UNIVERSITY OF CALIFORNIA,
IRVINE

Bottom-Up Approaches to Approximate Inference and Learning in Discrete Graphical Models

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Andrew Edward Gelfand

2014

DEDICATION

To my loving and ever-supportive wife Katie,
and my wonderful mother, father and sister.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to thank my core group of advisors Rina Dechter, Alex Ihler and Max Welling. I was incredibly fortunate to have worked on a wide range of exciting research problems while at UCI, but the peripatetic nature of my Ph.D. would not have led to completion without your collective guidance and support. Your complementary perspectives on graphical models and scientific research have led me to be a far more well-rounded researcher and provided insights that will undoubtedly shape my career going forward.

I am also indebted to my fourth committee member, Misha Chertkov, who took me under his wing while I visited Los Alamos and exposed me to the literature on linear programming decoding, matchings and permanents. Your physics-oriented viewpoint on graphical models, not only led to many thoroughly enjoyable discussions, but also broadened my horizons as a researcher.

I am also thankful to the bookends of my Ph.D. – Kalev Kask and Jinwoo Shin. Kalev guided me on my very first research project at UCI, a distributed implementation of bucket elimination, which ultimately led to my first publication at UCI. Jinwoo is a true mathematical magician and working with him on the weighted matching problem was both an inspiring and rewarding experience.

I was also helped by many other faculty, colleagues and friends along the way. The Dechter group, Welling group and Ihler group all contain some truly phenomenal individuals, without whom this thesis would not have been possible. I would also like to acknowledge my good friend David Stenning: I will always have fond memories of our "coffee" discussions and even fonder memories of our "pub" discussions.

Last, and most importantly, I want to thank my wonderful family. Mom, Dad and Sarah, your continued belief in me provided with the confidence I needed to make it to the end of this journey. Tom and Laura, your well-timed words of wisdom and encouragement no doubt made our journey more enjoyable. Finally, to my loving wife Katie, you simultaneously provided me with unwavering support and constant motivation during this quest and I am forever indebted to you for that.

I am grateful for the assistance I have received for my graduate studies from NSF awards IIS-0713118, IIS-1065618, NIH grant 5R01HG004175-03, NSF Grant No. 0914783, 0928427, 1018433, my funding from Los Alamos National Lab, and a Deans Fellowship at the Donald Bren School of Information and Computer Sciences.

CURRICULUM VITAE

Andrew Edward Gelfand

EDUCATION

Ph.D. in Computer Science	**April 2014**
University of California, Irvine	*Irvine, CA*
M.S. in Systems Engineering	**June 2009**
The George Washington University	*Washington, DC*
B.S. in Computer Engineering	**June 2004**
Tufts University	*Medford, MA*

RESEARCH EXPERIENCE

Graduate Research Assistant	**2009–2014**
University of California, Irvine	*Irvine, California*
Visiting Scholar	**2012–2013**
Los Alamos National Lab	*Los Alamos, New Mexico*

TEACHING EXPERIENCE

Teaching Assistant	**2011–2012**
University of California, Irvine	*Irvine, CA*

SELECTED HONORS AND AWARDS

Dean's Fellowship, Information and Computer Sciences	**2009–2013**
University of California, Irvine	

BOOK CHAPTERS

Herding for Structured Prediction **2014**
In: Advanced Structured Prediction, S. Nowozin, P. Gehler, J. Jancsary, & C. Lampert (Eds)

REFEREED CONFERENCE PUBLICATIONS

A Graphical Transformation for Belief Propagation: Maximum Weight Matchings and Odd-Sized Cycles **Dec 2013**
NIPS

Loop Calculus and Bootstrap-Belief Propagation for Perfect Matchings on Arbitrary Graphs **July 2013**
Proceedings of the International Meeting on 'Inference, Computation, and Spin Glasses'

Belief Propagation for Linear Programming **July 2013**
ISIT

Generalized Belief Propagation on Tree Robust Structured Region Graphs **Aug 2012**
UAI

A Cluster-Cumulant Expansion at the Fixed Points of Belief Propagation **Aug 2012**
UAI

Integrating Local Classifiers through Nonlinear Dynamics on Label Graphs with an Application to Image Segmentation **Nov 2011**
ICCV

Stopping Rules for Randomized Greedy Triangulation Schemes **Aug 2011**
AAAI

Pushing the Power of Stochastic Greedy Ordering Schemes for Inference in Graphical Models **Aug 2011**
AAAI

On Herding and the Perceptron Cycling Theorem **Dec 2010**
NIPS

BEEM: Bucket Elimination with External Memory **July 2010**
UAI

SOFTWARE

runGBP `https://github.com/aegelfand/GBP`
C++ project containing implementations of inference in discrete probabilistic graphical models using several variants of Generalized Belief Propagation (GBP) message passing.

ABSTRACT OF THE DISSERTATION

Bottom-Up Approaches to Approximate Inference and Learning in Discrete Graphical Models

By

Andrew Edward Gelfand

Doctor of Philosophy in Computer Science

University of California, Irvine, 2014

Professors Rina Dechter, Alexander Ihler, Co-Chairs

Probabilistic graphical models offer a convenient and compact way to describe complex and uncertain relationships in data. A graphical model defines a joint probability distribution over many random variables that factors over an underlying graph structure. Unfortunately, inference is generally intractable in graphical models which accurately describe the complex dependencies occurring in real data.

In this thesis, we focus on theory and algorithms for learning and approximate inference in graphical models. We propose and investigate a bottom-up approach to inference and learning, where we start with an initial, computationally cheap approximation and then improve upon the initial approximation through additional computation. We study the computation-accuracy trade-off inherent to the bottom-up approach in three different settings.

First, we consider the task of finding the most probable (MAP) configuration of a model. We focus on a class of graphical models corresponding to the weighted matching problem – a classic combinatorial optimization problem – and on MAP inference algorithms based on linear programming (LP) relaxations. In this setting, the optimum of the LP relaxation provides an upper bound on the MAP solution to the weighted matching problem that may be quite loose. We thus propose a bottom-up, cutting-plane algorithm which iteratively adds constraints that tighten the upper bound on the matching solution. We then derive a max-product belief propagation algorithm that provably solves the matching problem for certain choices of tightening constraints.

Second, we consider the task of computing the marginal probabilities of a model. Loopy Belief Propagation (BP) is an algorithm for obtaining marginal probability estimates by iteratively passing messages between neighboring nodes in a cyclic graphical model. Generalized Belief Propagation (GBP) is a class of approximate inference algorithms that builds upon Loopy BP by passing messages between clusters of nodes. GBP offers the promise to yield marginal estimates that are far more accurate than Loopy BP, but is also very sensitive to the choice of clusters used. We thus propose a criteria – tree-robustness – for choosing the collection of clusters used by GBP that is, in some sense, no worse than Loopy BP when the factors defining our model induce a tree. We propose a method to find a collection of clusters that are tree-robust and empirically demonstrate the effectiveness of the proposed criteria.

