e}^{(t+1)}$ with zero mean and covariance matrix $J_{T_t} + K_t$.
   (c) Compute $\mathbf{x}^{(t+1)}$ using the equation

$$\mathbf{x}^{(t+1)} = J_{T_t}^{-1}\left(\mathbf{h} + K_t\mathbf{x}^{(t)} + \mathbf{e}^{(t+1)}\right).$$

The authors in [32] have studied the use of periodic splittings (i.e., using the set of splittings $\{J = J_{T_t} - K_t\}_{t=1}^{P}$ in a periodic manner) for inference as a special case of using an arbitrary sequence of splittings. In this case, the average convergence rate is $-\frac{1}{P}\sum_{t=1}^{P} \ln \rho(J_{T_t}^{-1}K_t)$. While a non-trivial sufficient condition guaranteeing convergence for a general P is difficult to find, the authors have given a sufficient condition for the case $P = 2$. In [33] the inference problem for a GGM is solved by adaptively selecting the next graphical splitting given the current error residual. The authors have proven that if a GGM is walk-summable (c.f. [33]), then their algorithm converges to the correct solution for an arbitrary sequence of splittings where the diagonal of each of the K's is fixed to be zero.

In order for our non-stationary perturbation sampler to proceed, the noise covariance

matrix at each iteration needs to be positive semidefinite (which is equivalent to the P-regularity condition according to Proposition 4.2.1). Because of this extra constraint, the conclusions for inference using non-stationary splittings do not directly apply to sampling. In the following Theorem 4.3.1, we prove that as long as we have the condition in the theorem, namely that each element in the set of splittings would produce by itself a convergent stationary perturbation sampler, the use of any arbitrary sequence from this set (including, of course, periodic selection) also leads to a convergent algorithm.

Theorem 4.3.1 : *Consider a finite collection of graphical splittings* $\mathcal{S} = \{J = J_{T_i} - K_i\}_{i=1}^{N}$. *The non-stationary subgraph perturbation sampling algorithm (Algorithm 4.3.1) converges to the target distribution with an arbitrary sequence of splittings chosen from* $\mathcal{S}$ *if and only if the stationary sampling algorithm (Algorithm 4.2.1) converges to the target distribution with each of the splittings in the sequence.*

We now state several lemmas prior to proving Theorem 4.3.1. The proofs for these lemmas are provided in Appendix 4.6.

Lemma 4.3.2 : *If* $J \succ 0$ *and the graphical splitting* $J = J_T - K$ *is P-regular, then there exists* $\epsilon > 0$ *such that* $J - \left(J_T^{-1}K\right)^T J \left(J_T^{-1}K\right) \succeq \epsilon J$.

Lemma 4.3.3 : *For a positive definite matrix* J, *we define the induced matrix norm* $||A||_{J\to J}$ *as* $||A||_{J\to J} = \max_{\mathbf{u}\neq 0} \frac{||A\mathbf{u}||_J}{||\mathbf{u}||_J}$, *where the vector norm* $||\mathbf{u}||_J$ *is defined by* $||\mathbf{u}||_J = \mathbf{u}^T J\mathbf{u}$.

For a P-regular graphical splitting $J = J_T - K$, Lemma 4.3.3 states that $||J_T^{-1}K||_{J\to J} < 1$. In general it is not true that $||KJ_T^{-1}||_{J\to J} < 1$; however, the following Lemma 4.3.4 establishes that under mild conditions there exists an integer p such that the J-induced norm of the product $\prod_{i=1}^{p} K_i J_{T_i}^{-1}$ is less than $\frac{1}{2}$.

Lemma 4.3.4 : *Consider* $J \succ 0$ *and a sequence of P-regular graphical splittings* $\{J = J_{T_{u_t}} - K_{u_t}\}_{t=1}^{\infty}$. *If the splittings are chosen from a finite number of distinct graphical splittings* $\{J = J_{T_i} - K_i\}_{i=1}^{N}$, *then there exists a positive integer* p *depending only on* J *such that*

$$|| \prod_{i=m}^{p+m-1} \left(K_{u_i} J_{T_{u_i}}^{-1}\right) ||_{J\to J} < \frac{1}{2}$$

for any positive integer m.

Proof of Theorem 4.3.1

Proof. The necessity is easy to prove since for Algorithm 4.3.1 to proceed, the noise at each iteration needs to be valid, which implies the convergence with each of the splittings according to Proposition 4.2.1.

Now we prove the sufficiency. Similarly as in the proof of Theorem 4.2.3, we use $\boldsymbol{\mu}^{(t)}$ and $\Sigma^{(t)}$ to represent the mean and covariance matrix of the sample distribution at iteration t. From Step 2 of Algorithm 4.3.1, we can prove that

$$\boldsymbol{\mu}^{(t+1)} - \boldsymbol{\mu} = J_{T_t}^{-1} K_t \left(\boldsymbol{\mu}^{(t)} - \boldsymbol{\mu}\right) \tag{4.20}$$

and

$$\Sigma^{(t+1)} - \Sigma = \left(J_{T_t}^{-1} K_t\right) \left(\Sigma^{(t)} - \Sigma\right) \left(J_{T_t}^{-1} K_t\right)^T. \tag{4.21}$$

From Lemma 4.3.3, we have that $||J_{T_t}^{-1} K_t||_{J \to J} < 1$. Since $\mathcal{S}$ is a finite collection of splittings, let $\sigma_{\max} = \max_{i \in \mathcal{S}} ||J_{T_i}^{-1} K_i||_{J \to J} < 1$. Hence

$$||\boldsymbol{\mu}^{(t)} - \boldsymbol{\mu}||_J = ||\prod_{i=1}^{t} \left(J_{T_i}^{-1} K_i\right) \left(\boldsymbol{\mu}^{(0)} - \boldsymbol{\mu}\right) ||_J \tag{4.22}$$

$$\leq \prod_{i=1}^{t} ||J_{T_i}^{-1} K_i||_{J \to J} ||\boldsymbol{\mu}^{(0)} - \boldsymbol{\mu}||_J \tag{4.23}$$

$$\leq \sigma_{\max}^t ||\boldsymbol{\mu}^{(0)} - \boldsymbol{\mu}||_J. \tag{4.24}$$

Similarly,

$$||\Sigma^{(t)} - \Sigma||_{J \to J} = \left\| \left(\prod_{i=1}^{t} \left(J_{T_i}^{-1} K_i\right)\right) \left(\Sigma^{(0)} - \Sigma\right) \left(\prod_{i=1}^{t} \left(J_{T_i}^{-1} K_i\right)\right)^T \right\|_{J \to J} \tag{4.25}$$

$$\leq \left\| \prod_{i=1}^{t} \left(J_{T_i}^{-1} K_i\right) \right\|_{J \to J} \left\| \Sigma^{(0)} - \Sigma \right\|_{J \to J} \left\| \prod_{i=1}^{t} \left(K_i J_{T_i}^{-1}\right) \right\|_{J \to J} \tag{4.26}$$

Let p be the integer in Lemma 4.3.4. Then when $t > p$, we have that

$$||\Sigma^{(t)} - \Sigma||_{J\to J} \leq \sigma_{\max}^t \left\|\Sigma^{(0)} - \Sigma\right\|_{J\to J} C, \tag{4.27}$$

where $C = \max\{\delta_{\max}, \delta_{\max}^{p-1}, \frac{1}{2}\} \geq 0$ and $\delta_{\max} = \max_{i\in\mathcal{S}} ||K_i J_{T_i}^{-1}||_{J\to J}$.

For a positive definite symmetric matrix J of finite dimension, there exist $0 < D_1 \leq D_2$ such that $D_1||\boldsymbol{v}||_2 \leq ||\boldsymbol{v}||_J \leq D_2||\boldsymbol{v}||_2$ for any vector $\boldsymbol{v}$ and $0 < C_1 \leq C_2$ such that $C_1||A||_F \leq ||A||_{J\to J} \leq C_2||A||_F$ for any matrix A. Hence, we have that $||\boldsymbol{\mu}^{(t)} - \boldsymbol{\mu}||_2 \leq \frac{D_2}{D_1}\sigma_{\max}^t||\boldsymbol{\mu}^{(0)} - \boldsymbol{\mu}||_2$ and $||\Sigma^{(t)} - \Sigma||_F \leq \frac{C_2 C}{C_1}\sigma_{\max}^t||\Sigma^{(t)} - \Sigma||_F$. Therefore, Algorithm 4.3.1 converges using this sequence of splittings. This concludes the proof of Theorem 4.3.1.

□

Alternative Proof of Corollary 1 in [33]

One of the main results in [33] states that if the graphical model is walk-summable then the embedded tree algorithm converges to the correct mean using any sequence of graphical splittings $\{J = J_{T_t} - K_t\}$ where each J_{T_t} corresponds to a tree-structured graph and each K_t corresponds to the cut edges and has zero diagonal. The original proof in [33] uses walk-sum diagrams. Here we give an alternative proof using results presented in this section.[2]

Proof. Consider the splittings used in [33], where K_t has zero diagonal and the nonzero off-diagonal entries of K_t take the opposite values of the corresponding entries in J. We define $J_t^* = J_{T_t} + K_t$ and thus the entries in J_t^* have the same absolute values as the corresponding entries in J. Since J is walk-summable, we have that J_t^* is also walk-summable by the definition of walk-summability (c.f. [54]). Since walk-summability implies the validity of a model, we have that $J_t^* \succ 0$. By Lemma 4.3.3, we have that $||J_{T_t}^{-1}K_t||_{J\to J} < 1$ for all t. Since there are a finite number of different splittings in this setting, we can show the convergence using the same arguments as in the proof of Theorem 4.3.1.

□

[2]Note that our sampling algorithm requires additional constraints to ensure the validity of the added noise. It is coincidental that the results in this chapter lead to an alternative proof of one of the main results in [33].

4.4 The Selection of Tractable Subgraphs

In this section, we discuss the selection of tractable subgraphs. First, we discuss how to choose graph structures for stationary splittings; then, we propose an algorithm to adaptively select tractable subgraphs for non-stationary splittings.

4.4.1 Select Subgraph Structures for Stationary Splittings

Using Tree-Structured Subgraphs

From the inequalities in Corollary 4.2.5, a heuristic is to choose K with small absolute edge weights and at the same time ensure the rest of the graph is tree-structured. Hence, the tree-structured subgraph is encouraged to contain strong edges. An effective method is to find the maximum spanning tree (MST) with edge weights w_{ij} being the absolute values of the normalized edge weights in J, i.e., $w_{ij} = |J_{ij}|/\sqrt{J_{ii}J_{jj}}$. The idea of using an MST has been discussed in the support graph preconditioner literature [68] as well as in the studies of the embedded tree algorithm for inference [33]. An MST can be constructed using Kruskal's algorithm in $\mathcal{O}(m \log n)$ time, where m is the number of edges. This selection procedure is summarized in Algorithm 4.4.1.

Algorithm 4.4.1 Selecting a Tree-Structured Subgraph

Input: $J \succ 0$

Output: a tree-structured subgraph $\mathcal{T}$

1. Compute the normalized edge weights $w_{ij} = |J_{ij}| / \sqrt{J_{ii}J_{jj}}$ for all $(i,j) \in \mathcal{E}$.
2. Compute the maximum spanning tree $\mathcal{T}$ using edge weights w_{ij}.

In our perturbation sampling framework, the tractable subgraphs can be structures beyond trees. Here we also suggest several other tractable graph structures with existing efficient inference and sampling algorithms.

Using Subgraphs with Low Tree-width

Graphical models with low tree-width have efficient inference and sampling algorithms and have been widely studied. We can compute a low tree-width approximation J_T to

J using algorithms such as those in [69, 70, 40].

Using Subgraphs with Small FVSs

As mentioned in Section ??, an FVS is a set of nodes whose removal results in a cycle-free graph and a pseudo-FVS is a set of nodes that breaks most, but not all, of the cycles in the graph. The FMP algorithm described in Section ?? provides a tractable inference algorithm for graphical models with small FVSs. This allows us to consider a graph with a small FVS as the tractable subgraph in the framework developed in this chapter. We can first use Algorithm 2.2.1 to select a set of nodes $\mathcal{F}$ constituting a pseudo-FVS for the full graph. Then we compute a MST among the other nodes. We choose our subgraph to be the combination of nodes $\mathcal{F}$ (with all incident edges) as well as the MST of the remaining graph. Note that even though $\mathcal{F}$ is a pseudo-FVS of the original graph, it is a true FVS of the subgraph, and therefore the algorithm from [16] and Section ?? provides exact inference. Using this technique, there is a trade-off in choosing the size of $\mathcal{F}$: a larger set $\mathcal{F}$ means more computation per iteration but faster convergence.

Using Spectrally Sparsified Subgraphs

Many widely used GGMs such as thin-membrane or thin-plate models have diagonally dominant information matrices. Some recent studies have shown that the graph Laplacian of a dense graph can be well-approximated by the graph Laplacian of graphs with nearly-linear number of edges [71]. These spectrally sparsified graphs have efficient inference and sampling algorithms and can also be used as tractable subgraphs.

■ 4.4.2 Adaptive Selection of Graph Structures for Non-Stationary Splittings

In this subsection, we propose an algorithm to adaptively select the structure of the subgraphs for non-stationary splittings. We explain our algorithm assuming that each subgraph is tree-structured, but this algorithm can be extended to other tractable subgraphs such as those mentioned in the previous subsection.

From Algorithm 4.3.1, it can be shown that

$$\boldsymbol{\mu} - \boldsymbol{\mu}^{(t+1)} = (J^{-1} - J_{T_t}{}^{1}) \left(\mathbf{h} - J\boldsymbol{\mu}^{(t)} \right), \tag{4.28}$$

which characterizes the residual error for the mean. Similarly, for the sample covariance, we have

$$\Sigma - \Sigma^{(t+1)} = \left(J^{-1} - J_{T_t}^{-1}\right)\left(J - J\Sigma^{(t)}J\right)\left(J^{-1} - J_{T_t}^{-1}\right)^T. \tag{4.29}$$

In [33] the authors have proposed an adaptive method using the walk-sum analysis framework: at each iteration $t+1$, choose the MST $\mathcal{T}_t$ in (4.28) with weights $\delta_{u,v}^{(t)}$ for edge (u,v), where

$$\delta_{u,v}^{(t)} = \left(|h_u^{(t)}| + |h_v^{(t)}|\right)\frac{|J_{u,v}|}{1-|J_{u,v}|} \tag{4.30}$$

and $\mathbf{h}^{(t)} = \mathbf{h} - J\boldsymbol{\mu}^{(t)}$.[3] This adaptive method significantly improves the speed of convergence for inference compared with using stationary splittings. In our case of sampling, both the error for the mean and the error for the covariance matrix need to be considered. However, a similar relaxation for the covariance matrix based on (4.29) is too computationally costly. Hence, we resort to an auxiliary inference problem with the same information matrix J and the potential vector $\mathbf{h}^*$ being the all-one vector. At each iteration of our sampling algorithm, we use the subgraph adaptively selected based on the auxiliary inference algorithm (i.e., choosing the MST with weight as in (4.30) but using the potential vector $\mathbf{h}^*$).

4.5 Experimental Results

In this section, we present experimental results using our perturbation sampling algorithms with both stationary graphical splittings and non-stationary graphical splittings. In the first two sets of experiments, we use simulated models on grids of various sizes; in the third example, we use standard test data of a power network of moderate size; finally, we present results using a large-scale real example for sea surface temperature estimation.

4.5.1 Motivating Example: 3×10 Grids

In this motivating example, we consider a simple 3×10 grid (Figure 4.2a). In the simulated models, the model parameters J and $\mathbf{h}$ are randomly generated as follows:

[3]Note that here the matrix J is normalized to have unit diagonal.

the entries of the potential vector $\mathbf{h}$ are generated *i.i.d.* from a uniform distribution $U[-1,1]$; the sparsity pattern of J is determined by the graph structure and the non-zero entries of J are also generated *i.i.d.* from $U[-1,1]$ with a multiple of the identity matrix added to ensure $J \succ 0$. We compare several sampling algorithms, namely basic Gibbs sampling, chessboard (red-black) blocked Gibbs sampling (Figure 4.2b), forest Gibbs sampling (Figure 4.2c, c.f. [29]), and our algorithm using a stationary splitting (Figure 4.2d) selected with Algorithm 4.4.1 (listed as "1-Tree Perturbation" in Table 4.1). We randomly generate 100 sets of model parameters and compute the asymptotic convergence rates. The average numbers of iterations (to reduce the covariance error in half), i.e., the average $\log_2 \tau_\Sigma$, are shown in Table 4.1.

We also study the convergence rates using non-stationary splittings. For each generated model, we run Algorithm 4.3.1 for 20 iterations and obtain 20 tree-structured subgraphs adaptively selected using (4.30). Figure 4.3 shows the first four tree-structured subgraphs adaptively selected on one of the generated models. We summarize the asymptotic convergence rates in Table 4.2 for the following six cases: 1) the single tree that gives the best convergence among the 20 trees[4]; 2) the worst single tree of the 20 trees; 3) alternating between the best pair of trees (by an exhaustive search among all pairs of the 20 trees); 4) alternating between the worst pair of trees; 5) using the first two adaptively selected trees (and alternating between them); and 6) using adaptively selected trees at each of the 20 iterations. From the results, we can see that using different subgraph structures give significantly different performances. On average, the best single tree can reduce the residual covariance error in half in 6 iterations while the worst single tree takes 88 iterations. The best combination of two trees gives the best convergence rate, but is included only as a benchmark, as exhaustive search is not computationally feasible in practice. Using the sequence of adaptively selected trees gives the second best performance while having much less computational complexity. The sampling algorithm with non-stationary graphical splittings outperforms its stationary counterpart even using the best single tree, which demonstrates the advantages of using non-stationary graphical splittings for sampling.

[4]The number of all spanning trees of a grid is very large (there are more than 9.41×10^9 spanning trees even for this small 3×10 grid, computed using recursive equations in [72]), which makes it intractable to do exhaustive search among all spanning trees. In addition, for a fair comparison with the adaptive method, the single tree is chosen from the 20 adaptively selected trees.

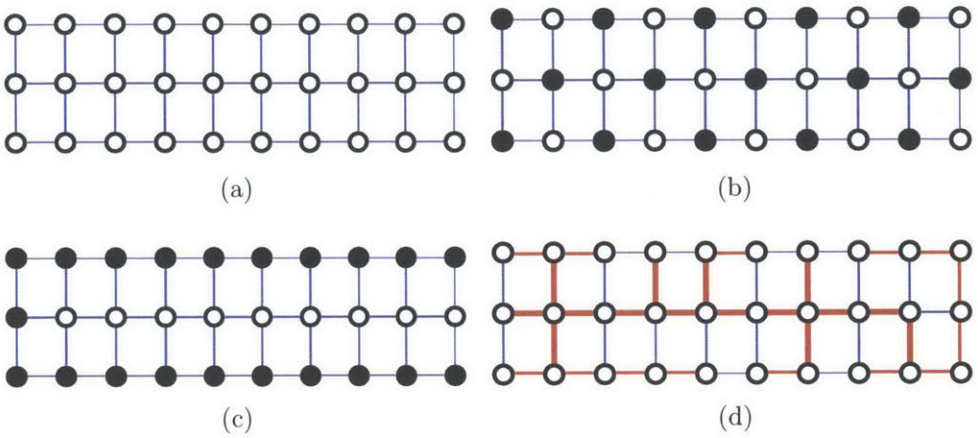


Figure 4.2: Sampling from a 3×10 grid using basic Gibbs sampling, chessboard (red-black) Gibbs sampling, forest Gibbs sampling, and our subgraph perturbation sampling using a stationary splitting. a) Graph structure of the 3×10 grid; b) Chessboard (red-black) blocked Gibbs sampling: the set of black nodes and the set of white nodes form two blocks; c) Forest Gibbs sampling: the set of black nodes and the set of white nodes form two separate trees. At each iteration of the forest Gibbs sampling, conditioned on one block, the other block is sampled by forward sampling; d) Subgraph perturbation sampling using a fixed tree-structured subgraph: the thicker red edges are edges in the tree-structured subgraph while the thinner blue edges are edges in the cut matrix.

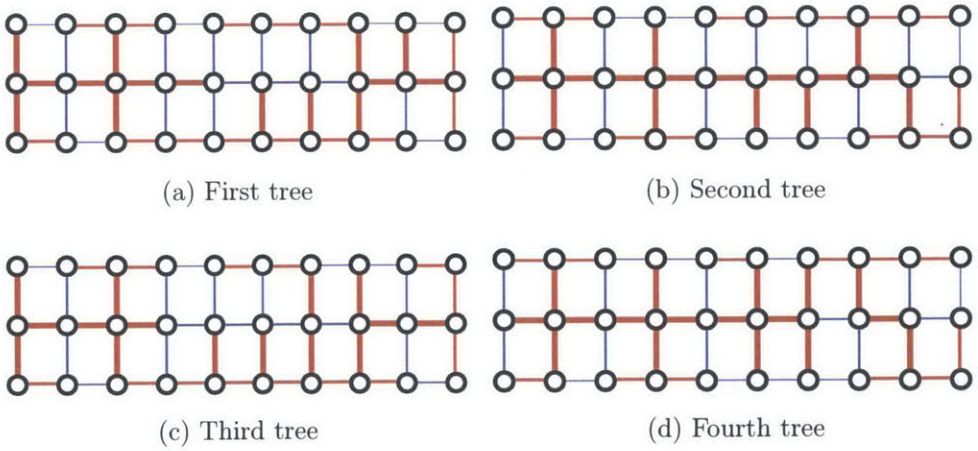


Figure 4.3: Sampling from a 3×10 grid using non-stationary splittings. (a)–(d) show the first four trees adaptively selected using (4.30) on one run.

	Average number of iteration (to reduce the covariance error in half)
Gibbs	42.842
Chessboard Gibbs	42.842
Forest Gibbs	18.846
1-Tree Perturbation	5.967

Table 4.1: Convergence rates of various sampling algorithms

	Average number of iteration (to reduce the covariance error in half)
Best single tree of the first 20 trees	5.4365
Worst single tree of the first 20 trees	87.397
Best pair of trees of the first 20 trees	3.6513
Worst pair of trees of the first 20 trees	87.397
The first two trees adaptively selected trees	5.5236
All of the 20 adaptively selected trees	4.9719

Table 4.2: Convergence rates of subgraph perturbation using non-stationary graphical splittings

4.5.2 Using Subgraphs Beyond Trees

In this experiment, we study the convergence rates using different subgraph structures on grids of various sizes. For each given structure, we randomly generate model parameters using the same method as in Subsection 4.5.1. We compute the numbers of iterations needed to achieve an approximating error of $\epsilon = 10^{-5}$, i.e., the minimum t such that $\left\|\Sigma^{(t)} - \Sigma\right\|_F \leq \epsilon$. We run the subgraph perturbation algorithm on l-by-l grids with l ranging from 3 to 30. For each grid, two different subgraphs are used: one is a tree-structured subgraph, the other is a subgraph with an FVS of size $\lceil \log l^2 \rceil$. For each size, we repeat the algorithm for 100 sets of random model parameters and the results shown are averaged over the 100 runs. Since the sizes of the simulated models are moderate, we are able to compute and compare with the exact solutions. As we can see from Figure 4.4, our subgraph perturbation algorithm outperforms the Gibbs sampler and the use of subgraphs with small FVSs gives further improvement on convergence

rate.[5]

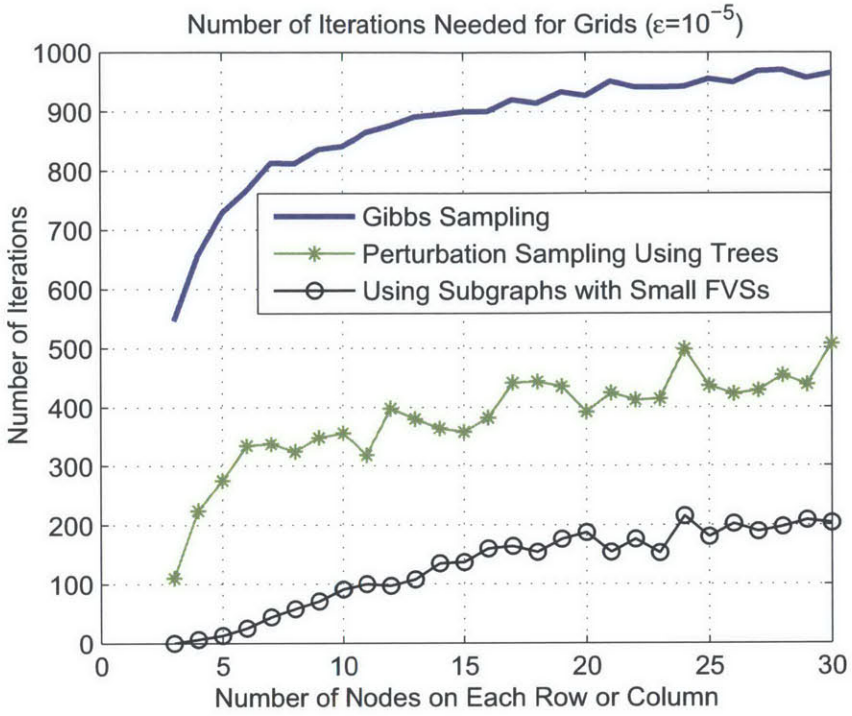


Figure 4.4: The performance of subgraph perturbation sampling using various kinds of subgraphs on grids of size 3-by-3 to 30-by-30. The tractable subgraphs used include tree-structured graphs and graphs with small FVSs.

■ 4.5.3 Power System Network: Standard Test Matrix 494 BUS

In this subsection, we use standard test data from the Harwell-Boeing Sparse Matrix Collection, which includes standard test matrices arising from a wide variety of scientific and engineering disciplines. We use the test matrix corresponding to a moderately sized (494 nodes) power system network[6]. We first add a multiple of the identity matrix to make the matrix positive definite and then normalize the matrix to have unit diagonal. Note that a diagonally dominant covariance matrix is easy to sample from (consider the extreme case of a diagonal matrix, which corresponds to independent Gaussian

[5]Note that more computation is involved at each iteration using FMP, but the complexity grows slowly if, as in this example, we use FVSs of sizes that are logarithmic in the size of the overall graph.

[6]The test matrix can be obtained from http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/psadmit/494_bus.html.

	Number of iterations (to reduce the covariance error in half)
Gibbs sampling (Gibbs)	32653
Subgraph perturbation with a tree (Embedded Tree)	3491
Subgraph perturbation sampling with a 1-FVS subgraph (1-FVS)	3452
Subgraph perturbation sampling with a 3-FVS subgraph (3-FVS)	2500
Subgraph perturbation sampling with a 5-FVS subgraph (5-FVS)	1944

Table 4.3: Convergence rates using a single tree and subgraphs with FVS of various sizes

variables) even with the basic Gibbs sampler, but they do not represent many real applications. Hence, in order to study the models that are challenging for the Gibbs sampler or other common algorithms (which is the scenario that we focus on in this chapter), we add just enough diagonal loading to make the matrix positive definite. We compare the performances of Gibbs sampling, subgraph perturbation sampling using a tree-structured subgraph and using subgraphs with FVSs of sizes one, three and five. In this experiment, we focus on stationary splittings since we are interested in comparing the performances using different types of subgraphs. The experimental results are shown in Table 4.3 and Figure 4.5. As these results show, for this problem using a single tree subgraph reduces the number of iterations needed to achieve 50% error reduction by almost an order of magnitude, and using a very small size-5 FVS cuts the number down significantly further.

4.5.4 Large-Scale Real Example: Sea Surface Temperature

We also run the algorithm on a large-scale GGM built to estimate the sea surface temperature (the dataset is publicly available at http://podaac.jpl.nasa.gov/dataset/). The data are preprocessed to have raw measurements at 720×1440 different locations. We construct a grid of 1,036,800 nodes with additional edges connecting the eastmost and westmost nodes at the same latitudes since they are neighbors geographically. We

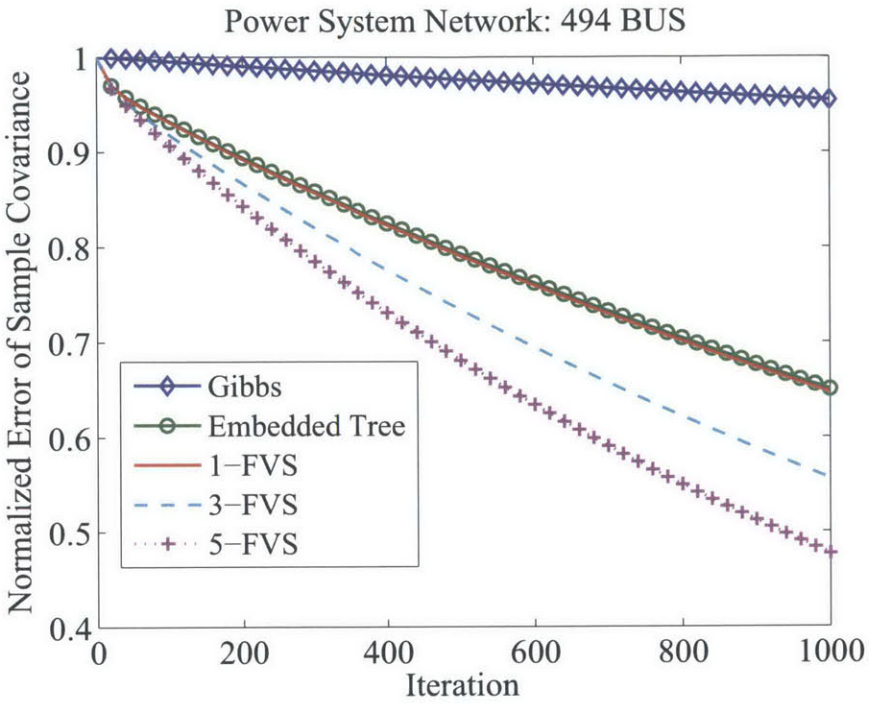
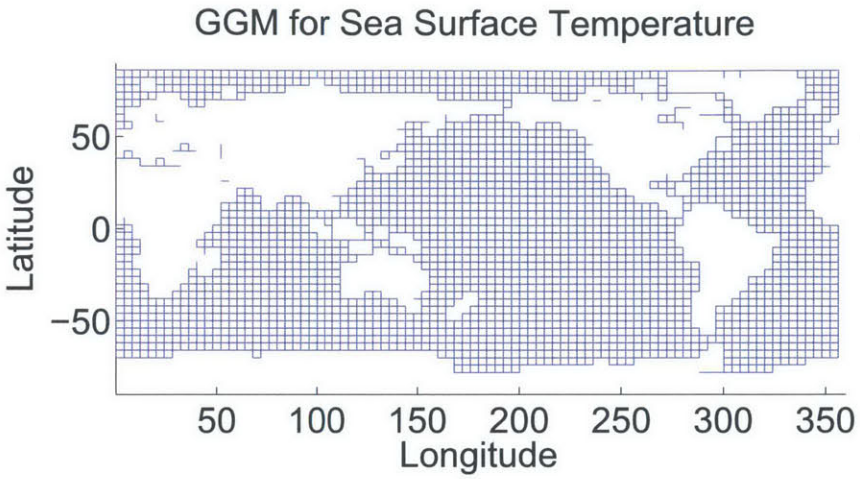
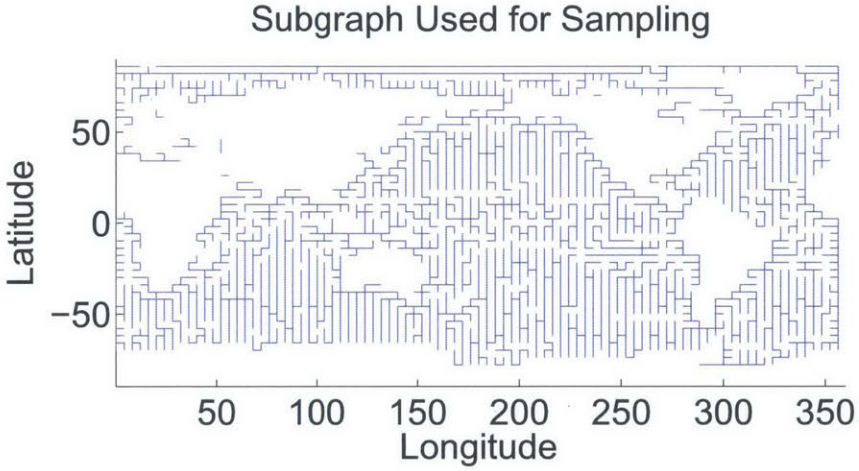


Figure 4.5: Perturbation sampling using various subgraph structures on a power system network. The normalized error of the sample covariance is defined as the ratio between the sample covariance error at each iteration and the initial covariance error.

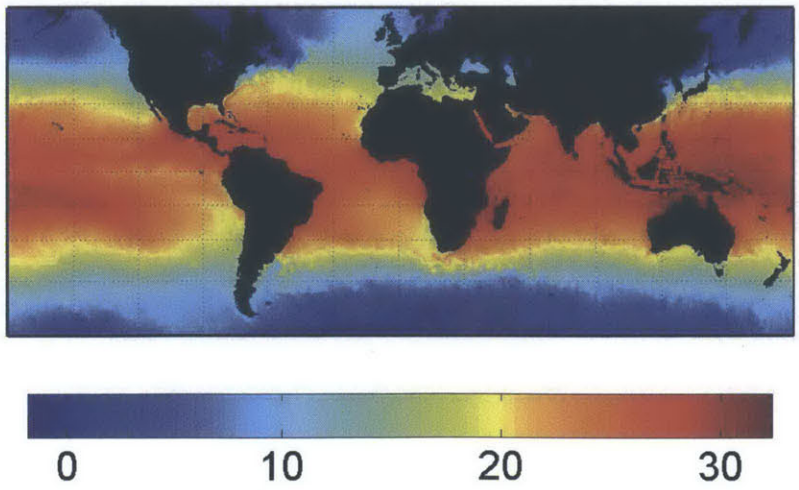
then remove the nodes that have invalid measurements (most of which correspond to land areas). We construct a GGM with this underlying structure using the thin-plate model [1]. Note that because of the significant number of observations, the information matrix for this model is far better conditioned than the one in the preceding section, implying that far fewer iterations are needed to reach approximate convergence. The structure of the resulting model is shown in Figure 4.6a and the tractable subgraph used for our perturbation sampling algorithm is shown in Figure 4.6b (for clarity, we plot a much coarser version and omit the edges connecting the eastmost and westmost nodes). A sample from the posterior distribution after 200 iterations is shown in Figure 4.6c.



(a) The entire GGM for sea surface temperature



(b) The spanning tree used as a tractable subgraph



(c) Sea surface temperature in degrees (Celsius)

Figure 4.6: Perturbation sampling from a GGM for sea surface temperature estimation

4.6 Appendix for Chapter 4

Proof of Proposition 4.2.1

Proposition 4.2.1 : *Assuming $J \succ 0$ and that $J = J_T - K$ is a graphical splitting, the condition $\rho(J_T^{-1}K) < 1$ is satisfied if and only if the splitting is P-regular, i.e., the added noise in Algorithm 4.2.1 has a valid covariance matrix $J_T + K \succ 0$.*

Proof. We first prove the sufficiency. If $J_T + K \succ 0$, then $2J_T - J \succ 0$ and thus $J_T \succ 0$. Hence, $J_T^{-1} \succ 0$ has a unique positive definite square root $J_T^{-\frac{1}{2}} \succ 0$. Then we have

$$0 \prec J_T^{-\frac{1}{2}} J J_T^{-\frac{1}{2}} = J_T^{-\frac{1}{2}} (J_T - K) J_T^{-\frac{1}{2}} = I - J_T^{-\frac{1}{2}} K J_T^{-\frac{1}{2}}. \tag{4.31}$$

Hence, $\lambda_i \left(J_T^{-\frac{1}{2}} K J_T^{-\frac{1}{2}} \right) < 1$, for all i, where $\lambda_i(\cdot)$ denotes the i-th eigenvalue of the argument. From the condition $J_T + K \succ 0$, we have that

$$I + J_T^{-\frac{1}{2}} K J_T^{-\frac{1}{2}} = J_T^{-\frac{1}{2}} (J_T + K) J_T^{-\frac{1}{2}} \succ 0, \tag{4.32}$$

and thus $\lambda_i(J_T^{-\frac{1}{2}} K J_T^{-\frac{1}{2}}) > -1$, for all i. Because $J_T^{-1}K = J_T^{-\frac{1}{2}} \left(J_T^{-\frac{1}{2}} K J_T^{-\frac{1}{2}} \right) J_T^{\frac{1}{2}}$, we have that $J_T^{-1}K$ has the same eigenvalues as $J_T^{-\frac{1}{2}} K J_T^{-\frac{1}{2}}$. Therefore, $\left| \lambda_i \left(J_T^{-1} K \right) \right| < 1$ for all i and thus $\rho(J_T^{-1}K) < 1$.

We now prove the necessity. If $\rho(J_T^{-1}K) < 1$, then $I - J_T^{-1}K = J_T^{-1}J$ has positive eigenvalues. Since $J \succ 0$, J has a unique positive definite square root $J^{-\frac{1}{2}} \succ 0$, and thus

$$0 \prec J^{\frac{1}{2}} J_T^{-1} J^{\frac{1}{2}} = J^{\frac{1}{2}} \left(J_T^{-1} J \right) J^{-\frac{1}{2}}. \tag{4.33}$$

So we have $J_T^{-1} \succ 0$. Hence $J_T \succ 0$ has a unique positive definite square root $J_T^{\frac{1}{2}} \succ 0$. So $J_T^{-\frac{1}{2}} K J_T^{-\frac{1}{2}}$ has the same eigenvalues as $J_T^{-1}K$ since $J_T^{-\frac{1}{2}} K J_T^{-\frac{1}{2}} = J_T^{\frac{1}{2}} \left(J_T^{-1} K \right) J_T^{-\frac{1}{2}}$, and thus $\rho(J_T^{-\frac{1}{2}} K J_T^{-\frac{1}{2}}) < 1$. Hence, $I + J_T^{-\frac{1}{2}} K J_T^{-\frac{1}{2}} \succ 0$, so

$$J_T + K = J_T^{\frac{1}{2}} \left(I + J_T^{-\frac{1}{2}} K J_T^{-\frac{1}{2}} \right) J_T^{\frac{1}{2}} \succ 0. \tag{4.34}$$

Therefore, $J = J_T - K$ is a P-regular splitting.