Third, we consider the task of learning the parameters of a model from data. Maximum likelihood estimation in graphical models is difficult to the intractability of computing the log-partition function and marginals. In *surrogate* likelihood training, one approximates these quantities using an approximate inference algorithm. We focus on approximate inference methods that utilize a control parameter to trade computation for accuracy and examine when investing more computation leads to more accurate parameter estimates and models that yield more accurate predictions. Surprisingly, we show that it is not always beneficial to increase computation during learning, particularly in data sets containing relatively few observations and when the model being fit is heavily misspecified. We also expose an interesting bias-variance trade-off between low and high computation inference methods.

Chapter 1

Introduction & Background

Statistical modeling is fundamentally concerned with building models that relate data, or observations, to quantities of interest. For example, given an observed image we might like to determine if the image contains a human being, a dog, or possibly a cat. Or, given a log of historical activity at an online retailer, we might like to determine a set of items that each user is likely to purchase. Statistical modeling is a powerful tool that can help extract such meaningful insight from data.

Probabilistic graphical models are a specific class of statistical model that provide a principled way to describe complex and uncertain relationships among observed and unknown variables[48, 53, 18]. A graphical model defines a joint probability distribution over many random variables that factors over an underlying graph structure. The graph structure makes the model's assumptions explicit and easily communicable to others. More importantly, the model's structure can be exploited by algorithms for efficient inference and learning.

This thesis focuses on theory and algorithms for both inference and learning in graphical models. Since exact inference is often infeasible in models with rich variable inter-dependencies, we focus primarily on approximate methods. In particular, we advocate a bottom-up approach to approximate inference and learning, where we start with some initial, base approximation to our problem and then improve upon the base approximation by increasing our computational budget. We study the bottom-up approach in three different tasks.

In Chapter 2 we consider the task of finding the most probable (or MAP) configuration in a model. We focus on a specific class of graphical model corresponding to the weighted matching problem – a classic combinatorial optimization problem – and on MAP inference algorithms based on linear programming (LP) relaxations. In this setting, the optimum of the LP relaxation provides an upper bound on the MAP solution to the weighted matching problem that may be loose. We thus propose a bottom-up, cutting-plane strategy which iteratively adds constraints that tighten the upper bound on the matching solution. We then derive a max-product belief propagation algorithm that provably solves the matching problem when the upper-bound is made tight.

In Chapter 3 we consider the task of calculating marginal probabilities in a graphical model. We focus on Loopy Belief Propagation (BP) and Generalized Belief propagation (GBP), two popular algorithms for approximately computing marginals. The Loopy BP algorithm is well-defined given a graphical model: just iteratively send messages between the nodes of the model. The GBP algorithm expands upon Loopy BP by grouping the model's nodes into clusters and sending messages between the clusters. It is not a well-defined algorithm, however, as many different clusterings are possible given a graphical model. Some of these clusterings will produce more accurate marginal estimates than Loopy BP, while others may not. We thus propose a criterion – tree-robustness – for selecting the collection of clusters used by GBP that is guaranteed, in some sense, to be no worse than Loopy BP. We propose a method to find a collection of clusters that are tree-robust and empirically demonstrate the effectiveness of the criteria.

In Chapter 4 we consider the task of learning the parameters of a model from data using maximum likelihood estimation. Finding the max likelihood estimate is not possible in models where inference is intractable. In such settings, it is common to use approximate inference and maximize an approximation to the likelihood function, known as the *surrogate likelihood* [87]. We study the effect of using different approximate methods and, therefore, different surrogate likelihoods in the joint estimation and prediction problem, where one first estimates the parameters of a model and then uses the fitted model to make predictions. In particular, we consider inference methods that utilize a control parameter to trade computation for accuracy. In our experiments, we identify regimes where it is beneficial to invest more computation and optimize a more accurate surrogate to the true likelihood. We also identify regimes where the increased computation and accuracy is actually detrimental.

1.1 Contributions

The main contributions of this dissertation are summarized by chapter as follows:

Chapter 2:

- We describe how to formulate Integer Linear Programming problems (ILPs), such as the weighted matching problem, as an equivalent probabilistic graphical model.
- We characterize when the linear programming (LP) relaxation of an ILP is equivalent to the pairwise LP relaxation of the MAP inference problem in its corresponding graphical model. This elaborates the set of ILPs that can (potentially) be solved by approximate inference methods based on pairwise LP relaxations, such as max-product belief propagation (BP).
- We expand the class of weighted matching problems (and ILPs in general) that are provably solvable by max-product BP to include matching problems whose LP relaxations are made tight by adding certain collections of "blossom" constraints. Prior

to this work, only matching problems whose natural LP relaxation was tight could provably be solved by max-product BP.

- We propose a cutting-plane BP algorithm for solving weighted matchings that tightens the LP relaxation by iteratively adding "blossom" constraints and employs max-product BP as its LP solver.

Chapter 3:

- We introduce a new criterion – tree-robustness – for choosing the form of the Kikuchi approximation to the entropy used in variational approximations solved by Generalized Belief Propagation (GBP). The Kikuchi entropy uses a combination of marginal entropies over clusters of variables to approximate the entropy. Tree-robustness requires the Kikichi entropy to be exact on every tree-structured sub-model, where a tree-structured sub-model is formed using a subset of the input model's factors that are tree-structured. We empirically demonstrate the effectiveness of the tree-robustness criterion and justify its accuracy using properties of the free energy.

- We propose a method to automatically choose collections of clusters satisfying the tree-robustness criteria and existing, "common-sense" criteria for Loop-Structured Region Graphs (SRGs) – a specific family of variational approximations. To our knowledge, this is the first automated bottom-up approach to constructing GBP approximations on arbitrary graphical models[1].

[1]In contrast, Dechter et al.[20, 62] propose a top-down scheme based on partitioning clusters. In addition, Welling [94] proposed a region-pursuit method that added clusters in a greedy manner, without satisfying either tree-robustness or existing "common-sense" criteria.

Chapter 4:

- We propose a framework for studying the error introduced by using approximate inference in graphical models where maximum likelihood estimation is intractable. We then use this framework to study the error introduced by approximate methods in parameter estimation and when the fitted model is used for making predictions.
- We perform an empirical evaluation with approximate inference methods that utilize a control parameter to trade computation for accuracy. We empirically demonstrate that better (i.e. more computationally demanding) inference does not necessarily result in learning a better model, particularly for small sized data sets and mis-specified models. We also characterize the bias and variance of different inference-based estimation and prediction strategies.

1.2 Discrete Graphical Models

Let $\boldsymbol{y} = \{y_1, ..., y_m\} \in \mathcal{Y}^m$ denote an assignment to the random variables $Y_1, ..., Y_m$, where each random variable is assumed to be K-ary so that $\mathcal{Y} = \{0, 1, ..., K-1\}$. We will use y_i to denote the assignment $Y_i = y_i$ to random variable Y_i. Similarly, we will use $\boldsymbol{y}$ to denote the assignment $\boldsymbol{Y} = \boldsymbol{y}$ to the collection of random variables Y.

Probabilistic graphical models offer a convenient and compact way to describe probability distributions over $\boldsymbol{Y}$. A graphical model defines a joint probability distribution with respect to an underlying graph $G = (V, E)$. Each random variable is associated with a vertex and the variable dependencies are specified by the graphs edge structure. We will index the variables in a graphical model as either $i = 1...m$, or $i \in V$ (meaning that $m = |V|$).

Graphical models come in two basic forms: directed and undirected. Directed graphical models, or *Bayesian Networks*, specify a joint probability distribution over $\boldsymbol{y}$ by a directed acyclic graph $G = (V, E)$ and the factorization [72]:

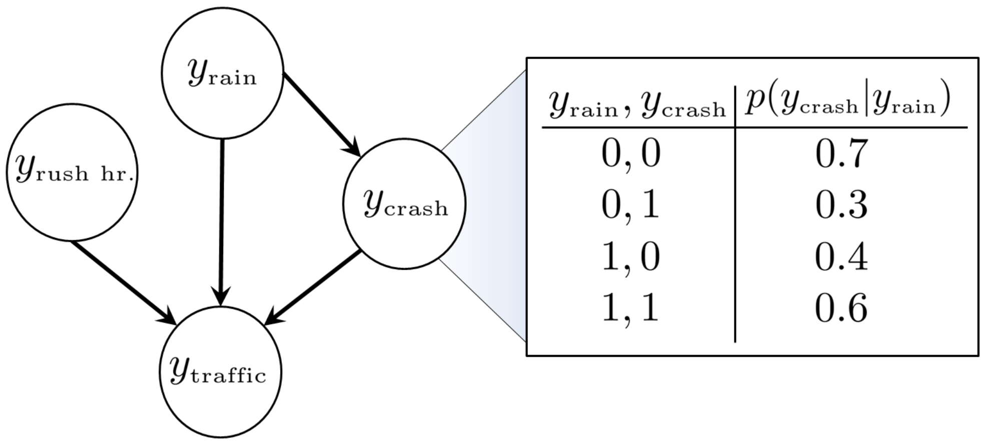
$$p(\boldsymbol{Y} = \boldsymbol{y}) = p(\boldsymbol{y}) = \prod_{i \in V} p(y_i | \boldsymbol{y}_{pa_{G(i)}}), \tag{1.1}$$

where $p(y_i | \boldsymbol{y}_{pa_{G(i)}})$ is a conditional probability distribution and $pa_{G(i)}$ is the set of parents of node i in graph G.

Figure 1.1 illustrates a simple Bayesian network model of traffic flow. The joint distribution over the 4 binary variables in this model factors in a manner consistent with the directed acyclic graph structure

$$p(\boldsymbol{y}) = p(y_{\text{rush hr.}})p(y_{\text{rain}})p(y_{\text{crash}}|y_{\text{rain}})p(y_{\text{traffic}}|y_{\text{rush hr.}}, y_{\text{rain}}, y_{\text{crash}}).$$

The model tells us, for example, that the probability of a crash is influenced by the presence of rain, where the conditional distribution $p(y_{\text{crash}}|y_{\text{rain}})$ is defined by the table of values given in Figure 1.1.



$y_{\text{rain}}, y_{\text{crash}}$	$p(y_{\text{crash}}\|y_{\text{rain}})$
0, 0	0.7
0, 1	0.3
1, 0	0.4
1, 1	0.6

Figure 1.1: A Bayesian network model of traffic flow.

The second type of graphical model is an undirected graphical model, or *Markov Random*

Field (MRF) [53]. An MRF defines a joint probability as

$$p(\boldsymbol{y}) = \frac{1}{Z} \prod_{C \in \mathcal{C}(G)} \psi_C(\boldsymbol{y}_C), \tag{1.2}$$

where $\mathcal{C}(G)$ is the set of maximal cliques[2] in graph G, $\boldsymbol{y}_C = \{y_i : i \in C\}$ is the subset of variables in clique C and Z is a normalization constant given by

$$Z = \sum_{\boldsymbol{y} \in \mathcal{Y}^m} \prod_{C \in \mathcal{C}(G)} \psi_C(\boldsymbol{y}_C). \tag{1.3}$$

The functions $\psi_C(\boldsymbol{y}_C)$ are positive, $\psi_C : \mathcal{Y}_C^{|C|} \mapsto \mathbb{R}^+$, and define a local interaction between the set of variables in clique C. Unlike Bayesian networks, these functions, or *factors*, need not be normalized probability distributions and can take arbitrary positive values. The functions ψ_C are required to be positive, however, to satisfy the conditions of the Hammersley-Clifford theorem[6], which tell us when a joint probability distribution $p(\boldsymbol{y})$ that is Markov with respect to G, can be written in the factorized form in (1.2). The normalization constant, Z, is often referred to as the partition function and it involves a sum over all of the joint assignments to $\boldsymbol{Y}$.

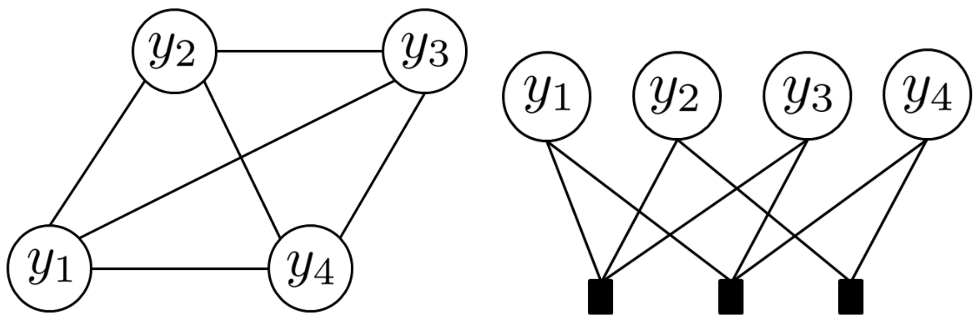
Figure 1.2a illustrates the undirected, or *moral* graph G, for an MRF on 4 variables. In Chapters 3 and 4, we will restrict attention to pairwise MRFs. In a *pairwise MRF*, the maximal cliques of G are assumed to be of cardinality two. As a result, the joint distribution over $\boldsymbol{y}$ for the pairwise MRF in Figure 1.2a factors as:

$$p(\boldsymbol{y}) = \frac{1}{Z}\psi(y_1, y_2)\psi(y_1, y_3)\psi(y_1, y_4)\psi(y_2, y_3)\psi(y_2, y_4)\psi(y_3, y_4)\psi(y_1)\psi(y_2)\psi(y_3)\psi(y_4). \tag{1.4}$$

Note that the undirected graph, G, underlying an MRF does not make the factorization of $p(\boldsymbol{y})$ explicit. In other words, just by looking at G we cannot tell if $p(\boldsymbol{y})$ factorizes as $p(\boldsymbol{y}) \propto$

[2] A clique is a complete subgraph in G, meaning that every two vertices in C are connected by an edge.