□

Proof of Lemma 4.2.2

Lemma 4.2.2 : *Let A and B be square matrices. If 1) A is invertible; 2) $A+B$ is symmetric and invertible, then $\Sigma = (A+B)^{-1}$ is a solution of the equation $A\Sigma A^T = B\Sigma B^T + A^T - B$.*

Proof. It is equivalent to showing

$$A(A+B)^{-1}A^T = B(A+B)^{-1}B^T + A^T - B. \tag{4.35}$$

To do so, consider

$$\begin{aligned}
\text{LHS} =& \left((A+B)-B\right)(A+B)^{-1}A^T && (4.36)\\
=& A^T - B(A+B)^{-1}A^T && (4.37)\\
=& A^T - B(A+B)^{-1}\left((A^T+B^T)-B^T\right) && (4.38)\\
=& A^T - B(A+B)^{-1}(A+B)^T + B(A+B)^{-1}B^T && (4.39)\\
\overset{(a)}{=}& A^T - B + B(A+B)^{-1}B^T && (4.40)\\
=& \text{RHS}, && (4.41)
\end{aligned}$$

where (a) is due to the assumption that $A+B$ is symmetric.

□

Proof of Lemma 4.3.2

Lemma 4.3.2 : *If $J \succ 0$ and the graphical splitting $J = J_T - K$ is P-regular, then there exists $\epsilon > 0$ such that $J - \left(J_T^{-1}K\right)^T J \left(J_T^{-1}K\right) \succeq \epsilon J$.*

Proof. Since $J \succ 0$, there exists some $\delta_h \geq \delta_l > 0$ such that $\delta_h I \succ J \succ \delta_l I$. Hence, to prove Lemma 4.3.2, it is sufficient to show that there exists $\tilde{\epsilon} > 0$ such that $J - \left(J_T^{-1}K\right)^T J \left(J_T^{-1}K\right) \geq \tilde{\epsilon} I$, which is equivalent to showing that $J - \left(J_T^{-1}K\right)^T J \left(J_T^{-1}K\right) \succ 0$.

$$J - \left(J_T^{-1}K\right)^T J \left(J_T^{-1}K\right) \succ 0 \tag{4.42}$$

$$\Leftrightarrow J - \left(I - J_T^{-1}J\right)^T J \left(I - J_T^{-1}J\right) \succ 0 \tag{4.43}$$

$$\Leftrightarrow J - \left(J + JJ_T^{-1}JJ_T^{-1}J - 2JJ_T^{-1}J\right) \succ 0 \tag{4.44}$$

$$\Leftrightarrow 2JJ_T^{-1}J - JJ_T^{-1}JJ_T^{-1} \succ 0 \tag{4.45}$$

$$\overset{(a)}{\Leftrightarrow} \left(J^{-1}J_T\right)^T \left(2JJ_T^{-1}J - JJ_T^{-1}JJ_T^{-1}J\right) \left(J^{-1}J_T\right) \succ 0 \tag{4.46}$$

$$\Leftrightarrow 2J_T - J \succ 0 \tag{4.47}$$

$$\Leftrightarrow J_T + K \succ 0, \tag{4.48}$$

where (a) is due to that J and J_T are both non-singular since $J \succ 0$ and $J_T = \frac{J+(J_T+K)}{2} \succ 0$.

□

Proof of Lemma 4.3.3

Lemma 4.3.3 : *If $J \succ 0$ and $J = J_T - K$ is a P-regular graphical splitting, then $||J_T^{-1}K||_{J\to J} < 1$.*

Proof. For any $u \neq 0$, we have that

$$\left(J_T^{-1}Ku\right)^T J \left(J_T^{-1}Ku\right) \tag{4.49}$$

$$= u^T \left(\left(J_T^{-1}K\right)^T J \left(J_T^{-1}K\right)\right) u \tag{4.50}$$

$$= u^T \left(\left(J_T^{-1}K\right)^T J \left(J_T^{-1}K\right) - J\right) u + u^T Ju \tag{4.51}$$

From Lemma 4.3.2, there exist $\epsilon > 0$ such that $\left(J_T^{-1}K\right)^T J \left(J_T^{-1}K\right) \preceq (1-\epsilon)J$. Hence, we have $\left(J_T^{-1}Ku\right)^T J \left(J_T^{-1}Ku\right) \preceq (1-\epsilon)u^T Ju$, i.e., $||J_T^{-1}Ku||_J \preceq (1-\epsilon)||u||_J$. Thus for any $u \neq 0$, $\frac{\left(J_T^{-1}Ku\right)^T J\left(J_T^{-1}Ku\right)}{u^T Ju} \leq (1-\epsilon) < 1$. Hence, by the definition of the J-induced norm, we have that $||J_T^{-1}K||_{J\to J} < 1$.

□

Proof of Lemma 4.3.4

Lemma 4.3.4 : *Consider $J \succ 0$ and a sequence of P-regular graphical splittings $\{J = J_{T_{u_t}} - K_{u_t}\}_{t=1}^{\infty}$. If the splittings are chosen from a finite number of distinct graphical splittings $\{J = J_{T_i} - K_i\}_{i=1}^{N}$, then there exists a positive integer p depending only on J such that*

$$|| \prod_{i=m}^{p+m-1} \left(K_{u_i} J_{T_{u_i}}^{-1}\right) ||_{J \to J} < \frac{1}{2} \tag{4.52}$$

for any positive integer m.

Proof. Since the sequence is arbitrary, without loss of generality, we only need to prove for $m = 1$. Since $||\cdot||_{J\to J}$ is an induced norm, there exists $0 < C_1 \leq C_2$ depending only on J such that $C_1||A||_F \leq ||A||_{J\to J} \leq C_2||A||_F$ for any square matrix A. From Lemma 4.3.3, $||J_{T_i}^{-1}K_i||_{J\to J} < 1$ for all i. Since there are finitely many distinct splittings, there exists $0 \leq \sigma_{\max} < 1$ such that $||J_{T_i}^{-1}K_i||_{J\to J} \leq \sigma_{\max} < 1$ for all i. For induced norms, it can be shown that $||AB||_{J\to J} \leq ||A||_{J\to J}||B||_{J\to J}$. Hence, there exists integer p depending only on J such that $||\prod_{i=p}^{1} J_{T_{u_i}}^{-1}K_{u_i}||_{J\to J} \leq \prod_{i=p}^{1} ||J_{T_{u_i}}^{-1}K_{u_i}|| \leq \sigma_{\max}^{p} \leq \frac{C_1}{2C_2}$. Since the Frobenius norm is invariant to transposition, we have that $||\prod_{i=1}^{p}\left(J_{T_{u_i}}^{-1}K_{u_i}\right)^T||_F = ||\prod_{i=p}^{1} J_{T_{u_i}}^{-1}K_{u_i}||_F$, and thus

$$\begin{aligned}
||\prod_{i=1}^{p}\left(K_{u_i}J_{T_{u_i}}^{-1}\right)||_{J\to J} &\leq C_2||\prod_{i=1}^{p}\left(K_{u_i}J_{T_{u_i}}^{-1}\right)||_F && (4.53)\\
&= C_2||\prod_{i=p}^{1}\left(J_{T_{u_i}}^{-1}K_{u_i}\right)||_F && (4.54)\\
&\leq \frac{C_2}{C_1}||\prod_{i=p}^{1}\left(J_{T_{u_i}}^{-1}K_{u_i}\right)||_{J\to J} && (4.55)\\
&\leq \frac{C_2}{C_1}\frac{C_1}{2C_2} && (4.56)\\
&= \frac{1}{2}. && (4.57)
\end{aligned}$$

This completes the proof. □

Chapter 5

Learning Gaussian Graphical Models with Small Feedback Vertex Sets

5.1 Introduction

As mentioned in Section 2.2.3, given a GGM with underlying distribution $\mathcal{N}^{-1}(\mathbf{h}, J)$ and with an FVS of size k, the marginal means and variances can be calculated *exactly* with computational complexity $\mathcal{O}(k^2 n)$ using the FMP algorithm proposed in [16], where standard BP is employed twice on the cycle-free subgraph among the non-feedback nodes while a special message-passing protocol is used for the FVS nodes. If we are not explicitly given an FVS, though the problem of finding an FVS of minimal size is NP-complete, the authors of [73] have proposed an efficient algorithm with computational complexity $\mathcal{O}(\min\{m \log n,\ n^2\})$, where m is the number of edges, that yields an FVS at most twice the minimum size (thus the inference complexity is increased only by a constant factor). As we will see, the complexity of such algorithms is manageable. Moreover, as our experiments will demonstrate, for many problems, quite modestly sized FVSs suffice.[1]

In this chapter, we study the family of GGMs with small FVSs. First in Section 5.2, we will provide some additional analysis of inference (i.e., the computation of the partition function) for such models, but the main focus in this chapter is ML *learning* of models with FVSs of modest size, including identifying the nodes to include in the FVS. Next in Section 5.3, we investigate the scenario where all of the variables, including any to be included in the FVS are observed. We provide an algorithm for exact ML

[1]For models with larger FVSs, approximate inference (obtained by replacing a full FVS by a pseudo-FVS) can work very well, with empirical evidence indicating that a pseudo-FVS of size $\mathcal{O}(\log n)$ gives excellent results.

estimation that, regardless of the maximum degree, has complexity $\mathcal{O}(kn^2 + n^2 \log n)$ if the FVS nodes are identified in advance and polynomial complexity if the FVS is to be learned and of bounded size. Moreover, we provide an approximate and much faster greedy algorithm when the FVS is unknown. In Section 5.4, we study the scenario where the FVS nodes are taken to be latent variables. The structure learning problem now corresponds to the (exact or approximate) decomposition of an inverse covariance matrix into the sum of a tree-structured matrix and a low-rank matrix. We propose an algorithm that iterates between two projections, which can also be interpreted as alternating *low-rank* corrections. We prove that even though the second projection is onto a highly non-convex set, it is carried out exactly, thanks to the properties of GGMs of this family. We also introduce an accelerated version that can further reduce the computational complexity to $\mathcal{O}(kn^2 + n^2 \log n)$ per iteration. Finally in Section 5.5, we perform experiments using both synthetic data and real data of flight delays to demonstrate the modeling capacity with FVSs of various sizes.

An important point to note is that the computational complexity of these inference and learning algorithms depends simply on the FVS size k and the number of nodes n. There is no loss in generality in assuming that the size-k FVS F is fully connected and that each of the feedback nodes has edges to every non-feedback node. In particular, after re-ordering the nodes so that the elements of F are the first k nodes ($T = \mathcal{V} \backslash F$ denotes the set of non-feedback nodes of size $n-k$), we have that $J = \begin{bmatrix} J_F & J_M^T \\ J_M & J_T \end{bmatrix} \succ 0$, where $J_T \succ 0$ corresponds to a tree-structured subgraph among the non-feedback nodes, $J_F \succ 0$ corresponds to a complete graph among the feedback nodes, and all entries of J_M may be non-zero as long as $J_T - J_M J_F^{-1} J_M^T \succ 0$ to ensure $J \succ 0$. Similarly, the covariance matrix is denoted as $\Sigma = \begin{bmatrix} \Sigma_F & \Sigma_M^T \\ \Sigma_M & \Sigma_T \end{bmatrix} = J^{-1} \succ 0$. We refer to the family of such models with a given FVS F as $\mathcal{Q}_F$. Note that in general a graph does not have a unique FVS and $\mathcal{Q}_F$ include all graphs that have F as *one* FVS. We refer to the class of models with some FVS of size at most k as $\mathcal{Q}_k$. The family of graphs with FVSs of size k includes all graphs where there *exists* an FVS of size k.

5.2 Computing the Partition Function of GGMs with Small FVSs

In graphical models, the computation of the partition function (c.f. Section 2.1 for definition) plays an important role in many problems [74]. Previously, various algorithms,

such as tree decomposition [74], variational methods [75], and Langevin Importance sampling [76], have been used to compute the partition function. In this section, we provide a new message-passing algorithm to compute $\det J$, the determinant of J, and hence the partition function of such a model, with computational complexity $\mathcal{O}(k^2 n)$.

The high-level idea of our algorithm is to factorize the determinant of the whole model into the product of (a) the determinant of the tree-structured subgraph $\mathcal{T}$ excluding the FVS; and (b) the determinant of the subgraph $\hat{\mathcal{F}}$ including only the FVS nodes but with modified structure and parameters.[2] Factor (a) can be computed using a tree decomposition method to be described in Lemma 5.2.2; and factor (b) is obtained by computing the determinant of $\hat{\mathcal{F}}$ directly, where the structure and parameters of $\hat{\mathcal{F}}$ are obtained using the FMP algorithm. In fact, the first round of LBP in FMP can be used to compute factor (a) and to obtain $\hat{\mathcal{F}}$ simultaneously. This algorithm is summarized in Algorithm 5.2.1.

Proposition 5.2.1 states the correctness and the computational complexity of Algorithm 5.2.1. The proof of Proposition 5.2.1 is deferred after we introduce Lemma 5.2.2.

Proposition 5.2.1 : *Algorithm 5.2.1 computes* $\det J$ *exactly and the computational complexity is* $\mathcal{O}(k^2 n)$.

The following Lemma 5.2.2 provides a factorization of the determinant of a tree-structured model, where the quantities in each factor can be computed using BP. The proof of Lemma 5.2.2 is provided in Appendix 5.7.

Lemma 5.2.2 : *If the information matrix* $J \succ 0$ *has tree structure* $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, *then we have*

$$\det(J)^{-1} = \prod_{i \in \mathcal{V}} P_{ii} \prod_{(i,j) \in \mathcal{E}} \frac{P_{ii}P_{jj} - P_{ij}^2}{P_{ii}P_{jj}}, \tag{5.1}$$

where $P = J^{-1}$.

[2] In fact, subgraph $\mathcal{T}$ corresponds to the submatrix J_T and subgraph $\hat{\mathcal{F}}$ corresponds to $J_F - J_M^T J_T^{-1} J_M$, the Schur complement of J_T in J.

Algorithm 5.2.1 Computing the Partition Function When an FVS Is Given

Input: an FVS F of size k and an $n \times n$ information matrix $J = \begin{bmatrix} J_F & J_M^T \\ J_M & J_T \end{bmatrix}$, where J_T has tree structure $\mathcal{T}$ with edge set $\mathcal{E}_T$.
Output: $\det J$

1. Run standard Gaussian BP on $\mathcal{T}$ with information matrix J_T to obtain $P_{ii}^{\mathcal{T}} = \left(J_T^{-1}\right)_{ii}$ for all $i \in T$, $P_{ij}^{\mathcal{T}} = (J_T^{-1})_{ij}$ for all $(i,j) \in \mathcal{E}_T$, and $(\mathbf{g}^p)_i = (J_T^{-1}\mathbf{h}^p)_i$ for all $i \in T$ and $p \in F$, where $\mathbf{h}^p$ is the column of J_M corresponding to node p.

2. Compute $\hat{J}_F$ with

$$\left(\hat{J}_F\right)_{pq} = J_{pq} - \sum_{j \in \mathcal{N}(p) \cap T} J_{pj} g_j^q, \ \forall\, p, q \in F$$

3. Compute $\det \hat{J}_F$

4. Output

$$\det J = \left(\prod_{(i,j) \in \mathcal{E}_T} \frac{P_{ii}^{\mathcal{T}} P_{jj}^{\mathcal{T}} - \left(P_{ij}^{\mathcal{T}}\right)^2}{P_{ii}^{\mathcal{T}} P_{jj}^{\mathcal{T}}} \prod_{i \in \mathcal{V}} P_{ii}^{\mathcal{T}} \right)^{-1} \det \hat{J}_{\mathcal{F}}.$$

Proof of Proposition 5.2.1

Proof. First, we show that $\hat{J}_F$ computed in Step 2 of Algorithm 5.2.1 equals $J_F - J_M^T J_T^{-1} J_M$. We have

$$\begin{bmatrix} \mathbf{g}^1 & \mathbf{g}^2 & \cdots & \mathbf{g}^k \end{bmatrix} = J_T^{-1} \begin{bmatrix} \mathbf{h}^1 & \mathbf{h}^2 & \cdots & \mathbf{h}^k \end{bmatrix} = J_T^{-1} J_M \tag{5.2}$$

from the definition in Step 1. From Step 2, we find that

$$\hat{J}_F = J_F - \begin{bmatrix} \mathbf{g}^1 & \mathbf{g}^2 & \cdots & \mathbf{g}^k \end{bmatrix}^T J_T \begin{bmatrix} \mathbf{g}^1 & \mathbf{g}^2 & \cdots & \mathbf{g}^k \end{bmatrix} \tag{5.3}$$

$$= J_F - \left(J_T^{-1} J_M\right)^T J_T \left(J_T^{-1} J_M\right) \tag{5.4}$$

$$= J_F - J_M^T J_T^{-1} J_M. \tag{5.5}$$

Hence,

$$\det J = \det\left(\begin{bmatrix} I & -J_M^T J_T^{-1} \\ \mathbf{0} & I \end{bmatrix}\right) \det\left(\begin{bmatrix} J_F & J_M^T \\ J_M & J_T \end{bmatrix}\right) \det\left(\begin{bmatrix} I & \mathbf{0} \\ -J_T^{-1} J_M & I \end{bmatrix}\right) \tag{5.6}$$

$$= \det\left(\begin{bmatrix} I & -J_M^T J_T^{-1} \\ \mathbf{0} & I \end{bmatrix} \begin{bmatrix} J_F & J_M^T \\ J_M & J_T \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ -J_T^{-1} J_M & I \end{bmatrix}\right) \tag{5.7}$$

$$= \det \begin{bmatrix} J_F - J_M^T J_T^{-1} J_M & \mathbf{0} \\ \mathbf{0} & J_T \end{bmatrix} \tag{5.8}$$

$$= \left(\det \hat{J}_F\right) \times (\det J_T), \tag{5.9}$$

According to Lemma 5.2.2, we have

$$\det (J_T)^{-1} = \prod_{i \in \mathcal{V}} P_{ii}^{\mathcal{T}} \prod_{(i,j) \in \mathcal{E}_T} \frac{P_{ii}^{\mathcal{T}} P_{jj}^{\mathcal{T}} - \left(P_{ij}^{\mathcal{T}}\right)^2}{P_{ii}^{\mathcal{T}} P_{jj}^{\mathcal{T}}}, \tag{5.10}$$

which establishes the correctness of Algorithm 5.2.1.

Now we calculate the computational complexity. The first step of Algorithm 5.2.1 has complexity $\mathcal{O}(n-k)$ using BP. Step 2 takes $\mathcal{O}\left(k^2(n-k)\right)$ and the complexity of Step 3 is $\mathcal{O}(k^3)$. Finally the complexity of Step 4 is $\mathcal{O}(n)$ since $\mathcal{T}$ is a tree. The total computational complexity is thus $\mathcal{O}(k^2 n)$. This completes the proof of Proposition 5.2.1. □

Note that if the FVS is not given in advance, we can use the factor-2 approximate algorithm in [73] to obtain an FVS of size at most twice the minimum size with complexity $\mathcal{O}(\min\{m \log n,\ n^2\})$, similarly as how we use the FMP algorithm for inference.