$\psi(y_1, y_2, y_3, y_4)$ or via the pairwise factorization in (1.4). The factor graph specification of a joint distribution, which we introduce next, removes this ambiguity.



(a) MRF on 4 variables. (b) Factor Graph on 4 variables.

Figure 1.2: A Markov Random Field and Factor Graph on a fully-connected graph.

A *factor graph* provides a compact representation of the factorization of a joint probability distribution[56]. A factor graph is a bipartite graph $G = (V, F, E)$, where V and F are two sets of vertices and $E \subseteq V \times F$ is a set of undirected edges. The vertices in V are referred to as variable nodes and depicted by circles when visualized. The vertices in F are referred to as factor nodes and depicted as squares. Let $\delta(\alpha) = \{i \in V : (i, \alpha) \in E\}$ denote the neighbors of factor α in factor graph G. A factor graph defines a joint probability distribution as:

$$p(\boldsymbol{y}) = \frac{1}{Z} \prod_{\alpha \in F} \psi_\alpha(\boldsymbol{y}_\alpha), \tag{1.5}$$

where each $\psi_\alpha(\boldsymbol{y}_\alpha)$ is a positive function over the subset of variables adjacent to factor α, $\boldsymbol{y}_\alpha = \{y_i : i \in \delta(\alpha)\}$, and Z is once again the partition function.

Figure 1.2b depicts a factor graph that specifies a joint distribution that factorizes as

$$p(\boldsymbol{y}) = \frac{1}{Z} \psi_{123}(y_1, y_2, y_3) \psi_{134}(y_1, y_3, y_4) \psi_{24}(y_2, y_4).$$

Often times we are interested in modeling the conditional probability of $\boldsymbol{y}$ given an observation $\boldsymbol{X} = \boldsymbol{x}$. The conditional distribution $p(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x})$ can be expressed as a Conditional Random Field (CRF)[57], which factors as

$$p(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x}) = p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{\alpha} \psi_{\alpha}(\boldsymbol{y}_{\alpha}, \boldsymbol{x}), \tag{1.6}$$

where $\psi_{\alpha}(\boldsymbol{y}_{\alpha}, \boldsymbol{x})$ is a function over $\boldsymbol{y}_{\alpha}$ whose value depends on the observation $\boldsymbol{x}$ and

$$Z(\boldsymbol{x}) = \sum_{\boldsymbol{y} \in \mathcal{Y}} \prod_{\alpha} \psi_{\alpha}(\boldsymbol{y}_{\alpha}, \boldsymbol{x}) \tag{1.7}$$

is a normalization constant dependent on the observation $\boldsymbol{x}$. The dependence of the factor $\psi_{\alpha}(\boldsymbol{y}_{\alpha}, \boldsymbol{x})$ on $\boldsymbol{x}$ can be very complex. For example, in Chapter 4 we will use a CRF model that integrates RGB color information to determine whether each pixel in an image is foreground or background.

1.2.1 Exponential Families

The discrete probabilistic graphical models discussed so far can also be viewed as members of the exponential family of distributions. Exponential families have been widely studied in the statistics literature and their introduction here will facilitate our discussion of variational inference in Section 1.4.2. In particular, the inference tasks of computing marginals, likelihoods and most probable (MAP) configurations can all be interpreted as finding mappings between different parameterizations of an exponential family distribution.

A probability distribution in the exponential family can be written as

$$p(\boldsymbol{Y} = \boldsymbol{y}; \boldsymbol{\theta}) = \exp\left(\boldsymbol{\theta} \cdot \boldsymbol{s}(\boldsymbol{y}) - \log Z(\boldsymbol{\theta})\right), \tag{1.8}$$

$$\log Z(\boldsymbol{\theta}) = \log \sum_{\boldsymbol{y}} \exp\left(\boldsymbol{\theta} \cdot s(\boldsymbol{y})\right), \tag{1.9}$$

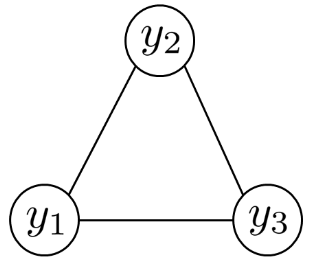
where $\boldsymbol{s}(\boldsymbol{y}) = (s_1(\boldsymbol{y}), ..., s_d(\boldsymbol{y}))$ is a d-dimensional vector of sufficient-statistics, $\boldsymbol{\theta} = (\theta_1, .., \theta_d) \in \mathbb{R}^d$ is a vector of *canonical* parameters and the log-partition function, $\log Z(\boldsymbol{\theta})$, ensures that the density function is properly normalized. Note that $\boldsymbol{\theta} \cdot \boldsymbol{s}(\boldsymbol{y})$ is the inner product between the parameters and sufficient statistics.

The sufficient statistics can be arbitrary features of $\boldsymbol{y}$ and different sufficient statistics define different probability distributions. For example, the MRF defined in (1.2) is a member of the exponential family. To see this, let $\boldsymbol{s}(\boldsymbol{y})$ be a vector containing an indicator for every configuration of each clique in G:

$$\boldsymbol{s}(\boldsymbol{y}) = \{I[\boldsymbol{Y}_C = \boldsymbol{y}_C] \mid \forall C \in \mathcal{C}(G), \boldsymbol{y}_C\}. \tag{1.10}$$

Also, let $\boldsymbol{\theta}(\boldsymbol{Y}_C = \boldsymbol{y}_C)$ denote the component of the parameter vector $\boldsymbol{\theta}$ corresponding to the indicator $I[\boldsymbol{Y}_C = \boldsymbol{y}_C]$. The MRF in (1.2) can then be expressed as a member of the exponential family by setting $\boldsymbol{\theta}(\boldsymbol{Y}_C = \boldsymbol{y}_C) = \log \psi_C(\boldsymbol{Y}_C = \boldsymbol{y}_C)$ for each configuration of every clique.

$p(\mathbf{y}) = \frac{1}{Z}\psi_{12}(y_1, y_2)\psi_{13}(y_1, y_3)\cdots\psi_3(y_3)$



$$\mathbf{s}(\mathbf{y}) = \begin{bmatrix} I[y_1 = 0] \\ I[y_1 = 1] \\ I[y_2 = 0] \\ I[y_2 = 1] \\ I[y_3 = 0] \\ I[y_3 = 1] \\ I[y_1 = 0, y_2 = 0] \\ I[y_1 = 0, y_2 = 1] \\ I[y_1 = 1, y_2 = 0] \\ I[y_1 = 1, y_2 = 1] \\ I[y_1 = 0, y_3 = 0] \\ \vdots \\ I[y_2 = 1, y_3 = 1] \end{bmatrix} \quad \theta(\mathbf{y}) = \begin{bmatrix} \log\psi_1(y_1 = 0) \\ \log\psi_1(y_1 = 1) \\ \log\psi_2(y_2 = 0) \\ \log\psi_2(y_2 = 1) \\ \log\psi_3(y_3 = 0) \\ \log\psi_3(y_3 = 1) \\ \log\psi_{12}(y_1 = 0, y_2 = 0) \\ \log\psi_{12}(y_1 = 0, y_2 = 1) \\ \log\psi_{12}(y_1 = 1, y_2 = 0) \\ \log\psi_{12}(y_1 = 1, y_2 = 1) \\ \log\psi_{13}(y_1 = 0, y_3 = 0) \\ \vdots \\ \log\psi_{23}(y_2 = 1, y_3 = 1) \end{bmatrix}$$

Figure 1.3: Expressing a pairwise MRF in exponential family form.

Figure 1.3 illustrates this transformation for a pairwise MRF defined on a set of 3 binary variables. The vector $\boldsymbol{s}(\boldsymbol{y})$ in this example is of length $d = 18$; however, a binary MRF on 3 variables has a total of only $2^3 = 8$ joint configurations, suggesting that there is some redundancy in this representation of an MRF. In fact, this representation, which uses indicators for each configuration of every clique, is referred to as the *standard overcomplete representation* of an MRF and it is an example of an exponential family in a *non-minimal representation*. In contrast, an exponential family model is in a *minimal representation* if the sufficient statistics, $\boldsymbol{s}(\boldsymbol{y})$, are linearly independent, meaning that one cannot find a non-zero vector $\boldsymbol{\alpha} \in \mathbb{R}^d$ such that $\boldsymbol{\alpha} \cdot \boldsymbol{s}(\boldsymbol{y})$ is constant for all settings of $\boldsymbol{y}$.

To make the distinction between *non-minimal* and *minimal* representations concrete, consider the following pairwise MRFs, where each variable $i \in V$ is binary $y_i \in \{0, 1\}$:

$$p(\boldsymbol{y}; \boldsymbol{\theta}) = \exp\left(\sum_{i \in V} \theta_i I[y_i = 1] + \sum_{(i,j) \in E} \theta_{ij} I[y_i = y_j]\right), \tag{1.11}$$

$$p(\boldsymbol{y}; \tilde{\boldsymbol{\theta}}) = \exp\left(\sum_{i \in V} \tilde{\theta}_i I[y_i = 1] + \sum_{(i,j) \in E} \tilde{\theta}_{ij} I[y_i = y_j = 1]\right). \tag{1.12}$$

The model in (1.11) is non-minimal. The reason why is demonstrated in Figure 1.4. Let $\boldsymbol{s}_{ij}(y_i, y_j) = (I[y_i = 1], I[y_j = 1], I[y_i = y_j])$ be the sufficient statistics on some edge $(i, j) \in E$ in our model. The table in Figure 1.4 shows the setting of $\boldsymbol{s}_{ij}(y_i, y_j)$ for all four configurations of y_i and y_j. Note that by setting $\boldsymbol{\alpha} = \mathbf{1}$ we have that $\boldsymbol{\alpha} \cdot \boldsymbol{s}_{ij}(y_i, y_j) = 1$ for all four configurations. This is true on any edge $(i, j) \in E$, meaning that the MRF representation in (1.11) is non-minimal. In contrast, the MRF in (1.12) is in a minimal exponential family representation, as the vector of sufficient statistics, $\boldsymbol{s}(\boldsymbol{y})$, is linearly independent for all $\boldsymbol{y}$.

		$\mathbf{s}_{ij}(y_i, y_j)$		
y_i	y_j	$I[y_i = 1]$	$I[y_j = 1]$	$I[y_i = y_j]$
0	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**0**
1	**0**	**1**	**0**	**0**
1	**1**	**1**	**1**	**1**

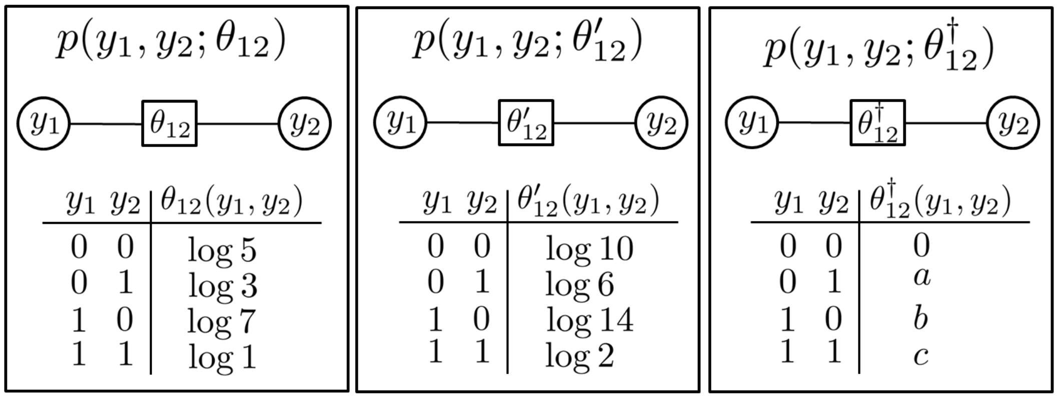
Figure 1.4: Non-minimal MRF.

There is an important practical distinction between non-minimal and minimal representations of an exponential family distribution when it comes to parameter estimation – namely, statistical identifiability. A model is identifiable if it is possible to learn the model's true parameter setting given an infinite number of observations: meaning that if we generate data by sampling from an identifiable model $p(\boldsymbol{y}; \boldsymbol{\theta}^{\star})$, then it is possible to recover the true parameter setting, $\boldsymbol{\theta}^{\star}$, given enough samples. More formally,

Definition 1.1. *A statistical model is **identifiable** if the mapping from $\boldsymbol{\theta} \mapsto p(\boldsymbol{y}; \boldsymbol{\theta})$ is one-to-one, so that if two distributions are the same, $p(\boldsymbol{y}; \boldsymbol{\theta}) = p(\boldsymbol{y}; \boldsymbol{\theta}')$, then $\boldsymbol{\theta} = \boldsymbol{\theta}'$.*

In a minimal representation, there is a unique distribution associated with each setting of $\boldsymbol{\theta}$, meaning that the model is identifiable. In a nonminimal representation, however, many distinct parameter settings may yield the same distribution.