5.3 Learning GGMs with Observed FVSs

In this section, we study the problem of recovering a GGM from *i.i.d.* samples, where all nodes including the feedback nodes in F are observed. The empirical distribution $\hat{p}(\mathbf{x}_F, \mathbf{x}_T)$ is parametrized by the empirical covariance matrix $\hat{\Sigma} = \begin{bmatrix} \hat{\Sigma}_F & \hat{\Sigma}_M^T \\ \hat{\Sigma}_M & \hat{\Sigma}_T \end{bmatrix}$. We propose learning algorithms for the following two case: 1) When an FVS of size k is given, we propose the *conditioned Chow-Liu algorithm,* which computes the *exact* ML estimate efficiently; 2) When no FVS is given *a priori*, we propose both an exact

algorithm and a greedy approximate algorithm for computing the ML estimate. In the following analysis, with a slight abuse of notation, we use $q(\mathbf{x}_A)$ to denote the marginal distribution of $\mathbf{x}_A$ under a distribution $q(\mathbf{x}_\mathcal{V})$ for any subset $A \subset \mathcal{V}$.

5.3.1 Case 1: An FVS of Size k Is Given.

According to Proposition 2.4.1 in Section 2.4, the ML learning problem is equivalent to minimizing the K-L divergence between the empirical distribution and distributions in the family considered. When a size-k FVS F is given, we have

$$p_{\mathrm{ML}}(\mathbf{x}_F, \mathbf{x}_T) = \underset{q(\mathbf{x}_F, \mathbf{x}_T) \in \mathcal{Q}_F}{\arg\min} D_{\mathrm{KL}}(\hat{p}(\mathbf{x}_F, \mathbf{x}_T) || q(\mathbf{x}_F, \mathbf{x}_T)). \tag{5.11}$$

The following Lemma 5.3.1 gives a closed-form expression of the K-L divergence between two Gaussian distributions. Lemma 5.3.1 is a well-known result and its proof can be found in texts such as [58].

Lemma 5.3.1 : *For two n-dimensional Gaussian distributions $\hat{p}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \hat{\boldsymbol{\mu}}, \hat{\Sigma})$ and $q(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \Sigma)$, we have*

$$D(\hat{p}||q) = \frac{1}{2}\left(Tr(\Sigma^{-1}\hat{\Sigma}) + (\boldsymbol{\mu} - \hat{\boldsymbol{\mu}})^T \Sigma^{-1} (\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}) - n \ln \det\left(\Sigma^{-1}\hat{\Sigma}\right)\right). \tag{5.12}$$

An immediate implication of Lemma 5.3.1 is that when learning GGMs we always have that $\boldsymbol{\mu}_{\mathrm{ML}} = \hat{\boldsymbol{\mu}}$ if there is no other constraint on the mean. Therefore, in the following analysis of this chapter, without loss of generality, we assume both the empirical mean and the estimated mean are zero.

The optimization problem of (5.11) is defined on a highly non-convex set $\mathcal{Q}_F$ with combinatorial structure: indeed, there are $(n-k)^{n-k-2}$ possible spanning trees among the subgraph induced by the non-feedback nodes. However, we are still able to solve Problem (5.11) *exactly* using the conditioned Chow-Liu algorithm described in Algorithm 5.3.1.[3] The intuition behind this algorithm is that even though the entire graph

[3]Note that the conditioned Chow-Liu algorithm here is different from other variations of the Chow-Liu algorithm such as in [77] where the extensions are to enforce the inclusion or exclusion of a set of edges.

is not tree-structured, the subgraph induced by the non-feedback nodes (which corresponds to the distribution of the non-feedback nodes conditioned on the feedback nodes) has tree structure, and thus we can find the best tree among the non-feedback nodes using the Chow-Liu algorithm applied on the conditional distribution. To obtain a concise expression, we also exploit a property of Gaussian distributions: the conditional information matrix (the information matrix of the conditional distribution) is simply a submatrix of the whole information matrix and does not depend on the values of the conditioned variables. Hence, the integration over all variable values is not necessary in computing the conditional mutual information.

In Step 1 of Algorithm 5.3.1, we compute the conditional covariance matrix using the Schur complement, and then in Step 2 we use the Chow-Liu algorithm to obtain the best approximate Σ_{CL} (whose inverse is tree-structured). In Step 3, we match exactly the covariance matrix among the feedback nodes and the cross-covariance matrix between the feedback nodes and the non-feedback nodes. For the covariance matrix among the non-feedback nodes, we add the matrix subtracted in Step 1 back to Σ_{CL}. We denote the output covariance matrix of Algorithm 5.3.1 as $\mathrm{CCL}(\hat{\Sigma})$.

Algorithm 5.3.1 The Conditioned Chow-Liu Algorithm

Input: $\hat{\Sigma} \succ 0$ and an FVS F
Output: $\mathcal{E}_{\mathrm{ML}}$ and Σ_{ML}

1. Compute the conditional covariance matrix $\hat{\Sigma}_{T|F} = \hat{\Sigma}_T - \hat{\Sigma}_M \hat{\Sigma}_F^{-1} \hat{\Sigma}_M^T$.
2. Let $\Sigma_{\mathrm{CL}} = \mathrm{CL}(\hat{\Sigma}_{T|F})$ and $\mathcal{E}_{\mathrm{CL}} = \mathrm{CL}_{\mathcal{E}}(\hat{\Sigma}_{T|F})$.
3. $\mathcal{E}_{\mathrm{ML}} = \mathcal{E}_{\mathrm{CL}}$ and $\Sigma_{\mathrm{ML}} = \begin{bmatrix} \hat{\Sigma}_F & \hat{\Sigma}_M^T \\ \hat{\Sigma}_M & \Sigma_{\mathrm{CL}} + \hat{\Sigma}_M \hat{\Sigma}_F^{-1} \hat{\Sigma}_M^T \end{bmatrix}$.

The following Proposition 5.3.2 states the correctness and the complexity of Algorithm 5.3.1. The proof of Proposition 5.3.2 is provided after we states Lemma 5.3.3.

Proposition 5.3.2 : *Algorithm 5.3.1 computes the ML estimate Σ_{ML} with edge set $\mathcal{E}_{ML}$ exactly with computational complexity $\mathcal{O}(kn^2 + n^2 \log n)$.*

We use $\mathcal{Q}_{F,\mathcal{T}}$ to denote the family of distributions with a given FVS F and a fixed

tree structure $\mathcal{T}$ with edge set $\mathcal{E}_T$ among the non-feedback nodes. The following Lemma 5.3.3 gives a closed-form solution that minimizes the K-L divergence in $\mathcal{Q}_{F,\mathcal{T}}$.

Lemma 5.3.3 :

$$\min_{q\in\mathcal{Q}_{F,\mathcal{T}}} D_{KL}(\hat{p}||q) = -H_{\hat{p}}(\mathbf{x}) + H_{\hat{p}}(\mathbf{x}_F) + \sum_{i\in\mathcal{V}\setminus F} H_{\hat{p}}(\mathbf{x}_i|\mathbf{x}_F) - \sum_{(i,j)\in\mathcal{E}_T} I_{\hat{p}}(\mathbf{x}_i;\mathbf{x}_j|\mathbf{x}_F), \quad (5.13)$$

where the minimum K-L divergence is obtained if and only if: 1) $q(\mathbf{x}_F) = \hat{p}(\mathbf{x}_F)$*; 2)* $q(x_i, x_j|\mathbf{x}_F,) = \hat{p}(x_i, x_j|\mathbf{x}_F)$ *for any* $(i,j) \in \mathcal{E}_T$.

The proof of Lemma 5.3.3 is included in Appendix 5.7.

Proof of Proposition 5.3.2

Proof. According to Lemma 5.3.3, the optimal solution of Problem (5.11) is given by the left hand side (LHS) of (5.13) if we fix the FVS F and the tree structure $\mathcal{T}$. When we further optimize over all possible spanning trees among the non-feedback nodes, the optimal set of edges among the non-feedback nodes can be obtained by finding the maximum spanning tree of the subgraph induced by T with $I_{\hat{p}}(\mathbf{x}_i;\mathbf{x}_j|\mathbf{x}_F) \geq 0$ being the edge weight between i and j because all other terms in the LHS of (5.13) are invariant to the selection of trees.[4]

For Gaussian distributions, the conditional mutual information is only a function of the conditional covariance matrix $\hat{\Sigma}_{T|F} = \hat{\Sigma}_T - \hat{\Sigma}_M\hat{\Sigma}_F^{-1}\hat{\Sigma}_M^T$. Hence, finding the optimal edge set of the tree part is equivalent to running the Chow-Liu algorithm with the input being the covariance matrix $\hat{\Sigma}_{T|F}$. Let $\mathcal{E}_{\mathrm{CL}} = \mathrm{CL}_{\mathcal{E}}(\hat{\Sigma}_{T|F})$ and $\Sigma_{\mathrm{CL}} = \mathrm{CL}(\hat{\Sigma}_{T|F})$ and denote the optimal covariance matrix as

$$\Sigma_{\mathrm{ML}} = \begin{bmatrix} \Sigma_F^{\mathrm{ML}} & \left(\Sigma_M^{\mathrm{ML}}\right)^T \\ \Sigma_M^{\mathrm{ML}} & \Sigma_T^{\mathrm{ML}} \end{bmatrix}. \quad (5.14)$$

According to Lemma 5.3.3, we must have $\Sigma_F^{\mathrm{ML}} = \hat{\Sigma}_F$. Since $\mathcal{T}$ is a spanning tree among nodes in T, Lemma 5.3.3 implies that for all $i \in T$, $q(\mathbf{x}_F, x_i) = \hat{p}(\mathbf{x}_F, x_i)$. Hence we also have that $\Sigma_M^{\mathrm{ML}} = \hat{\Sigma}_M$. Furthermore, the corresponding conditional covariance

[4]In fact, we have given an algorithm to learn general models (not only for GGMs, but also for other models, e.g., discrete ones) defined on graphs with a given FVS F. However, we do not explore the general setting in this chapter.

matrix $\Sigma_{T|F}^{\mathrm{ML}}$ of Σ_{ML} must equal Σ_{CL}, i.e.,

$$\Sigma_{T|F}^{\mathrm{ML}} = \Sigma_{T}^{\mathrm{ML}} - \Sigma_{M}^{\mathrm{ML}} \left(\Sigma_{F}^{\mathrm{ML}}\right)^{-1} \left(\Sigma_{M}^{\mathrm{ML}}\right)^{T} = \Sigma_{\mathrm{CL}}. \tag{5.15}$$

Therefore, we can obtain

$$\Sigma_{T}^{\mathrm{ML}} = \mathrm{CL}(\hat{\Sigma}_{T|F}) + \hat{\Sigma}_{M} \hat{\Sigma}_{F}^{-1} \hat{\Sigma}_{M}^{T}. \tag{5.16}$$

We also have that $\mathcal{E}_{\mathrm{ML}} = \mathcal{E}_{\mathrm{CL}}$ since $\mathcal{E}_{\mathrm{ML}}$ is defined to be the set of edges among the feedback nodes.

Now we analyze the computational complexity of Algorithm 5.3.1. The matrix $\hat{\Sigma}_{T|F}$ is computed with complexity $\mathcal{O}(kn^2)$. Computing the maximum weight spanning tree has complexity $\mathcal{O}(n^2 \log n)$ using Kruskal's algorithm (the amortized complexity can be further reduced, but it is not the focus of this chapter). Other operations have complexity $\mathcal{O}(n^2)$. Hence, the total complexity of Algorithm 5.3.1 is $\mathcal{O}(kn^2 + n^2 \log n)$. We have thus completed the proof of Proposition 5.3.2.

□

In many situations, computing the information matrix $J_{\mathrm{ML}} = \Sigma_{\mathrm{ML}}^{-1}$ is also useful. A straightforward method is to use direct matrix inversion, but the computational complexity is $\mathcal{O}(n^3)$. In the following part of this subsection, we describe an algorithm that can make use of the intermediate results in Algorithm 5.3.1 to compute J_{ML} with complexity $\mathcal{O}(k^2 n)$, which is a significant reduction when k is small. Before we introduce the algorithm we first state Lemma 5.3.4, which provides an algorithm to compute the inverse of a given covariance matrix in linear time when the underlying model has a known tree structure. The proof of Lemma 5.3.4 is deferred to Appendix 5.7.

Lemma 5.3.4 : *If $\Sigma \succ 0$ is given and we know that its inverse $J = \Sigma^{-1}$ is sparse with respect to a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, then the non-zero entries of J can be computed using* (5.17) *in time $\mathcal{O}(n)$.*

$$J_{ij} = \begin{cases} (1 - deg(i))\, \Sigma_{ii}^{-1} + \sum_{j \in \mathcal{N}(i)} \left(\Sigma_{ii} - \Sigma_{ij} \Sigma_{jj}^{-1} \Sigma_{ji}\right)^{-1} & i = j \in \mathcal{V} \\ \frac{\Sigma_{ij}}{\Sigma_{ij}^2 - \Sigma_{ii}\Sigma_{jj}} & (i,\ j) \in \mathcal{E} \\ 0 & otherwise. \end{cases} \tag{5.17}$$

Our efficient algorithm to compute $J_{\mathrm{ML}} = (\Sigma_{\mathrm{ML}})^{-1}$ is summarized in Algorithm 5.3.2.

Algorithm 5.3.2 Compute $J_{\mathrm{ML}} = (\Sigma_{\mathrm{ML}})^{-1}$ After Running Algorithm 5.3.1

Input: Σ_{ML} and intermediate results in Algorithm 5.3.1

Output: $J_{\mathrm{ML}} = (\Sigma_{\mathrm{ML}})^{-1}$

1. Compute J_T^{ML} using (5.17) with Σ_{CL} as the input covariance matrix and $\mathcal{E}_T$ as the tree structure.
2. Compute $J_M^{\mathrm{ML}} = -J_T^{\mathrm{ML}} \Sigma_M^{\mathrm{ML}} \Sigma_F^{-1}$ using sparse matrix multiplication.
3. Compute $\left(\Sigma_F^{\mathrm{ML}}\right)^{-1} \left(I + \left(\left(\Sigma_M^{\mathrm{ML}}\right)^T J_T^{\mathrm{ML}}\right) \left(\Sigma_M^{\mathrm{ML}} \left(\Sigma_F^{\mathrm{ML}}\right)\right)\right)$ following the order specified by the parentheses using sparse matrix multiplication.

The following Proposition 5.3.5 states the correctness and the computational complexity of Algorithm 5.3.2. Its proof is provided after we introduce Lemma 5.3.6.

Proposition 5.3.5 : *The non-zero entries of $J_{ML} \triangleq \Sigma_{ML}^{-1}$ can be computed with extra complexity $\mathcal{O}(k^2 n)$ using Algorithm 5.3.2 after we run Algorithm 5.3.1.*

Lemma 5.3.6 : *(The Matrix Inversion Lemmas)*

If $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$ *is invertible, we have*

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} (A - BD^{-1}C)^{-1} & -(A - BD^{-1}C)^{-1}BD^{-1} \\ -D^{-1}C(A - BD^{-1}C)^{-1} & D^{-1} + D^{-1}C(A - BD^{-1}C)^{-1}BD^{-1} \end{bmatrix} \tag{5.18}$$

or

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix}, \tag{5.19}$$

which implies that

$$\left(A - BD^{-1}C\right)^{-1} = A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1}. \tag{5.20}$$

The proof of Lemma 5.3.6 can be find in standard texts such as [67].

Proof of Proposition 5.3.5

Proof. According to the proof of Algorithm 5.3.1, we have that $J_T^{\mathrm{ML}} = \left(\mathrm{CL}(\hat{\Sigma}_{T|F})\right)^{-1}$ has tree structure $\mathcal{T}$. Hence, according to Lemma 5.3.4, the non-zero entries of J_T^{ML}can be computed with complexity $\mathcal{O}(n-k)$ using (5.17).

From (5.19) in Lemma 5.3.6, we have

$$J_M^{\mathrm{ML}} = -J_T^{\mathrm{ML}}\Sigma_M^{\mathrm{ML}}\left(\Sigma_F^{\mathrm{ML}}\right)^{-1}, \tag{5.21}$$

which can be computed with complexity $\mathcal{O}(k^2 n)$ by matrix multiplication in the "regular order".[5] Note that $J_T^{\mathrm{ML}}\Sigma_M^{\mathrm{ML}}$ is computed in $\mathcal{O}(kn)$ since J_T^{ML} only has $\mathcal{O}(n)$ non-zero entries.

Again, from (5.19), we have

$$J_F^{\mathrm{ML}} = \left(\Sigma_F^{\mathrm{ML}}\right)^{-1}\left(I + \left(\left(\Sigma_M^{\mathrm{ML}}\right)^T J_T^{\mathrm{ML}}\right)\left(\Sigma_M^{\mathrm{ML}}\left(\Sigma_F^{\mathrm{ML}}\right)\right)\right), \tag{5.22}$$

which has complexity $\mathcal{O}(k^2 n)$ following the order specified by the parentheses. Note that $\left(P_M^{\mathrm{ML}}\right)^T J_T^{\mathrm{ML}}$is computed in $\mathcal{O}(kn)$ because J_T^{ML} only has $\mathcal{O}(n)$ non-zero entries. Therefore, we only need an extra complexity of $\mathcal{O}(k^2 n)$ to compute all the non-zero entries of J_{ML}. This completes the proof of Proposition 5.3.5.

□

■ 5.3.2 Case 2: The FVS Is to Be Learned

Structure learning becomes more computationally involved when the FVS is unknown. In this subsection, we present both exact and approximate algorithms for learning models when an FVS of size no larger than k is to be learned.

For a fixed empirical distribution $\hat{p}(\mathbf{x}_F, \mathbf{x}_T)$, we define $d(F)$, a set function of the FVS F, as the minimum value of (5.11), i.e.,

$$d(F) = \min_{q(\mathbf{x}_F,\mathbf{x}_T)\in\mathcal{Q}_F} D_{\mathrm{KL}}(\hat{p}(\mathbf{x}_F,\mathbf{x}_T)||q(\mathbf{x}_F,\mathbf{x}_T)). \tag{5.23}$$

[5] In this example, "regular order" means first computing the inverse of Σ_F^{ML} and then following the left-to-right order for matrix multiplication.

Hence, obtaining the ML estimate in this case is equivalent to solving

$$\min_{|F|\leq k} d(F). \tag{5.24}$$

For any given F, the value of $d(F)$ can be computed using Algorithm 5.3.1 and then using (5.13) in Lemma 5.3.3. The ML estimate in this case, i.e., the solution of (5.24) can be computed exactly by enumerating all possible $\binom{n}{k}$ FVSs of size k to find the F that minimizes $d(F)$. Hence, the exact solution can be obtained with complexity $\mathcal{O}(n^{k+2}k)$, which is polynomial in n for fixed k. However, as our empirical results suggest, choosing $k = \mathcal{O}(\log(n))$ works well, leading to quasi-polynomial complexity even for this exact algorithm. That observation notwithstanding, the following greedy algorithm (Algorithm 5.3.3), which, at each iteration, selects the single best node to add to the current set of feedback nodes, has polynomial complexity for arbitrarily large FVSs. As we will demonstrate in Section 5.5, this greedy algorithm works extremely well in practice.

Algorithm 5.3.3 Selecting an FVS by a Greedy Approach

Initialization: $F_0 = \emptyset$

For $t = 1$ **to** k,

$$k_t^* = \arg\min_{k \in V \setminus F_{t-1}} d(F_{t-1} \cup \{k\}),$$
$$F_t = F_{t-1} \cup \{k_t^*\}.$$

5.4 Learning GGMs with Latent FVSs

In this section, we study the structure learning problem when the feedback nodes are latent variables. Since the information matrix of the entire model is $J = \begin{bmatrix} J_F & J_M^T \\ J_M & J_T \end{bmatrix}$, the marginal distribution of observed variables (the non-feedback nodes) has information matrix $\hat{J}_T = \hat{\Sigma}_T^{-1} = J_T - J_M J_F^{-1} J_M^T$ by taking the Schur complement. If the exact $\hat{J}_T$ is known, the learning problem is equivalent to decomposing a given inverse covariance

matrix $\hat{J}_T$ into the sum of a tree-structured matrix J_T and a rank-k matrix $-J_M J_F^{-1} J_M^T$.[6] In general, the observations are noisy and we use the ML criterion:

$$q_{\mathrm{ML}}(\mathbf{x}_F, \mathbf{x}_T) = \arg \min_{q(\mathbf{x}_F, \mathbf{x}_T) \in Q_F} D_{\mathrm{KL}}(\hat{p}(\mathbf{x}_T) || q(\mathbf{x}_T)), \tag{5.25}$$

where the optimization is over all nodes (latent and observed) while the K-L divergence in the objective function is defined on the marginal distribution of the observed nodes only. In the following part of this section, we propose the *latent Chow-Liu algorithm* to solve (5.25) in Section 5.4.1 and also provide its accelerated version in Section 5.4.2.

5.4.1 The Latent Chow-Liu Algorithm

In this subsection, we propose the latent Chow-Liu algorithm, an alternating projection algorithm that has a similar structure to the EM algorithm and can be viewed as an instance of the majorization-minimization algorithm [78]. The general form of the algorithm is summarized in Algorithm 5.4.1.

Algorithm 5.4.1 Alternating Projection

1. Propose an initial distribution $q^{(0)}(\mathbf{x}_F, \mathbf{x}_T) \in \mathcal{Q}_F$
2. Alternate between projections **P1** and **P2**
   (a) **P1:** Project to the empirical distribution:
   $$\hat{p}^{(t)}(\mathbf{x}_F, \mathbf{x}_T) = \hat{p}(\mathbf{x}_T) q^{(t)}(\mathbf{x}_F | \mathbf{x}_T).$$
   (b) **P2:** Project to the best fitting structure on all variables:
   $$q^{(t+1)}(\mathbf{x}_F, \mathbf{x}_T) = \arg \min_{q(\mathbf{x}_F, \mathbf{x}_T) \in \mathcal{Q}_F} D(\hat{p}^{(t)}(\mathbf{x}_F, \mathbf{x}_T) || q(\mathbf{x}_F, \mathbf{x}_T)).$$

In the first projection **P1**, we obtain a distribution (on both observed and latent variables) whose marginal (on the observed variables) matches exactly the empirical

[6] It is easy to see that different models having the same $J_M J_F^{-1} J_M$ cannot be distinguished using the samples, and thus without loss of generality we can assume J_F is normalized to be the identify matrix in the final solution.

distribution while maintaining the conditional distribution (of the latent variables given the observed ones). In the second projection **P2**, we compute a distribution (on all variables) in the family considered that is the closest to the distribution obtained in the first projection. We find that among various EM type algorithms, this formulation is the most revealing for our problems because it clearly relates the second projection to the scenario where an FVS F is both observed and known (Section 5.3.1). Therefore, we are able to compute the second projection *exactly* even though the graph structure is *unknown* (which allows *any* tree structure among the observed nodes). Note that when the feedback nodes are latent, we do not need to select the FVS since it is simply the set of latent nodes. This is the source of the simplification when we use latent nodes for the FVS: We have no search of sets of observed variables to include in the FVS.

The following Lemma 5.4.1 states that the K-L divergence decreases monotonically using Algorithm 5.4.1 and also provides conditions for stationary points. The proof of Lemma 5.4.1 is included in Appendix 5.7.

Lemma 5.4.1 : *In Algorithm 5.4.1, if Step 2(a) and Step 2(b) can be computed exactly, then we have that*

$$D(\hat{p}(\mathbf{x}_T)||q^{(t+1)}(\mathbf{x}_T)) \leq D(\hat{p}(\mathbf{x}_T)||q^{(t)}(\mathbf{x}_T)), \tag{5.26}$$

where the equality is satisfied if and only if

$$\hat{p}^{(t)}(\mathbf{x}_F, \mathbf{x}_T) = \hat{p}^{(t+1)}(\mathbf{x}_F, \mathbf{x}_T). \tag{5.27}$$

In Algorithm 5.4.2, we summarize the latent Chow-Liu algorithm specialized for our family of GGMs, where both projections have exact closed-form solutions and exhibit complementary structure—one using the covariance and the other using the information parametrization. In projection **P1**, three blocks of the information matrix remain the same; In projection **P2,** three blocks of the covariance matrix remain the same.

As a rule of thumb, we often use the spanning tree obtained by the standard Chow-Liu algorithm as an initial tree among the observed nodes. But note that **P2** involves solving a combinatorial problem exactly, so the algorithm is able to jump among different graph structures which reduces the chance of getting stuck at a bad local minimum and gives us much more flexibility in initializing graph structures. In the experiments, we

Algorithm 5.4.2 The Latent Chow-Liu Algorithm

Input: the empirical covariance matrix $\hat{\Sigma}_T$

Output: information matrix $J = \begin{bmatrix} J_F & J_M^T \\ J_M & J_T \end{bmatrix}$, where J_T is tree-structured

1. Initialization: $J^{(0)} = \begin{bmatrix} J_F^{(0)} & \left(J_M^{(0)}\right)^T \\ J_M^{(0)} & J_T^{(0)} \end{bmatrix}$.

2. Repeat for $t = 1, 2, 3, \ldots$

   (a) **P1**: Project to the empirical distribution:

   $$\hat{J}^{(t)} = \begin{bmatrix} J_F^{(t)} & (J_M^{(t)})^T \\ J_M^{(t)} & \left(\hat{\Sigma}_T\right)^{-1} + J_M^{(t)}(J_F^{(t)})^{-1}(J_M^{(t)})^T \end{bmatrix}.$$

   Define $\hat{\Sigma}^{(t)} = \left(\hat{J}^{(t)}\right)^{-1}$.

   (b) **P2:** Project to the best fitting structure:

   $$\begin{aligned} \Sigma^{(t+1)} &= \begin{bmatrix} \hat{\Sigma}_F^{(t)} & \left(\hat{\Sigma}_M^{(t)}\right)^T \\ \hat{\Sigma}_M^{(t)} & \mathrm{CL}(\hat{\Sigma}_{T|F}^{(t)}) + \hat{\Sigma}_M^{(t)}\left(\hat{\Sigma}_F^{(t)}\right)^{-1}\left(\hat{\Sigma}_M^{(t)}\right)^T \end{bmatrix} \\ &= \mathrm{CCL}(\hat{\Sigma}^{(t)}), \end{aligned}$$

   where $\hat{\Sigma}_{T|F}^{(t)} = \hat{\Sigma}_T^{(t)} - \hat{\Sigma}_M^{(t)}\left(\hat{\Sigma}_F^{(t)}\right)^{-1}\left(\hat{\Sigma}_M^{(t)}\right)^T$.

   Define $J^{(t+1)} = \left(\Sigma^{(t+1)}\right)^{-1}$.

will demonstrate that Algorithm 5.4.2 is not sensitive to the initial graph structure.

The two projections in Algorithm 5.4.2 can also be interpreted as alternating *low-rank* corrections: indeed,

$$\text{In P1}: \quad \hat{J}^{(t)} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \left(\hat{\Sigma}_T\right)^{-1} \end{bmatrix} + \begin{bmatrix} J_F^{(t)} \\ J_M^{(t)} \end{bmatrix} \left(J_F^{(t)}\right)^{-1} \begin{bmatrix} J_F^{(t)} & \left(J_M^{(t)}\right)^T \end{bmatrix}, \tag{5.28}$$

$$\text{and in P2 : } \Sigma^{(t+1)} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathrm{CL}(\hat{\Sigma}_{T|F}) \end{bmatrix} + \begin{bmatrix} \hat{\Sigma}_F^{(t)} \\ \hat{\Sigma}_M^{(t)} \end{bmatrix} \left(\hat{\Sigma}_F^{(t)}\right)^{-1} \begin{bmatrix} \hat{\Sigma}_F^{(t)} & \left(\hat{\Sigma}_M^{(t)}\right)^T \end{bmatrix}, \tag{5.29}$$

where the second terms of both expressions are of low rank when the size of the latent FVS is small. This formulation is the most intuitive and simple, but a naive implementation of Algorithm 5.4.2 has complexity $\mathcal{O}(n^3)$ per iteration, where the bottleneck is inverting full matrices $\hat{J}^{(t)}$ and $\Sigma^{(t+1)}$. In Section 5.4.2 to follow, we introduce an accelerated version of Algorithm 5.4.2 that can reduce the overall computational complexity to $\mathcal{O}(kn^2 + n^2 \log n)$.

The following Proposition 5.4.2 states the correctness of Algorithm 5.4.2.

Proposition 5.4.2 : *Using Algorithm 5.4.2, the objective function of* (5.25) *decreases with the number of iterations, i.e.,*

$$D_{KL}(\mathcal{N}(0, \hat{\Sigma}_T)||\mathcal{N}(0, \Sigma_T^{(t+1)})) \leq \mathcal{N}(0, \hat{\Sigma}_T)||\mathcal{N}(0, \Sigma_T^{(t)})), \tag{5.30}$$

where the equality is obtained if and only if

$$\hat{\Sigma}^{(t)} = \hat{\Sigma}^{(t+1)}. \tag{5.31}$$

Proof of Proposition 5.4.2

Proof. Let $\hat{p}(\mathbf{x}_T) = \mathcal{N}(\mathbf{0}, \hat{\Sigma}_T)$, $p^{(t)}(\mathbf{x}_F, \mathbf{x}_T) = \mathcal{N}(\mathbf{0}, \Sigma^{(t)})$. Then,

$$\hat{p}(\mathbf{x}_T) = \frac{1}{\sqrt{\det\left(2\pi\hat{\Sigma}_T\right)}} \exp\{-\frac{1}{2}\mathbf{x}_T^T \hat{\Sigma}_T^{-1} \mathbf{x}_T\}, \tag{5.32}$$

$$p^{(t)}(\mathbf{x}_F|\mathbf{x}_T) = \frac{1}{\sqrt{\det\left(2\pi\left(J_F^{(t)}\right)^{-1}\right)}} \exp\{-\frac{1}{2}\left(\mathbf{x}_F - \left(J_F^{(t)}\right)^{-1} J_M^{(t)} \mathbf{x}_T\right)^T J_F^{(t)} \left(\mathbf{x}_F - \left(J_F^{(t)}\right)^{-1} J_M^{(t)} \mathbf{x}_T\right)^T\}. \tag{5.33}$$

Applying Algorithm 5.4.1, we have

$$\hat{p}^{(t)}(\mathbf{x}_F, \mathbf{x}_T) = \hat{p}(\mathbf{x}_T) q^{(t)}(\mathbf{x}_F|\mathbf{x}_T)$$
$$\propto \exp\{-\frac{1}{2}\begin{bmatrix} \mathbf{x}_F \\ \mathbf{x}_T \end{bmatrix}^T \begin{bmatrix} J_F^{(t)} & \left(J_M^{(t)}\right)^T \\ J_M^{(t)} & \hat{\Sigma}_T^{-1} + J_M^{(t)}(J_F^{(t)})^{-1}(J_M^{(t)})^T \end{bmatrix} \begin{bmatrix} \mathbf{x}_F \\ \mathbf{x}_T \end{bmatrix}\}, \quad (5.34)$$

which gives the same expression as in **P1** of Algorithm 5.4.2.

The next projection

$$q^{(t+1)}(\mathbf{x}_F, \mathbf{x}_T) = \arg \min_{q(\mathbf{x}_F, \mathbf{x}_T) \in \mathcal{Q}_F} D(\hat{p}^{(t)}(\mathbf{x}_F, \mathbf{x}_T) || q(\mathbf{x}_F, \mathbf{x}_T)) \quad (5.35)$$

has same form as the ML learning problem of (5.11) in Section 5.3.1, and therefore can be computed exactly using Algorithm 5.3.1.

According to Lemma 5.4.1, we then see that

$$D_{\mathrm{KL}}(\mathcal{N}(0, \hat{\Sigma}_T) || \mathcal{N}(0, \Sigma_T^{(t+1)})) \leq \mathcal{N}(0, \hat{\Sigma}_T) || \mathcal{N}(0, \Sigma_T^{(t)})) \quad (5.36)$$

and $\hat{\Sigma}^{(t)} = \hat{\Sigma}^{(t+1)}$ is the necessary and sufficient condition for stationary points.

□

■ 5.4.2 The Accelerated Latent Chow-Liu Algorithm

In this subsection, we describe *the accelerated latent Chow-Liu algorithm*, which gives the same results as Algorithm 5.4.2 but has significant speedup. The key idea of this accelerated version is to carefully incorporate the inference algorithms into the projection steps, so that we are able to further exploit the power of the models and reduce the per-iteration complexity to $\mathcal{O}(kn^2 + n^2 \log n)$, which is the same as the complexity of the conditioned Chow-Liu algorithm alone. This accelerated version is summarized in Algorithm 5.4.3.

The following Proposition 5.4.3 states the correctness and the computational complexity of Algorithm 5.4.3.

Proposition 5.4.3 : *Algorithm 5.4.3 has computational complexity* $\mathcal{O}(kn^2 + n^2 \log n)$ *and gives the same results as Algorithm 5.4.2.*

Algorithm 5.4.3 The Accelerated Latent Chow-Liu algorithm

Input: the empirical covariance matrix $\hat{\Sigma}_T$

Output: information matrix $J = \begin{bmatrix} J_F & J_M^T \\ J_M & J_T \end{bmatrix}$

1. Initialization: $J^{(0)} = \begin{bmatrix} J_F^{(0)} & \left(J_M^{(0)}\right)^T \\ J_M^{(0)} & J_T^{(0)} \end{bmatrix}$.

2. Repeat

   (a) **AP1:** Compute

$$\begin{aligned} \hat{\Sigma}_F^{(t)} &= \left(J_F^{(t)}\right)^{-1} + \left(Y^{(t)}\right)^T \hat{\Sigma}_T Y^{(t)} \\ \hat{\Sigma}_T^{(t)} &= \hat{\Sigma}_T \\ \hat{\Sigma}_M^{(t)} &= -\hat{\Sigma}_T Y^{(t)}, \end{aligned}$$

   where $Y^{(t)} = \hat{J}_M^{(t)} \left(J_F^{(t)}\right)^{-1}$ and $\hat{\Sigma}^{(t)} = \begin{bmatrix} \hat{\Sigma}_F^{(t)} & \left(\hat{\Sigma}_M^{(t)}\right)^T \\ \hat{\Sigma}_M^{(t)} & \hat{\Sigma}_T \end{bmatrix}$.

   (b) **AP2:** Compute $\Sigma^{(t+1)}$ and $J^{(t+1)} = \left(\Sigma^{(t+1)}\right)^{-1}$ from $\hat{\Sigma}^{(t)}$ using Algorithm 5.3.1 and Algorithm 5.3.2 to obtain:

$$\begin{aligned} J^{(t+1)} &= \begin{bmatrix} J_F^{(t+1)} & \left(J_M^{(t+1)}\right)^T \\ J_M^{(t+1)} & J_T^{(t+1)} \end{bmatrix}, \\ \Sigma^{(t+1)} &= \begin{bmatrix} \Sigma_F^{(t+1)} & \left(\Sigma_M^{(t+1)}\right)^T \\ \Sigma_M^{(t+1)} & \Sigma_T^{(t+1)} \end{bmatrix}. \end{aligned}$$

Proof. In **P1** of Algorithm 5.4.2, we have

$$\hat{J}^{(t)} = \begin{bmatrix} J_F^{(t)} & (J_M^{(t)})^T \\ J_M^{(t)} & \left(\hat{\Sigma}_T\right)^{-1} + J_M^{(t)}(J_F^{(t)})^{-1}(J_M^{(t)})^T \end{bmatrix}. \tag{5.37}$$

Without explicitly computing $\hat{J}^{(t)}$, we can directly compute $\hat{\Sigma}^{(t)} = \left(\hat{J}^{(t)}\right)^{-1}$ as follows.

Let $A = J_F^{(t)}$, $B = (J_M^{(t)})^T$, $C = \hat{J}_M^{(t)}$, and $D = \left(\hat{\Sigma}_T\right)^{-1} + J_M^{(t)}(J_F^{(t)})^{-1}(J_M^{(t)})^T)$. From (5.19) in Lemma 5.3.6, we have

$$\hat{\Sigma}_F^{(t)} = \left(J_F^{(t)}\right)^{-1} + \left(J_F^{(t)}\right)^{-1}\left(J_M^{(t)}\right)^T (D - CA^{-1}B)^{-1}\hat{J}_M^{(t)}\left(J_F^{(t)}\right)^{-1}, \tag{5.38}$$

$$\hat{\Sigma}_T^{(t)} = (D - CA^{-1}B)^{-1} = \hat{\Sigma}_T, \tag{5.39}$$

and

$$\hat{\Sigma}_F^{(t)} = \left(J_F^{(t)}\right)^{-1} + \left(J_F^{(t)}\right)^{-1}\left(J_M^{(t)}\right)^T \hat{\Sigma}_T \hat{J}_M^{(t)}\left(J_F^{(t)}\right)^{-1}. \tag{5.40}$$

Again, from (5.19) in Lemma 5.3.6, we have that

$$\hat{\Sigma}_M^{(t)} = -\hat{\Sigma}_T J_M^{(t)}\left(J_F^{(t)}\right)^{-1}. \tag{5.41}$$

The equations (5.39)–(5.41) are the same as the equations in **AP1** in Algorithm 5.4.3, and thus give the same results as **P1** in Algorithm 5.4.2. It can be checked that the matrix multiplications in equations (5.39), (5.40), and (5.41) have a combined complexity of $\mathcal{O}(kn^2)$.

AP2 in Algorithm 5.4.3 can be computed with complexity $\mathcal{O}(n^2k + n^2 \log n)$ from Proposition 5.3.2. Since **AP2** in Algorithm 5.4.3 is the same as **P2** in Algorithm 5.4.2, they give exactly the same results.

Therefore, Algorithm 5.4.3 gives the same results as Algorithm 5.4.2 and the complexity of Algorithm 5.4.3 is $\mathcal{O}(n^2k + n^2 \log n)$ per iteration. We have thus completed the proof for Proposition 5.4.3.

□

5.5 Experiments

In this section, we present experimental results for learning GGMs with small FVSs, observed or latent, using both synthetic data and real data of flight delays.

5.5.1 Fractional Brownian Motion: Latent FVS

We consider a fractional Brownian motion (FBM) with Hurst parameter $H = 0.2$ defined on the time interval $(0, 1]$. The covariance function is $\Sigma(t_1, t_2) = \frac{1}{2}(|t_1|^{2H} + |t_2|^{2H} -$

$|t_1 - t_2|^{2H}$). Figure 5.1 shows the covariance matrices of approximate models using spanning trees (learned by the Chow-Liu algorithm), latent trees (learned by the CLRG and NJ algorithms in [79]) and our latent FVS model (learned by Algorithm 5.4.2) using 64 time samples (nodes). We can see that in the spanning tree the correlation decays quickly (in fact exponentially) with distance, which models the FBM poorly. The latent trees that are learned exhibit blocky artifacts and have little or no improvement over the spanning tree measured in the K-L divergence. In Figure 5.2, we plot the K-L divergence (between the true model and the learned models using Algorithm 5.4.2) versus the size of the latent FVSs for models with 32, 64, 128, and 256 time samples respectively. For these models, we need about 1, 3, 5, and 7 feedback nodes respectively to reduce the K-L divergence to 25% of that achieved by the best spanning tree model. Hence, we speculate that empirically $k = \mathcal{O}(\log n)$ is a proper choice of the size of the latent FVS. We also study the sensitivity of Algorithm 5.4.2 to the initial graph structure. In our experiments, for different initial structures, Algorithm 5.4.2 converges to the same graph structures (that give the K-L divergence as shown in Figure 5.2) within three iterations.