Figure 1.5 illustrates two distinct parameter settings, $\boldsymbol{\theta} \neq \boldsymbol{\theta}'$, of an MRF that yield the same probability distribution – i.e. $p(y_1, y_2; \boldsymbol{\theta}) = p(y_1, y_2; \boldsymbol{\theta}')$ for all configurations of $\boldsymbol{y} = (y_1, y_2)$. These models are thus not identifiable. The parameter setting $\boldsymbol{\theta}^{\dagger}$, however, is statistically identifiable for any distinct and non-zero setting of a, b, and c.


$p(y_1, y_2; \theta_{12})$
y_1
θ_{12}
y_2
$p(y_1, y_2; \theta'_{12})$
y_1
θ'_{12}
y_2
$p(y_1, y_2; \theta^{\dagger}_{12})$
y_1
$\theta^{\dagger}_{12}$
y_2

y_1	y_2	$\theta_{12}(y_1, y_2)$
0	0	$\log 5$
0	1	$\log 3$
1	0	$\log 7$
1	1	$\log 1$

y_1	y_2	$\theta'_{12}(y_1, y_2)$
0	0	$\log 10$
0	1	$\log 6$
1	0	$\log 14$
1	1	$\log 2$

y_1	y_2	$\theta^{\dagger}_{12}(y_1, y_2)$
0	0	0
0	1	a
1	0	b
1	1	c

Figure 1.5: Illustration of Statistical identifiability.

Mean Parameterization

Up to this point we have discussed the exponential family as a distribution parametrized by a vector $\boldsymbol{\theta} \in \mathbb{R}^d$ of canonical parameters. The exponential family has an alternative parametrization, however, that is particularly useful in variational analysis. The *mean parameters* are found by computing the mean of the vector of sufficient statistics,

$$E_{\boldsymbol{\theta}}\left[\boldsymbol{s}(\boldsymbol{y})\right] = \sum_{\boldsymbol{y}} p(\boldsymbol{y};\boldsymbol{\theta})\boldsymbol{s}(\boldsymbol{y}) \stackrel{\text{def}}{=} \boldsymbol{\mu}(\boldsymbol{\theta}) = (\mu_1(\boldsymbol{\theta}), ..., \mu_d(\boldsymbol{\theta})). \tag{1.13}$$

We will use $\boldsymbol{\mu}(\boldsymbol{\theta})$ to denote the d-dimensional mean vector of an exponential family model. This notation helps to signify that each setting of the canonical parameters, $\boldsymbol{\theta}$, maps forward to a unique mean parameter vector, $\boldsymbol{\mu}(\boldsymbol{\theta})$. The reverse mapping, $\boldsymbol{\theta}(\boldsymbol{\mu})$, from mean parameters to canonical parameters is not unique in general (see e.g. Figure 1.5 for a counter-example). However, if the model $p(\boldsymbol{y};\boldsymbol{\theta})$ is in a minimal representation, then the mapping between mean parameters $\boldsymbol{\mu}$ and canonical parameters $\boldsymbol{\theta}$ is one-to-one.

The mean parameters have a particularly nice interpretation in the exponential family as being equal to the first derivative of the log-partition function

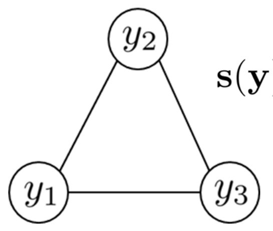
$$\frac{d}{d\boldsymbol{\theta}} \log Z(\boldsymbol{\theta}) = \frac{\sum_{\boldsymbol{y}} \exp\left(\boldsymbol{\theta}\cdot s(\boldsymbol{y})\right) s(\boldsymbol{y})}{\sum_{\boldsymbol{y}} \exp\left(\boldsymbol{\theta}\cdot s(\boldsymbol{y})\right)} = \sum_{\boldsymbol{y}} p(\boldsymbol{y};\boldsymbol{\theta})\boldsymbol{s}(\boldsymbol{y}) = \boldsymbol{\mu}(\boldsymbol{\theta}). \tag{1.14}$$

In the case of an MRF in standard overcomplete representation, the vector of mean parameters $\boldsymbol{\mu}(\boldsymbol{\theta})$ is exactly the vector of marginal probabilities under the model $p(\boldsymbol{y};\boldsymbol{\theta})$,

$$E_{\boldsymbol{\theta}}\left[I[\boldsymbol{Y}_c = \boldsymbol{y}_c]\right] = \sum_{\boldsymbol{y}'} p(\boldsymbol{y}';\boldsymbol{\theta}) I[\boldsymbol{Y}_c = \boldsymbol{y}_c] = p(\boldsymbol{Y}_c = \boldsymbol{y}_c;\boldsymbol{\theta}) = \mu(\boldsymbol{y}_c;\boldsymbol{\theta}), \tag{1.15}$$

where once again we index the components of the vector $\boldsymbol{\mu}(\boldsymbol{\theta})$ by each configuration of every clique in G. Figure 1.6 illustrates the mean parameterization of the pairwise MRF on three variables introduced in Figure 1.3.

Finding the mapping from canonical parameters to the mean parameters of an MRF, $\boldsymbol{\mu}(\boldsymbol{\theta})$, or equivalently computing the gradient of the log-partition function, $\frac{d}{d\boldsymbol{\theta}} \log Z(\boldsymbol{\theta})$, is a task



$$\mathbf{s}(\mathbf{y}) = \begin{bmatrix} I[y_1 = 0] \\ I[y_1 = 1] \\ I[y_2 = 0] \\ I[y_2 = 1] \\ I[y_3 = 0] \\ I[y_3 = 1] \\ I[y_1 = 0, y_2 = 0] \\ I[y_1 = 0, y_2 = 1] \\ I[y_1 = 1, y_2 = 0] \\ I[y_1 = 1, y_2 = 1] \\ I[y_1 = 0, y_3 = 0] \\ \vdots \\ I[y_2 = 1, y_3 = 1] \end{bmatrix} \quad \mu(\theta) = \begin{bmatrix} E_\theta\left[I[y_1 = 0]\right] = p(y_1 = 0; \theta) \\ E_\theta\left[I[y_1 = 1]\right] = p(y_1 = 1; \theta) \\ E_\theta\left[I[y_2 = 0]\right] = p(y_2 = 0; \theta) \\ E_\theta\left[I[y_2 = 1]\right] = p(y_2 = 1; \theta) \\ E_\theta\left[I[y_3 = 0]\right] = p(y_3 = 0; \theta) \\ E_\theta\left[I[y_3 = 1]\right] = p(y_3 = 1; \theta) \\ E_\theta\left[I[y_1 = 0, y_2 = 0]\right] = p(y_1 = 0, y_2 = 0; \theta) \\ E_\theta\left[I[y_1 = 0, y_2 = 1]\right] = p(y_1 = 0, y_2 = 1; \theta) \\ E_\theta\left[I[y_1 = 1, y_2 = 0]\right] = p(y_1 = 1, y_2 = 0; \theta) \\ E_\theta\left[I[y_1 = 1, y_2 = 1]\right] = p(y_1 = 1, y_2 = 1; \theta) \\ E_\theta\left[I[y_1 = 0, y_3 = 0]\right] = p(y_1 = 0, y_3 = 0; \theta) \\ \vdots \\ E_\theta\left[I[y_2 = 1, y_3 = 1]\right] = p(y_2 = 1, y_3 = 1; \theta) \end{bmatrix}$$

Figure 1.6: Illustration of the mean parameterization of the pairwise MRF in Figure 1.3.