5.5.2 Performance of the Greedy Algorithm: Observed FVS

In this experiment, we examine the performance of the greedy algorithm (Algorithm 5.3.3) when the FVS nodes are not latent, i.e., they are observed. For each run, we construct a GGM that has 20 nodes and an FVS of size three as the true model. We first generate a random spanning tree among the non-feedback nodes. Then the corresponding information matrix J is also randomly generated: non-zero entries of J are drawn *i.i.d.* from the uniform distribution $U[-1, 1]$ with a multiple of the identity matrix added to ensure $J \succ 0$. From each generated GGM, we draw 1000 samples and use Algorithm 5.3.3 to learn the model. For the 100 runs that we have performed, we recover the true graph structures successfully. Figure 5.3 shows the graphs (and the K-L divergence) obtained using the greedy algorithm for a typical run. We can see that we have the most divergence reduction (from 12.7651 to 1.3832) when the first feedback node is selected (See Figure 5.3b and Figure 5.3c). When the size of the FVS increases to three (Figure 5.3e), the graph structure is recovered correctly.

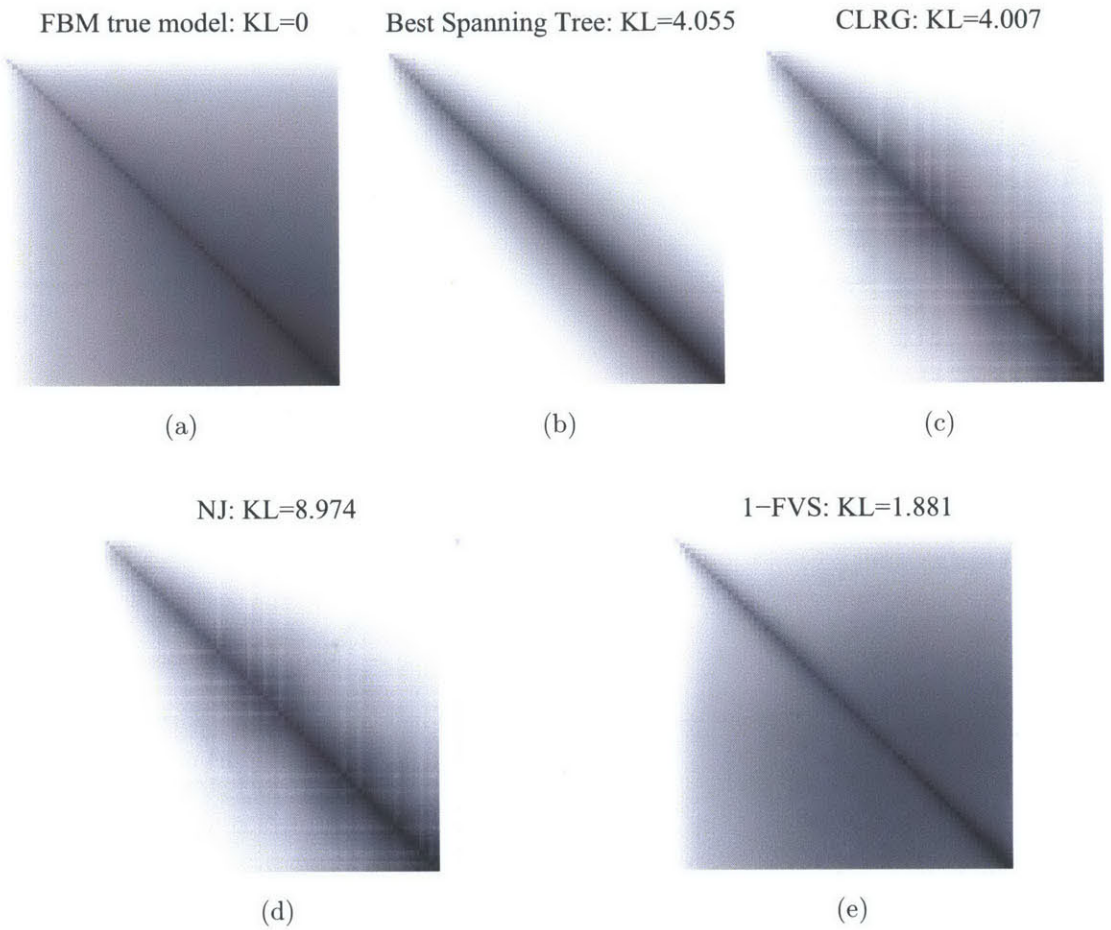
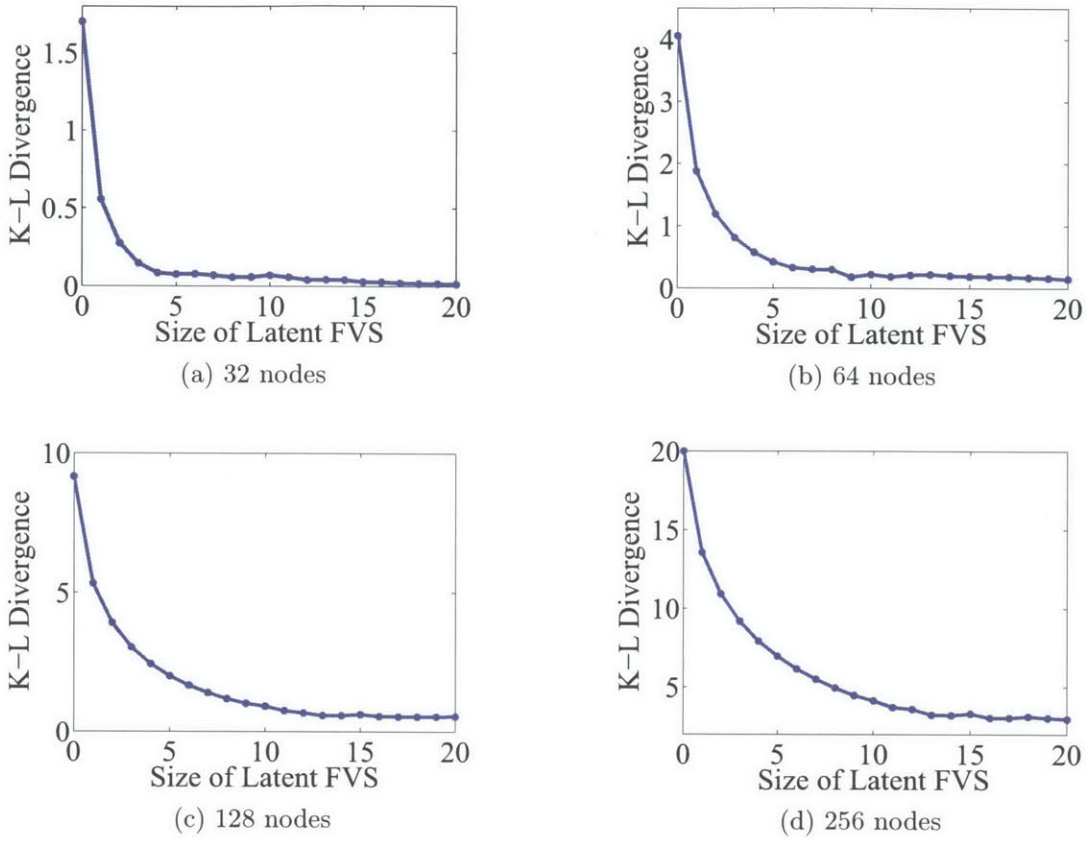


Figure 5.1: Covariance matrix obtained using various algorithms and structures. (a) The true model (FBM with 64 time samples); (b) The best spanning tree; (c) The latent tree learned using the CLRG algorithm in [79]; (d) The latent tree learned using the NJ algorithm in [79]; (e) The model with a size-one latent FVS learned using Algorithm 5.4.2. The gray scale is normalized for visual clarity.

5.5.3 Flight Delay Model: Observed FVS

In this experiment, we model the relationships among airports for flight delays using models with observed FVSs (non-latent FVSs). The raw dataset comes from the Research and Innovative Technology Administration of the Bureau of Transportation



(a) 32 nodes

(b) 64 nodes

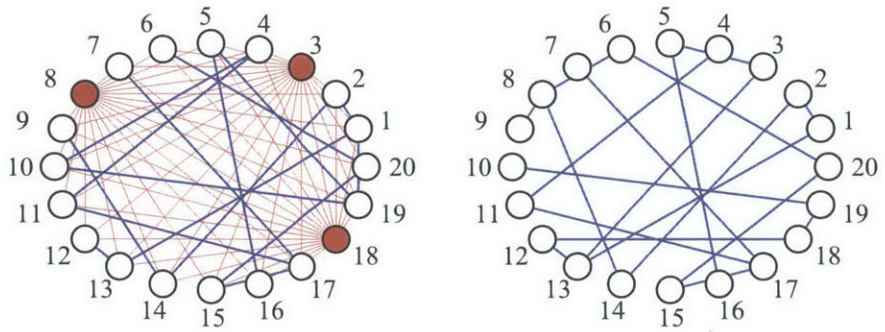
(c) 128 nodes

(d) 256 nodes

Figure 5.2: The relationship between the K-L divergence and the latent FVS size. All models are learned using Algorithm 5.4.2 with 40 iterations.

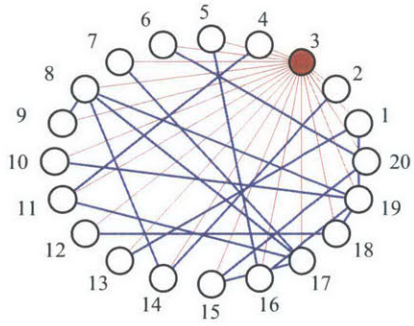
Statistics.[7] It contains flight information in the U.S. from 1987 to 2008 including information such as scheduled departure time, scheduled arrival time, departure delay, arrival delay, cancellation, and reasons for cancellation for all domestic flights in the U.S. We want to model how the flight delays at different airports are related to each other using GGMs. First, we compute the average departure delay for each day and each airport (of the top 200 busiest airports) using data from the year 2008. Note that the average departure delay does not directly indicate whether an airport is one of the major airports that has heavy traffic. It is interesting to see whether major airports (especially those notorious for delays) correspond to feedback nodes in the learned models.

[7]The data we used in this experiment can be obtained at http://www.transtats.bts.gov/OT_Delay/OT_DelayCause1.asp

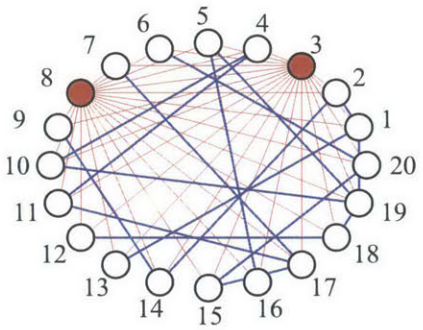


(a) True Model

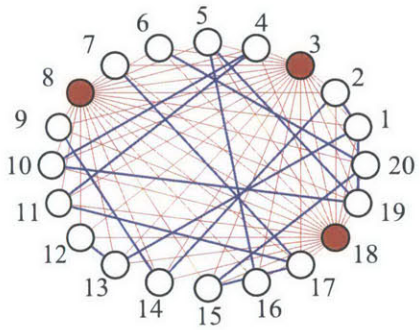
(b) KL=12.7651



(c) KL=1.3832



(d) KL=0.6074



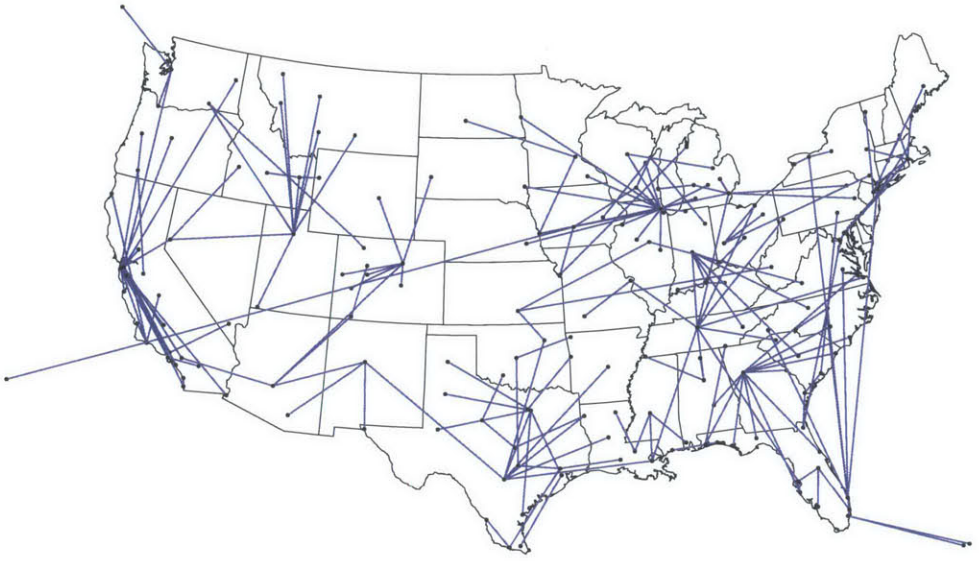
(e) KL=0.0048

Figure 5.3: Learning a GGM using Algorithm 5.3.3. The thicker blue lines represent the edges among the non-feedback nodes and the thinner red lines represent other edges. (a) True model; (b) Tree-structured model (0-FVS) learned from samples; (c) 1-FVS model; (d) 2-FVS model; (e) 3-FVS model.

Figure 5.4a shows the best tree-structured graph obtained by the Chow-Liu algorithms (with input being the covariance matrix of the average delay). Figure 5.4b, Figure 5.5a, and Figure 5.5b show the GGMs learned using Algorithm 5.3.3 with FVSs of sizes 1, 3, and 10 respectively. It is interesting that the first node selected is Nashville (BNA), which is not one of the top "hubs" of the air system. The reason is that much of the statistical relationships related to those hubs are approximated well, when we consider a 1-FVS approximation, by a spanning tree (excluding BNA) and it is the breaking of the cycles involving BNA that provide the most reduction in K-L divergence over a spanning tree. Starting with the next node selected in our greedy algorithm, we begin to see hubs being chosen. In particular, the first ten airports selected in order are: BNA, Chicago, Atlanta, Oakland, Newark, Dallas, San Francisco, Seattle, Washington DC, Salt Lake City. Several major airports on the coasts (e.g., Los Angeles and JFK) are not selected, as their influence on delays at other domestic airports is well-captured with a tree structure.

5.6 Future Directions

Our experimental results demonstrate the potential of these algorithms, and, as in the work [16], suggests that choosing FVSs of size $\mathcal{O}(\log n)$ works well, leading to algorithms which can be scaled to large problems. Providing theoretical guarantees for this scaling (e.g., by specifying classes of models for which such a size FVS provides asymptotically accurate models) is thus a compelling open problem. In addition, incorporating complexity into the FVS-order problem (e.g., as in AIC or BIC) is another direction worthy of consideration.



(a) Spanning Tree

(b) 1-FVS GGM

Figure 5.4: GGMs with FVSs of sizes 0 and 1 for modeling flight delays. The red dots denote the selected feedback nodes and the blue lines represent the edges among the non-feedback nodes (other edges involving the feedback nodes are omitted for clarity).

(a) 3-FVS GGM

(b) 10-FVS GGM

Figure 5.5: GGMs with FVSs of sizes 3 and 10 for modeling flight delays. The red dots denote the selected feedback nodes and the blue lines represent the edges among the non-feedback nodes (other edges involving the feedback nodes are omitted for clarity).

5.7 Appendix for Chapter 5

Proof of Lemma 5.2.2

Lemma 5.2.2 : *If the information matrix $J \succ 0$ has tree structure $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, then we have*

$$\det(J)^{-1} = \prod_{i \in \mathcal{V}} P_{ii} \prod_{(i,j) \in \mathcal{E}} \frac{P_{ii}P_{jj} - P_{ij}^2}{P_{ii}P_{jj}}, \tag{5.42}$$

where $P = J^{-1}$.

Proof. Without loss of generality, we assume the means are zero. For any tree-structured distribution $p(\mathbf{x})$ with underlying tree $\mathcal{T}$, we have the following factorization according to Proposition 2.1.1.

$$p(\mathbf{x}) = \prod_{i \in \mathcal{V}} p(x_i) \prod_{(i,j) \in \mathcal{E}_{\mathcal{T}}} \frac{p(x_i, x_j)}{p(x_i)p(x_j)}. \tag{5.43}$$

For a GGM of n nodes, the joint distribution, the singleton marginal distributions, and the pairwise marginal distributions can be expressed as follows.

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{n}{2}} (\det J)^{-\frac{1}{2}}} \exp\{-\frac{1}{2}\mathbf{x}^T J \mathbf{x}\} \tag{5.44}$$

$$p(x_i) = \frac{1}{(2\pi)^{\frac{1}{2}} P_{ii}^{\frac{1}{2}}} \exp\{-\frac{1}{2}\mathbf{x}^T P_{ii}^{-1} \mathbf{x}\} \tag{5.45}$$

$$p(x_i, x_j) = \frac{1}{2\pi \left(\det \begin{bmatrix} P_{ii} & P_{ij} \\ P_{ji} & P_{jj} \end{bmatrix}\right)^{\frac{1}{2}}} \exp\{-\frac{1}{2}\mathbf{x}^T \begin{bmatrix} P_{ii} & P_{ij} \\ P_{ji} & P_{jj} \end{bmatrix}^{-1} \mathbf{x}\}. \tag{5.46}$$

Matching the normalization factors using (5.43), we obtain

$$\det(J)^{-1} = \prod_{i \in \mathcal{V}} P_{ii} \prod_{(i,j) \in \mathcal{E}} \frac{\det \begin{bmatrix} P_{ii} & P_{ij} \\ P_{ji} & P_{jj} \end{bmatrix}}{P_{ii}P_{jj}} \tag{5.47}$$

$$= \prod_{i \in \mathcal{V}} P_{ii} \prod_{(i,j) \in \mathcal{E}} \frac{P_{ii}P_{jj} - P_{ij}^2}{P_{ii}P_{jj}}. \tag{5.48}$$

This completes the proof of Lemma 5.2.2.

□

Proof of Lemma 5.3.3

Lemma 5.3.3 :

$$\min_{q\in\mathcal{Q}_{F,\mathcal{T}}} D_{KL}(\hat{p}||q) = -H_{\hat{p}}(\mathbf{x}) + H_{\hat{p}}(\mathbf{x}_F) + \sum_{i\in\mathcal{V}\setminus F} H_{\hat{p}}(\mathbf{x}_i|\mathbf{x}_F) - \sum_{(i,j)\in\mathcal{E}_T} I_{\hat{p}}(\mathbf{x}_i;\mathbf{x}_j|\mathbf{x}_F), \quad (5.49)$$

where the minimum K-L divergence is obtained if and only if: 1) $q(\mathbf{x}_F) = \hat{p}(\mathbf{x}_F)$*; 2)* $q(x_i, x_j|\mathbf{x}_F,) = \hat{p}(x_i, x_j|\mathbf{x}_F)$ *for any* $(i,j) \in \mathcal{E}_T$.