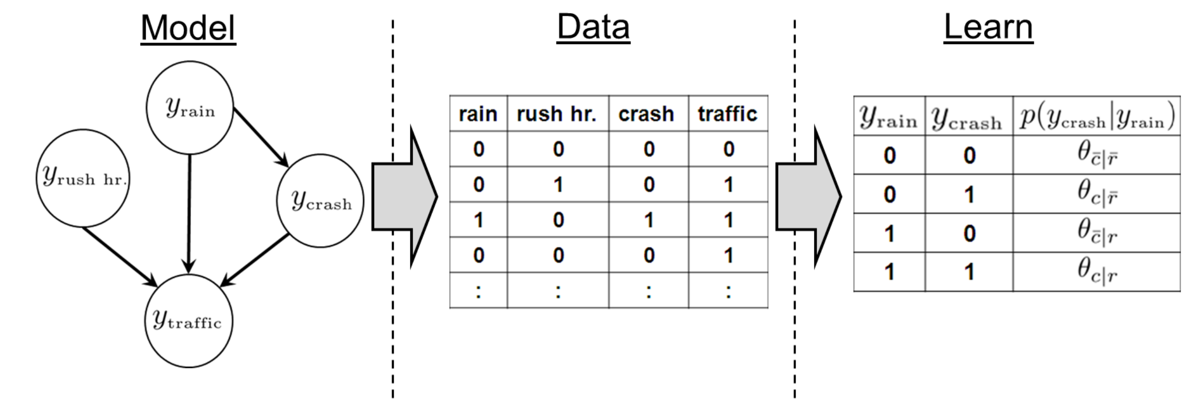
that requires computing marginals under the model $p(\boldsymbol{y}; \boldsymbol{\theta})$. The variational approximations we discuss in Section 1.4.2 leverage this interrelation between the marginals and log-partition function.

1.3 Inference and Learning Tasks

The previous section presented different representations of a joint probability distribution using the language of graphical models. In particular, we saw that the joint distribution of a graphical model factors according to some underlying graph structure, where the form of the model's factors depended on whether the model was a Bayesian network, MRF or CRF.

After the form of a graphical model has been fully specified – meaning that the model's structure and parameterization have been determined – a few key tasks remain. The first task is that of learning the parameters of the model from empirical data. The second is performing inference in the learned model to, for example, find the most probable configuration under the model or to compute the probability of certain events. We introduce each of these tasks using the traffic flow Bayesian network from the previous section.

Consider the traffic flow model shown again in Figure 1.7. The Bayesian network's structure fully specifies the factorization of the joint distribution $p(y_{\text{rain}}, y_{\text{rush hr.}}, y_{\text{crash}}, y_{\text{traffic}})$. And since all of the variables are binary, the parameterization of each conditional probability table is known as well. For example, the table $p(y_{\text{crash}}|y_{\text{rain}})$ has a total of 4 entries with parameters $(\theta_{\bar{c}|\bar{r}}, \theta_{c|\bar{r}}, \theta_{\bar{c}|r}, \theta_{c|r})$ as shown in the right-hand pane of Figure 1.7.



rain	rush hr.	crash	traffic
0	0	0	0
0	1	0	1
1	0	1	1
0	0	0	1
:	:	:	:

y_{rain}	y_{crash}	$p(y_{\text{crash}}\|y_{\text{rain}})$
0	0	$\theta_{\bar{c}\|\bar{r}}$
0	1	$\theta_{c\|\bar{r}}$
1	0	$\theta_{\bar{c}\|r}$
1	1	$\theta_{c\|r}$

Figure 1.7: Illustration of Learning and Inference in traffic flow model.

The task of learning involves estimating the setting of the real-valued parameters $\boldsymbol{\theta}$ from an empirical data set. The middle pane of Figure 1.7 shows some sample data that could be used to train our traffic model, where in the first data point, for example, it wasn't raining or rush hour and no accidents or traffic were observed. We will assume throughout this thesis that our data is fully observed and sampled independently from some true and unknown probability distribution. We focus on learning using the maximum likelihood principle, where our goal is find a setting of the model parameters $\boldsymbol{\theta}$ that makes our learned model $p(y_{\text{rain}}, y_{\text{rush hr.}}, y_{\text{crash}}, y_{\text{traffic}}; \boldsymbol{\theta})$ as close as possible to the true and unknown distribution. The maximum likelihood principle will be formally introduced in Chapter 4.

After learning, we want to use our learned model to answer queries given new data. One task of particular importance is that of computing the *maximum a posteriori (MAP)* configuration in our model. In particular, the goal of the *MAP inference task* is to find the assignment $\boldsymbol{y}$

of maximum total probability

$$\text{MAP}: \quad \operatorname*{arg\,max}_{\boldsymbol{y} \in \mathcal{Y}^m} p(\boldsymbol{y}; \boldsymbol{\theta}). \tag{1.16}$$

Often times, we are interested in finding the MAP configuration given a new observation, or evidence. For example, we might be interested in knowing the most likely configuration of our system when we know that it is both raining ($y_{\text{rain}} = 1$) and rush hour ($y_{\text{rush hr.}} = 1$), i.e.

$$\operatorname*{arg\,max}_{y_{\text{crash}}, y_{\text{traffic}}} \; p(y_{\text{crash}}, y_{\text{traffic}} | y_{\text{rain}} = 1, y_{\text{rush hr.}} = 1; \boldsymbol{\theta}).$$

Another important task is computing the marginal probability of events under our model. In particular, the goal of the *marginal inference task* is to find the marginal distributions of each factor in our model

$$\text{MAR}: \quad p(\boldsymbol{Y}_C = \boldsymbol{y}_C; \boldsymbol{\theta}) \; \forall C \in \mathcal{C}(G), \boldsymbol{y}_C. \tag{1.17}$$

For example, given that it is raining and rush hour, we might want to know the probability that there will be traffic on our drive home – i.e., $p(y_{\text{traffic}} = 1 | y_{\text{rain}} = 1, y_{\text{rush hr.}} = 1; \boldsymbol{\theta})$.