Proof. With fixed F and $\mathcal{T}$,

$$\begin{aligned}
D_{\mathrm{KL}}(\hat{p}||q) &= \int \hat{p}(\mathbf{x}) \log \frac{\hat{p}(\mathbf{x})}{q(\mathbf{x})} \mathrm{d}\mathbf{x} && (5.50)\\
&= -H_{\hat{p}}(\mathbf{x}) - \int \hat{p}(\mathbf{x}) \log q(\mathbf{x}) \mathrm{d}\mathbf{x} && (5.51)\\
&= -H_{\hat{p}}(\mathbf{x}) - \int \hat{p}(\mathbf{x}) \log \left(q(\mathbf{x}_F) q(\mathbf{x}_T|\mathbf{x}_F)\right) \mathrm{d}\mathbf{x} && (5.52)\\
&\overset{(a)}{=} -H_{\hat{p}}(\mathbf{x}) - \int \hat{p}(\mathbf{x}) \log \left(q(\mathbf{x}_F) q(\mathbf{x}_r|\mathbf{x}_F) \prod_{i\in\mathcal{V}\setminus F\setminus r} q(\mathbf{x}_i|\mathbf{x}_F, \mathbf{x}_{\pi(i)}) \right) \mathrm{d}\mathbf{x} && (5.53)\\
&= -H_{\hat{p}}(\mathbf{x}) - \int \hat{p}(\mathbf{x}_F) \log q(\mathbf{x}_F) \mathrm{d}\mathbf{x}_F - \int \hat{p}(\mathbf{x}_F, \mathbf{x}_r) \log q(\mathbf{x}_r|\mathbf{x}_F) \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_r \\
&\quad - \sum_{i\in\mathcal{V}\setminus F\setminus r} \int \hat{p}(\mathbf{x}_F, \mathbf{x}_{\pi(i)}, \mathbf{x}_i) \log q(\mathbf{x}_i|\mathbf{x}_F, \mathbf{x}_{\pi(i)}) \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_{\pi(i)} \mathrm{d}\mathbf{x}_i && (5.54)\\
&\overset{(b)}{=} -H_{\hat{p}}(\mathbf{x}) + H_{\hat{p}}(\mathbf{x}_F) + D(\hat{p}_F||q_F) + H_{\hat{p}}(\mathbf{x}_r|\mathbf{x}_F) + D(\hat{p}_{r|F}||q_{r|F}|\hat{p}_F) \\
&\quad + \sum_{i\in V\setminus F\setminus r} H_{\hat{p}}(\mathbf{x}_i|\mathbf{x}_{F,\pi(i)}) + D(\hat{p}_{i|F,r}||q_{i|F,r}|\hat{p}_{F,r}) && (5.55)\\
&\overset{(c)}{\geq} -H_{\hat{p}}(\mathbf{x}) + H_{\hat{p}}(\mathbf{x}_F) + H_{\hat{p}}(\mathbf{x}_r|\mathbf{x}_F) + \sum_{i\in V\setminus F\setminus r} H_{\hat{p}}(\mathbf{x}_i|\mathbf{x}_{F,\pi(i)}), && (5.56)
\end{aligned}$$

where (a) is obtained by using Factorization 1 in Proposition 2.1.1 with an arbitrary root node r; (b) can be directly verified using the definition of the information quantities, and the equality in (c) is satisfied when $q_F = \hat{p}_F$, $q_{r|F} = \hat{p}_{r|F}$, and $q_{i|F,\pi(i)} = \hat{p}_{i|F,\pi(i)}, \forall i \in$

$T\backslash r$, or equivalently when

$$q(\mathbf{x}_F) = \hat{p}(\mathbf{x}_F) \tag{5.57}$$

$$q(x_i, x_j|\mathbf{x}_F) = \hat{p}(x_i, x_j|\mathbf{x}_F), \ \forall (i,j) \in \mathcal{E}_T. \tag{5.58}$$

Next, we derive another expression for (5.56). By substituting (5.58) into Factorization s of Proposition 2.1.1, we have

$$q^*(\mathbf{x}) \ = \ \hat{p}(\mathbf{x}_F) \prod_{i \in T} \hat{p}(x_i|\mathbf{x}_F) \prod_{(i,j)\in\mathcal{E}_T} \frac{\hat{p}(\mathbf{x}_i, \mathbf{x}_j|\mathbf{x}_F)}{\hat{p}(\mathbf{x}_i|\mathbf{x}_F)\hat{p}(\mathbf{x}_j|\mathbf{x}_F)}. \tag{5.59}$$

Hence,

$$\begin{aligned}
\min_{q\in\mathcal{Q}_{F,\mathcal{T}}} D(\hat{p}||q) &= D(\hat{p}||q^*) \\
&\quad - H_{\hat{p}}(\mathbf{x}) + H_{\hat{p}}(\mathbf{x}_F) + \sum_{i\in V\backslash F} H_{\hat{p}}(\mathbf{x}_i|\mathbf{x}_F) && (5.60)\\
&\quad + \sum_{(i,j)\in\mathcal{E}_\mathcal{T}} \int \hat{p}_{F,i,j}(\mathbf{x}_F, \mathbf{x}_i, \mathbf{x}_j) \log \frac{\hat{p}(\mathbf{x}_i, \mathbf{x}_j|\mathbf{x}_F)}{\hat{p}(\mathbf{x}_i|\mathbf{x}_F)\hat{p}(\mathbf{x}_j|\mathbf{x}_F)} \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_i \mathrm{d}\mathbf{x}_j && (5.61)\\
&= H_{\hat{p}}(\mathbf{x}) + H_{\hat{p}}(\mathbf{x}_F) + \sum_{i\in V\backslash F} H_{\hat{p}}(\mathbf{x}_i|\mathbf{x}_F) && (5.62)\\
&\quad - \sum_{(i,j)\in\mathcal{E}_\mathcal{T}} \int \hat{p}_{F,i,j}(\mathbf{x}_F, \mathbf{x}_i, \mathbf{x}_j) \log \frac{\hat{p}(\mathbf{x}_i|\mathbf{x}_F)\hat{p}(\mathbf{x}_j|\mathbf{x}_F)}{\hat{p}(\mathbf{x}_i, \mathbf{x}_j|\mathbf{x}_F)} \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_i \mathrm{d}\mathbf{x}_j && (5.63)\\
&= -H_{\hat{p}}(\mathbf{x}) + H(\hat{p}_F) + \sum_{i\in\mathcal{V}\backslash F} H(\hat{p}_{i|F}|\mathbf{x}_F) - \sum_{(i,j)\in\mathcal{E}_\mathcal{T}} I_{\hat{p}}(\mathbf{x}_i; \mathbf{x}_j|\mathbf{x}_F). && (5.64)
\end{aligned}$$

This completes the proof of Lemma 5.3.3.

□

Proof of Lemma 5.3.4

Lemma 5.3.4 : *If $\Sigma \succ 0$ is given and we know that its inverse $J = \Sigma^{-1}$ is sparse with respect to a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, then the non-zero entries of J can be computed using* (5.65) *in time $\mathcal{O}(n)$.*

$$J_{ij} = \begin{cases} (1 - deg(i))\,\Sigma_{ii}^{-1} + \sum_{j \in \mathcal{N}(i)} \left(\Sigma_{ii} - \Sigma_{ij}\Sigma_{jj}^{-1}\Sigma_{ji}\right)^{-1} & i = j \in \mathcal{V} \\ \frac{\Sigma_{ij}}{\Sigma_{ij}^2 - \Sigma_{ii}\Sigma_{jj}} & (i,\ j) \in \mathcal{E} \\ 0 & otherwise. \end{cases} \tag{5.65}$$

Proof. Since $\Sigma \succ 0$, we can construct a Gaussian distribution $p(\mathbf{x})$ with zero mean and covariance matrix Σ. The distribution is tree-structured because $J = \Sigma^{-1}$ has tree structure $\mathcal{T}$. Hence, we have the following factorization according to Proposition 2.1.1.

$$p(\mathbf{x}) = \prod_{i \in \mathcal{V}} p(x_i) \prod_{(i,j) \in \mathcal{E}} \frac{p(x_i, x_j)}{p(x_i)p(x_j)}, \tag{5.66}$$

where

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{n}{2}} (\det J)^{-\frac{1}{2}}} \exp\{-\frac{1}{2}\mathbf{x}^T J \mathbf{x}\} \tag{5.67}$$

$$p(x_i) = \frac{1}{(2\pi)^{\frac{1}{2}} P_{ii}^{\frac{1}{2}}} \exp\{-\frac{1}{2}\mathbf{x}^T \Sigma_{ii}^{-1} \mathbf{x}\} \tag{5.68}$$

$$p(x_i, x_j) = \frac{1}{2\pi \left(\det \begin{bmatrix} \Sigma_{ii} & \Sigma_{ij} \\ \Sigma_{ji} & \Sigma_{jj} \end{bmatrix}\right)^{\frac{1}{2}}} \exp\{-\frac{1}{2}\mathbf{x}^T \begin{bmatrix} \Sigma_{ii} & \Sigma_{ij} \\ \Sigma_{ji} & \Sigma_{jj} \end{bmatrix}^{-1} \mathbf{x}\}. \tag{5.69}$$

By matching the quadratic coefficients in the exponents, we have that

$$J_{ii} = \Sigma_{ii}^{-1} + \sum_{j \in \mathcal{N}(i)} \left(\left(\begin{bmatrix} \Sigma_{ii} & \Sigma_{ji} \\ \Sigma_{ij} & \Sigma_{jj} \end{bmatrix}^{-1} \right)_{11} - \Sigma_{ii}^{-1} \right) \tag{5.70}$$

$$= (1 - \deg(i))\,\Sigma_{ii}^{-1} + \sum_{j \in \mathcal{N}(i)} \left(\Sigma_{ii} - \Sigma_{ij}\Sigma_{jj}^{-1}\Sigma_{ji}\right)^{-1} \tag{5.71}$$

and for $(i, j) \in \mathcal{E}$,

$$J_{ij} = \left(\begin{bmatrix} \Sigma_{ii} & \Sigma_{ij} \\ \Sigma_{ji} & \Sigma_{jj} \end{bmatrix}^{-1} \right)_{12} \tag{5.72}$$

$$= \frac{\Sigma_{ij}}{\Sigma_{ij}^2 - \Sigma_{ii}\Sigma_{jj}} \tag{5.73}$$

The complexity of computing J_{ij} for each $(i,j) \in \mathcal{E}$ is $O(1)$ and the complexity of computing each J_{ii} is $O(\deg i)$. Since $\Sigma_{i\in\mathcal{V}} \deg(i)$ equals twice the number of edges, which is $\mathcal{O}(n)$, the total computational complexity is $\mathcal{O}(n)$. This completes the proof of Lemma 5.3.4.

□

Proof of Lemma 5.4.1

Lemma 5.4.1 : *In Algorithm 5.4.1, if Step 2(a) and Step 2(b) can be computed exactly, then we have that* $D(\hat{p}(\mathbf{x}_T)||q^{(t+1)}(\mathbf{x}_T)) \leq D(\hat{p}(\mathbf{x}_T)||q^{(t)}(\mathbf{x}_T))$, *where the equality is satisfied if and only if* $\hat{p}^{(t)}(\mathbf{x}_F, \mathbf{x}_T) = \hat{p}^{(t+1)}(\mathbf{x}_F, \mathbf{x}_T)$.

Proof. For any t,

$$D(\hat{p}^{(t)}(\mathbf{x}_T, \mathbf{x}_F)||q^{(t)}(\mathbf{x}_F, \mathbf{x}_T)) \tag{5.74}$$

$$= \int_{\mathbf{x}_T, \mathbf{x}_F} \hat{p}(\mathbf{x}_T) q^{(t)}(\mathbf{x}_F|\mathbf{x}_T) \log \frac{\hat{p}(\mathbf{x}_T) q^{(t)}(\mathbf{x}_F|\mathbf{x}_T)}{q^{(t)}(\mathbf{x}_F, \mathbf{x}_T)} \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_T \tag{5.75}$$

$$= \int_{\mathbf{x}_T, \mathbf{x}_F} \hat{p}(\mathbf{x}_T) q^{(t)}(\mathbf{x}_F|\mathbf{x}_T) \log \frac{\hat{p}(\mathbf{x}_T)}{q^{(t)}(\mathbf{x}_T)} \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_T \tag{5.76}$$

$$= \int_{\mathbf{x}_T} \hat{p}(\mathbf{x}_T) \log \frac{\hat{p}(\mathbf{x}_T)}{q^{(t)}(\mathbf{x}_T)} \mathrm{d}\mathbf{x}_T \tag{5.77}$$

$$= D(\hat{p}^{(t)}(\mathbf{x}_T)||q^{(t)}(\mathbf{x}_T)) \tag{5.78}$$

By the definition of $q^{(t+1)}$ in step (b), we have

$$D(\hat{p}(\mathbf{x}_T, \mathbf{x}_F)||q^{(t+1)}(\mathbf{x}_F, \mathbf{x}_T)) \leq D(\hat{p}^{(t)}(\mathbf{x}_T, \mathbf{x}_F)||q^{(t)}(\mathbf{x}_F, \mathbf{x}_T)). \tag{5.79}$$

Therefore,

$$D(\hat{p}(\mathbf{x}_T)||q^{(t)}(\mathbf{x}_T)) \tag{5.80}$$

$$\overset{(a)}{=} D(\hat{p}^{(t)}(\mathbf{x}_T, \mathbf{x}_F)||q^{(t)}(\mathbf{x}_F, \mathbf{x}_T)) \tag{5.81}$$

$$\overset{(b)}{\geq} D(\hat{p}^{(t)}(\mathbf{x}_T, \mathbf{x}_F)||q^{(t+1)}(\mathbf{x}_F, \mathbf{x}_T)) \tag{5.82}$$

$$= \int_{\mathbf{x}_T,\mathbf{x}_F} \hat{p}(\mathbf{x}_T) q^{(t)}(\mathbf{x}_F|\mathbf{x}_T) \log \frac{\hat{p}(\mathbf{x}_T) q^{(t)}(\mathbf{x}_F|\mathbf{x}_T)}{q^{(t+1)}(\mathbf{x}_F,\mathbf{x}_T)} \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_T \tag{5.83}$$

$$= \int_{\mathbf{x}_T,\mathbf{x}_F} \hat{p}(\mathbf{x}_T) q^{(t)}(\mathbf{x}_F|\mathbf{x}_T) \log \frac{\hat{p}(\mathbf{x}_T)}{q^{(t+1)}(\mathbf{x}_T)} \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_T$$
$$+ \int_{\mathbf{x}_T,\mathbf{x}_F} \hat{p}(\mathbf{x}_T) q^{(t)}(\mathbf{x}_F|\mathbf{x}_T) \log \frac{q^{(t)}(\mathbf{x}_F|\mathbf{x}_T)}{q^{(t+1)}(\mathbf{x}_F|\mathbf{x}_T)} \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_T \tag{5.84}$$

$$= \int_{\mathbf{x}_T} \hat{p}(\mathbf{x}_T) \log \frac{\hat{p}(\mathbf{x}_T)}{q^{(t+1)}(\mathbf{x}_T)} \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_T$$
$$+ \int_{\mathbf{x}_T,\mathbf{x}_F} \hat{p}(\mathbf{x}_T) q^{(t)}(\mathbf{x}_F|\mathbf{x}_T) \log \frac{q^{(t)}(\mathbf{x}_F|\mathbf{x}_T)\hat{p}(\mathbf{x}_T)}{q^{(t+1)}(\mathbf{x}_F|\mathbf{x}_T)\hat{p}(\mathbf{x}_T)} \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_T \tag{5.85}$$

$$= D(\hat{p}(\mathbf{x}_T)||q^{(t+1)}(\mathbf{x}_T)) + \int_{\mathbf{x}_T,\mathbf{x}_F} \hat{p}^{(t)}(\mathbf{x}_F,\mathbf{x}_T) \log \frac{\hat{p}^{(t)}(\mathbf{x}_F,\mathbf{x}_T)}{\hat{p}^{(t+1)}(\mathbf{x}_F,\mathbf{x}_T)} \mathrm{d}\mathbf{x}_F \mathrm{d}\mathbf{x}_T \tag{5.86}$$

$$= D(\hat{p}(\mathbf{x}_T)||q^{(t+1)}(\mathbf{x}_T)) + D(\hat{p}^{(t)}(\mathbf{x}_F,\mathbf{x}_T)||\hat{p}^{(t+1)}(\mathbf{x}_F,\mathbf{x}_T)) \tag{5.87}$$

$$\overset{(c)}{\geq} D(\hat{p}(\mathbf{x}_T)||q^{(t+1)}(\mathbf{x}_T)), \tag{5.88}$$

where (a) is due to (5.78), (b) is due to (5.79), and (c) is due to that

$$D(\hat{p}^{(t)}(\mathbf{x}_F,\mathbf{x}_T)||\hat{p}^{(t+1)}(\mathbf{x}_F,\mathbf{x}_T)) \geq 0. \tag{5.89}$$

Therefore, we always have $D(\hat{p}(\mathbf{x}_T)||q^{(t)}) \geq D(\hat{p}(\mathbf{x}_T)||q^{(t+1)})$.

A necessary condition for the objective function to remain the same is that

$$D(\hat{p}^{(t)}(\mathbf{x}_F,\mathbf{x}_T)||\hat{p}^{(t+1)}(\mathbf{x}_F,\mathbf{x}_T)) = 0, \tag{5.90}$$

which is equivalent to $\hat{p}^{(t)}(\mathbf{x}_F,\mathbf{x}_T) = \hat{p}^{(t+1)}(\mathbf{x}_F,\mathbf{x}_T)$.

When $\hat{p}^{(t)}(\mathbf{x}_F,\mathbf{x}_T) = \hat{p}^{(t+1)}(\mathbf{x}_F,\mathbf{x}_T)$, under non-degenerate cases, we have

$$q^{(t)}(\mathbf{x}_F,\mathbf{x}_T) = q^{(t+1)}(\mathbf{x}_F,\mathbf{x}_T) \tag{5.91}$$

according to **P2** of Algorithm 5.4.1 and thus a stationary point is reached.

Therefore, $\hat{p}^{(t)}(\mathbf{x}_F,\mathbf{x}_T) = \hat{p}^{(t+1)}(\mathbf{x}_F,\mathbf{x}_T)$ is a necessary and sufficient condition for the objective function to remain the same. This completes the proof for Lemma 5.4.1.

□

Chapter 6

Conclusion

The central theme of this thesis is providing efficient solutions to some challenging problems in probabilistic graphical models which characterize the interactions among random variables. In this chapter, we conclude this thesis by summarizing the main contributions and suggesting some future research directions.

6.1 Summary of Contributions

Recursive Feedback Message Passing for Distributed Inference

Inference problems for graphical models have become more and more challenging with the increasing popularity of very large-scale models. In particular, a purely distributed algorithm is of great importance since centralized computations are often inefficient, expensive, or impractical. In Chapter 3, we have proposed such a distributed algorithm called recursive FMP to perform inference in GGMs. Recursive FMP extends the previously developed hybrid FMP algorithm by eliminating the centralized communication and computation among the feedback nodes. In recursive FMP, nodes identify their own status (e.g., whether they behave like feedback nodes) in a distributed manner. We have shown that in recursive FMP the accuracy of the inference results are consistent with hybrid FMP while allowing much more flexibility, as different parts of the graph may use different subsets of feedback nodes. Furthermore, we have analyzed the results obtained by recursive FMP using the walk-sum framework and provided new walk-sum interpretations for the intermediate results and the added correction terms.

Sampling Gaussian Graphical Models Using Subgraph Perturbations

Efficient sampling from GGMs has become very useful not only in modeling large-scale phenomena with underlying Gaussianity, but also in statistical models where a GGM is one of several interacting components. In Chapter 4, we have proposed a general framework for converting subgraph-based iterative solvers to samplers with convergence guarantees. In particular, we have provided a construction where the injected noise at each iteration can be generated simply using a set of *i.i.d.* scalar Gaussian random variables. Moreover, we have also extended the perturbation sampling algorithm from stationary graphical splittings to non-stationary graphical splittings since using multiple subgraphs often gives much better convergence than using any of the individual subgraphs. Furthermore, we have studied the use of different kinds of tractable subgraphs and provided an algorithm to adaptively select the subgraphs based on an auxiliary inference problem.

Learning Gaussian Graphical Models with Small Feedback Vertex Sets

In general, a larger family of graphs represent a larger collection of distributions (and thus can better approximate arbitrary empirical distributions), but often lead to computationally expensive inference and learning algorithms. Hence, it is important to study the trade-off between modeling capacity and efficiency. In Chapter 5, we have studied the family of GGMs with small FVSs and presented several learning algorithms for different cases. For the case where all of the variables are observed, including any to be included in the FVS, we provided an efficient algorithm for exact ML estimation. In addition, we have given an approximate and much faster greedy algorithm for this case when the FVS is unknown and large. For a second case where the FVS nodes are taken to be latent variables, we showed the equivalence between the structure learning problem and the (exact or approximate) decomposition of an inverse covariance matrix into the sum of a tree-structured matrix and a low-rank matrix. For this case, we proposed an alternating low-rank projection algorithm for model learning and proved that even though the projections are onto a highly non-convex set, they are carried out exactly, thanks to the properties of GGMs of this family. Furthermore, we performed experiments using both synthetic data and real data of flight delays to demonstrate the modeling capacity with FVSs of various sizes.

6.2 Future Research Directions

Recursive Feedback Message Passing for Distributed Inference

The theoretical results we have presented in Chapter 3 have assumed that the local capacity and the effective diameters are sufficiently large. However, in practice, the local capacity and the effective diameters are often small due to the constraints on local resources and computational power. It is of interest to seek additional theoretical results on the convergence and accuracy in such cases. This is a challenging problem because of the complex and heterogeneous message behaviors. Moreover, the algorithms and analysis in Chapter 3 are presented assuming the underlying distributions are Gaussian. In many applications of interest, the random variables of interest are non-Gaussian (e.g., in Ising models). In such a setting, the idea of using a special message-passing protocol for a special set of nodes can still apply, leading to a hybrid message-passing algorithm similar to standard FMP. However, it is still an open problem to develop a purely distributed algorithm where all node use the same integrated protocol.

Sampling Gaussian Graphical Models Using Subgraph Perturbations

In Chapter 4, we have discussed the use of different families of tractable subgraphs. Using subgraphs in a richer family (e.g., the family of graphs with small FVSs compared with the family of tree-structured graphs) increases the computational complexity per iteration while reducing the number of iterations required for convergence. It is of interest to obtain more theoretical results on this trade-off, which may lead to new adaptive selection criteria that can choose graphs across different model families.

Learning Gaussian Graphical Models with Small Feedback Vertex Sets

In Chapter 5, our experimental results have demonstrated the potential of the proposed learning algorithms, and, as in the work [16], suggests that choosing FVSs of size $\mathcal{O}(\log n)$ works well, leading to algorithms which can be scaled to large problems. Providing theoretical guarantees for this scaling (e.g., by specifying classes of models for which such a size FVS provides asymptotically accurate models) is a compelling open problem. In addition, incorporating complexity into the FVS-order problem (e.g., as in AIC or BIC) is another direction worthy of consideration.

Bibliography

[1] D. Malioutov, J. K. Johnson, M. J. Choi, and A. S. Willsky, "Low-rank variance approximation in GMRF models: single and multiscale approaches," *IEEE Transactions on Signal Processing*, vol. 56, no. 10, pp. 4621–4634, 2008. 1, 2.1.3, 3.5, 4.5.4

[2] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 4th ed. Cengage Learning, 2014. 1, 2.1.3

[3] D. Heckerman and J. Breese, "Causal independence for probability assessment and inference using Bayesian networks," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 26, no. 6, pp. 826–831, 1996. 1

[4] C. Wunsch and P. Heimbach, "Practical global oceanic state estimation," *Physica D: Nonlinear Phenomena*, vol. 230, no. 1-2, pp. 197–208, 2007. 1, 1.1, 2.1.3

[5] L.-W. Yang, X. Liu, C. J. Jursa, M. Holliman, A. Rader, H. A. Karimi, and I. Bahar, "iGNM: a database of protein functional motions based on Gaussian Network Model," *Bioinformatics*, vol. 21, no. 13, p. 2978, 2005. 1.1, 2.1.3

[6] Y. Weiss and W. Freeman, "Correctness of belief propagation in Gaussian graphical models of arbitrary topology," *Neural Computation*, vol. 13, no. 10, pp. 2173–2200, 2001. 1.1

[7] K. Murphy, Y. Weiss, and M. Jordan, "Loopy belief propagation for approximate inference: an empirical study," in *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1999, pp. 467–475. 1.1, 2.2.1

[8] D. Bickson, O. Shental, and D. Dolev, "Distributed Kalman filter via Gaussian belief propagation," in *46th Annu. Allerton Conf. Commun., Control, and Computing*, 2008, pp. 628–635. 1.1

[9] M. Wainwright and M. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends in Machine Learning*, vol. 1, no. 1-2, pp. 1 305, 2008. 1.1

[10] J. Yedidia, W. Freeman, and Y. Weiss, "Constructing free-energy approximations and generalized belief propagation algorithms," *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2282–2312, 2005. 1.1

[11] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "Tree-based reparameterization framework for analysis of sum-product and related algorithms," *IEEE Transactions on Information Theory*, vol. 49, no. 5, pp. 1120–1146, 2003. 1.1

[12] D. Dolev, D. Bickson, and J. K. Johnson, "Fixing convergence of gaussian belief propagation," in *Information Theory, 2009. ISIT 2009. IEEE International Symposium on.* IEEE, 2009, pp. 1674–1678. 1.1

[13] Y. El-Kurdi, D. Giannacopoulos, and W. J. Gross, "Relaxed gaussian belief propagation," in *Proceedings of the International Symposium on Information Theory Proceedings (ISIT).* IEEE, 2012, pp. 2002–2006. 1.1

[14] C. C. Moallemi and B. Van Roy, "Convergence of min-sum message passing for quadratic optimization," *IEEE Transactions on Information Theory*, vol. 55, no. 5, pp. 2413–2423, 2009. 1.1

[15] N. Ruozzi and S. Tatikonda, "Message-passing algorithms for quadratic minimization," *arXiv preprint arXiv:1212.0171*, 2012. 1.1

[16] Y. Liu, V. Chandrasekaran, A. Anandkumar, and A. S. Willsky, "Feedback message passing for inference in Gaussian graphical models," *IEEE Transactions on Signal Processing*, vol. 60, no. 8, pp. 4135–4150, 2012. 1.1, 1.2, 1.3, 2.2.3, 3.1, 3.2.1, 6, 8, 11, 3.2.2, 4.1, 4.4.1, 5.1, 5.6, 6.2

[17] M. K. Titsias, N. D. Lawrence, and M. Rattray, "Efficient sampling for Gaussian process inference using control variables," *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, vol. 21, pp. 1681–1688, 2008. 1.2

[18] R. Salakhutdinov, "Learning deep Boltzmann machines using adaptive MCMC," in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 27, 2010. 1.2

[19] A. E. Gelfand, "Model determination using sampling-based methods," in *Markov Chain Monte Carlo in Practice*, W. Gilks, R. S., and S. D.J., Eds., 1996, pp. 145–161. 1.2

[20] S. Fine, Y. Singer, and N. Tishby, "The hierarchical hidden Markov model: analysis and applications," *Machine learning*, vol. 32, no. 1, pp. 41–62, 1998. 1.2

[21] T. Kailath, A. H. Sayed, and B. Hassibi, *Linear Estimation.* Prentice-Hall, 2000, vol. 1. 1.2

[22] K. Daoudi, A. B. Frakt, and A. S. Willsky, "Multiscale autoregressive models and wavelets," *IEEE Transactions on Information Theory*, vol. 45, no. 3, pp. 828–845, 1999. 1.2

[23] M. I. Jordan, "Graphical models," *Statistical Science*, pp. 140–155, 2004. 1.2, 1.3, 2.1, 2.1.2, 2.1.3, 2.2.1, 2.3

[24] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*. Citeseer, 2004. 1.2, 2.3, 4.1

[25] Y. Amit and U. Grenander, "Comparing sweep strategies for stochastic relaxation," *Journal of Multivariate Analysis*, vol. 37, no. 2, pp. 197–222, 1991. 1.2

[26] A. Thomas, A. Gutin, V. Abkevich, and A. Bansal, "Multilocus linkage analysis by blocked Gibbs sampling," *Statistics and Computing*, vol. 10, no. 3, pp. 259–269, 2000. 1.2

[27] H. Rue, "Fast sampling of Gaussian Markov random fields," *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, vol. 63, no. 2, pp. 325–338, 2001. 1.2

[28] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling, "Fast collapsed Gibbs sampling for latent Dirichlet allocation," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, 2008, pp. 569–577. 1.2

[29] F. Hamze and N. de Freitas, "From fields to trees," in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004, pp. 243 250. 1.2, 2.3, 4.5.1

[30] G. Papandreou and A. L. Yuille, "Gaussian sampling by local perturbations," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2010, pp. 1858–1866. 1.2

[31] T. A. Davis, *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2006. 1.2, 9

[32] E. B. Sudderth, M. J. Wainwright, and A. S. Willsky, "Embedded trees: estimation of Gaussian processes on graphs with cycles," *IEEE Transactions on Signal Processing*, vol. 52, no. 11, pp. 3136–3150, 2004. 1.2, 4.1, 4.2.1, 4.2.1, 4.2.2, 4.2.2, 4.3, 4.3

[33] V. Chandrasekaran, J. K. Johnson, and A. S. Willsky, "Estimation in Gaussian graphical models using tractable subgraphs: a walk-sum analysis," *IEEE Transactions on Signal Processing*, vol. 56, no. 5, pp. 1916–1930, 2008. 1.2, 2.2.2, 4.1, 4.2.1, 4.3, 4.3, 4.3, 2, 4.4.1, 4.4.2

[34] J. M. Ortega, *Numerical Analysis: a Second Course.* Society for Industrial and Applied Mathematics, 1990. 1.2, 4.1, 4.2.2

[35] J. Pearl, "A constraint propagation approach to probabilistic reasoning," *Proceedings of the 2nd Conference on Uncertainty in Artificial Intelligence (UAI)*, 1986. 1.3

[36] C. Chow and C. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transactions on Information Theory*, vol. 14, no. 3, pp. 462–467, 1968. 1.3, 2.4.3

[37] M. J. Choi, V. Chandrasekaran, and A. S. Willsky, "Exploiting sparse Markov and covariance structure in multiresolution models," in *Proceedings of the 26th Annual International Conference on Machine Learning (ICML).* ACM, 2009, pp. 177–184. 1.3

[38] M. L. Comer and E. J. Delp, "Segmentation of textured images using a multiresolution Gaussian autoregressive model," *IEEE Transactions on Image Processing*, vol. 8, no. 3, pp. 408–420, 1999. 1.3

[39] C. A. Bouman and M. Shapiro, "A multiscale random field model for Bayesian image segmentation," *IEEE Transactions on Image Processing*, vol. 3, no. 2, pp. 162–177, 1994. 1.3

[40] D. Karger and N. Srebro, "Learning Markov networks: maximum bounded tree-width graphs," in *Proceedings of the 12th annual ACM-SIAM symposium on Discrete algorithms.* Society for Industrial and Applied Mathematics, 2001, pp. 392–401. 1.3, 4.4.1

[41] P. Abbeel, D. Koller, and A. Y. Ng, "Learning factor graphs in polynomial time and sample complexity," *The Journal of Machine Learning Research (JMLR)*, vol. 7, pp. 1743–1788, 2006. 1.3

[42] A. Dobra, C. Hans, B. Jones, J. R. Nevins, G. Yao, and M. West, "Sparse graphical models for exploring gene expression data," *Journal of Multivariate Analysis*, vol. 90, no. 1, pp. 196–212, 2004. 1.3

[43] M. E. Tipping, "Sparse Bayesian learning and the relevance vector machine," *The Journal of Machine Learning Research (JMLR)*, vol. 1, pp. 211–244, 2001. 1.3

[44] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," *Biostatistics*, vol. 9, no. 3, pp. 432–441, 2008. 1.3

[45] P. Ravikumar, G. Raskutti, M. Wainwright, and B. Yu, "Model selection in Gaussian graphical models: High-dimensional consistency of l_1-regularized MLE," *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, vol. 21, 2008. 1.3

[46] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2, pp. 131–163, 1997. 1.3

[47] V. Chandrasekaran, P. A. Parrilo, and A. S. Willsky, "Latent variable graphical model selection via convex optimization," in *Proceedings of the 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2010, pp. 1610–1613. 1.3

[48] S. Lauritzen, *Graphical Models*. New York: Oxford University Press, 1996. 2.1.2

[49] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, "Using Bayesian networks to analyze expression data," *Journal of Computational Biology*, vol. 7, no. 3-4, pp. 601 620, 2000. 2.1.3

[50] Y. Zhang, M. Brady, and S. Smith, "Segmentation of brain MR images through a hidden Markov random field model and the expectation-maximization algorithm," *IEEE Transactions on Medical Imaging*, vol. 20, no. 1, pp. 45–57, 2001. 2.1.3

[51] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Sel. Areas in Commun.*, vol. 16, no. 2, pp. 219 230, 1998. 2.1.3

[52] C. Crick and A. Pfeffer, "Loopy belief propagation as a basis for communication in sensor networks," in *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI)*, vol. 18, 2003. 2.2.1

[53] R. McEliece, D. MacKay, and J. Cheng, "Turbo decoding as an instance of Pearl's belief propagation algorithm," *IEEE J. Select. Areas in Commun.*, vol. 16, no. 2, pp. 140–152, 1998. 2.2.1

[54] D. M. Malioutov, J. K. Johnson, and A. S. Willsky, "Walk-sums and belief propagation in Gaussian graphical models," *The Journal of Machine Learning Research (JMLR)*, vol. 7, pp. 2031–2064, 2006. 2.2.1, 2.2.2, 4, 2.2.2, 4.3

[55] V. V. Vazirani, *Approximation Algorithms*. New York: Springer, 2004. 2.2.3

[56] Y. Liu, "Feedback message passing for inference in Gaussian graphical models," Master's thesis, Massachusetts Institute of Technology, 2010. 2.2.3, 2.2.3

[57] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin, "Parallel Gibbs sampling: from colored fields to thin junction trees," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011, pp. 324–332. 2.3

[58] K. P. Murphy, *Machine learning: a probabilistic perspective.* MIT Press, 2012. 2.4.2, 5.3.1

[59] C. Bishop, *Pattern Recognition and Machine Learning.* New York: Springer, 2006. 2.4.2

[60] N. A. Lynch, *Distributed algorithms.* Morgan Kaufmann, 1996. 3.2, 3.2.1

[61] C. Fox and A. Parker, "Convergence in variance of Chebyshev accelerated Gibbs samplers," *SIAM Journal of Scientific Computing*, vol. 36, pp. 124–147, 2013. 4.1

[62] A. Galli and H. Gao, "Rate of convergence of the Gibbs sampler in the Gaussian case," *Mathematical Geology*, vol. 33, no. 6, pp. 653–677, 2001. 4.1

[63] Y. Liu and A. S. Willsky, "Learning Gaussian graphical models with observed or latent FVSs," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 1833–1841. 4.2.1

[64] D. P. O'Leary and R. E. White, "Multi-splittings of matrices and parallel solution of linear systems," *SIAM Journal on Algebraic Discrete Methods*, vol. 6, no. 4, pp. 630–640, 1985. 4.2.2

[65] A. J. Laub, *Matrix Analysis for Scientists and Engineers.* Society of Industrial and Applied Mathematics, 2005. 4.2.2

[66] O. Axelsson, "Bounds of eigenvalues of preconditioned matrices," *SIAM Journal on Matrix Analysis and Application*, vol. 13, no. 3, pp. 847–862, 1992. 4.2.2

[67] R. A. Horn and C. R. Johnson, *Matrix Analysis.* Cambridge University Press, 1990. 4.2.2, 5.3.1

[68] M. Bern, J. R. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo, "Support-graph preconditioners," *SIAM Journal on Matrix Analysis and Application*, vol. 27, no. 4, pp. 930–951, 2006. 4.4.1

[69] N. Srebro, "Maximum likelihood bounded tree-width Markov networks," in *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2001, pp. 504–511. 4.4.1

[70] D. Shahaf and C. Guestrin, "Learning thin junction trees via graph cuts," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009, pp. 113–120. 4.4.1

[71] D. A. Spielman and S.-H. Teng, "Spectral sparsification of graphs," *SIAM Journal on Computing*, vol. 40, p. 981, 2011. 4.4.1

[72] M. Desjarlais and R. Molina, "Counting spanning trees in grid graphs," *Congressus Numerantium*, pp. 177–186, 2000. 4

[73] V. Bafna, P. Berman, and T. Fujito, "A 2-approximation algorithm for the undirected feedback vertex set problem," *SIAM Journal on Discrete Mathematics*, vol. 12, p. 289, 1999. 5.1, 5.2

[74] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Machine learning*, vol. 37, no. 2, pp. 183–233, 1999. 5.2

[75] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "A new class of upper bounds on the log partition function," *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2313–2335, 2005. 5.2

[76] J. Ma, J. Peng, S. Wang, and J. Xu, "Estimating the partition function of graphical models using langevin importance sampling," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2013, pp. 433–441. 5.2

[77] S. Kirshner, P. Smyth, and A. W. Robertson, "Conditional Chow-Liu tree structures for modeling discrete-valued vector time series," in *Proceedings of the 20th conference on Uncertainty in Artificial Intelligence (UAI)*, 2004, pp. 317–324. 3

[78] M. A. Figueiredo, J. M. Bioucas-Dias, and R. D. Nowak, "Majorization–minimization algorithms for wavelet-based image restoration," *IEEE Transactions on Image Processing*, vol. 16, no. 12, pp. 2980–2991, 2007. 5.4.1

[79] M. J. Choi, V. Y. Tan, A. Anandkumar, and A. S. Willsky, "Learning latent tree graphical models," *The Journal of Machine Learning Research (JMLR)*, vol. 12, pp. 1771–1812, 2011. 5.5.1, 5.1