In principle, both the MAP inference task and marginal inference task can be solved by first generating the joint probability distribution under the current evidence and then finding the maximizing assignment by exhaustive search, in the case of the MAP task, or by summing over all variables not in the query factor, in the case of the marginal task. This brute force approach results in an exponential explosion in the space and time needed to compute even the simplest of queries in our model. In the next section, we discuss several inference algorithms that exploit the underlying graph structure to more efficiently solve these tasks.

1.4 Inference Algorithms for Graphical Models

We now discuss a few key algorithms for performing inference in graphical models. We focus in this section on solving the marginal inference task, only briefly discussing how the algorithms can be modified for MAP inference. Our description of the algorithms in this section will be based on the factor graph representation of a joint probability distribution introduced in (1.5) as this representation removes potential confusion by making the factorization of a distribution explicit.

This section is divided into two parts. In the first part, we discuss inference from a purely algorithmic, message-passing point of view. Then in the second part, we discuss the algorithms from a variational perspective. In both parts, we begin by describing exact inference and then move to approximate methods.

1.4.1 Inference: A Message-Passing Perspective

Consider computing the marginal probability of variable y_2 for the distribution shown in Figure 1.8. The brute force approach to this problem is to first generate the joint probability distribution, $p(y_1, y_2, y_3)$, by taking the point-wise product, or *combining*, all of the model's factors and then summing out, or *eliminating*, variables y_1 and y_3

$$p(y_2) \propto \sum_{y_1} \sum_{y_3} p(y_1, y_2, y_3) = \sum_{y_1} \sum_{y_3} \psi_{12}(y_1, y_2)\psi_{23}(y_2, y_3)\psi_1(y_1)\psi_2(y_2)\psi_3(y_3). \quad (1.18)$$

If all of the variables are K-ary, then the joint distribution $p(y_1, y_2, y_3)$ is represented as a table with K^3 entries and computing the single variable marginal, $p(y_2)$, has complexity $\mathcal{O}(K^3)$. For a chain of m variables, this brute force strategy has complexity $\mathcal{O}(K^m)$ and becomes infeasible for even modest m.

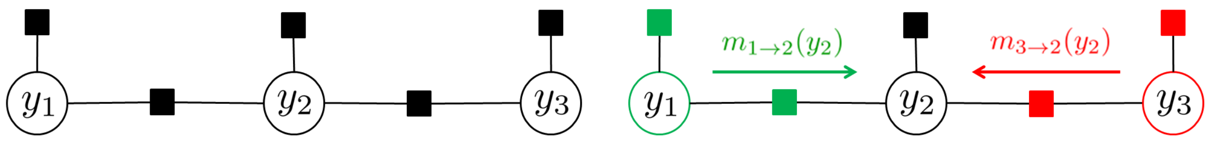


Figure 1.8: Illustration of inference on a chain.

We can reduce this complexity, however, by dividing the summation over the joint distribution into summations over smaller sub-problems. By grouping the factors and migrating the summation operators we arrive at the following expression for the marginal of y_2,

$$p(y_2) \propto \psi_2(y_2) \underbrace{\sum_{y_1} \psi_{12}(y_1, y_2)\psi_1(y_1)}_{m_{1\to 2}(y_2)} \underbrace{\sum_{y_3} \psi_{23}(y_2, y_3)\psi_3(y_3)}_{m_{3\to 2}(y_2)} . \tag{1.19}$$

This reorganization of factors and summation operators reduces the complexity of computing $p(y_2)$ to $\mathcal{O}(K^2)$, as we now only need to sum over tables containing pairs of variables.

It is convenient to think of the computations performed in each sub-problem as computing a message that is passed to a neighboring node in the factor graph. For example, the sub-problem shown in red takes the product of factors $\psi_3(y_3)$ and $\psi_{23}(y_2, y_3)$ and then eliminates variable y_3. The result is an intermediate table of K-values, one for each state of variable y_2, that we treat as a message passed from variable 3 to 2, denoted as $m_{3\to 2}(y_2)$. Similarly, eliminating y_1 from the product of factors $\psi_1(y_1)$ and $\psi_{12}(y_1, y_2)$ generates a message $m_{1\to 2}(y_2)$. By normalizing the product of $\psi_2(y_2)$, $m_{1\to 2}(y_2)$ and $m_{3\to 2}(y_2)$ we get the desired marginal probability: $p(y_2) \propto \psi_2(y_2)m_{1\to 2}(y_2)m_{3\to 2}(y_2)$.

Sum-Product Belief Propagation

While reducing the complexity from $\mathcal{O}(K^3)$ to $\mathcal{O}(K^2)$ is insignificant in this toy problem, the idea of rearranging the factors and summation operators generalizes so that one can reduce the complexity from $\mathcal{O}(K^m)$ to $\mathcal{O}(K^2)$ in arbitrary tree-structured factor graphs of

m variables. The core idea for achieving this efficiency in general tree-structured graphs is to choose a root variable and then pass messages from the leaves of the tree towards the root variable. An algorithm called *Sum-Product Belief Propagation (BP)* operationalizes this core idea[72, 56].

Given a factor graph, $G = (V, F, E)$, the Sum-Product BP algorithm works by passing two kind of messages: messages from variables to factors,

$$\text{Variable-to-Factor :} \quad m_{i\to\alpha}(y_i) = \prod_{\alpha'\in\delta(i)\setminus\alpha} m_{\alpha'\to i}(y_i), \tag{1.20}$$

and messages from factors to variables,

$$\text{Factor-to-Variable :} \quad m_{\alpha\to i}(y_i) = \sum_{\boldsymbol{y}_\alpha\setminus y_i} \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{j\in\delta(\alpha)\setminus i} m_{j\to\alpha}(y_j), \tag{1.21}$$

where $\delta(i) = \{\alpha \in F : (i, \alpha) \in E\}$ is the set of factors neighboring variable i in the factor graph, $\delta(\alpha) = \{i \in V : (i, \alpha) \in E\}$ is the set of variables neighboring factor α, and $\sum_{\boldsymbol{y}_\alpha\setminus y_i}$ is a summation over all variables $\boldsymbol{y}_\alpha$ except y_i – i.e. $\boldsymbol{y}_\alpha \setminus y_i = \{y_j : j \in \delta(\alpha),\ j \neq i\}$.

Note that the message updates in (1.20) and (1.21) are dependent on previously computed messages. In a tree-structured graph, we can order our message computations so that all dependencies are resolved prior to updating. This is accomplished by choosing a root variable and passing messages forward from the leaves towards the root as described before. The marginals of all variables and factors in the model can be computed by subsequently passing messages from the root back towards the leaves. After passing messages forward and backwards, the univariate marginals are computed as

$$p(y_i) \propto \prod_{\alpha\in\delta(i)} m_{\alpha\to i}(y_i) \tag{1.22}$$

and the factor marginals are computed as

$$p(\boldsymbol{y}_\alpha) \propto \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{i \in \delta(\alpha)} m_{i \to \alpha}(y_i). \tag{1.23}$$

In addition, the log-partition function, $\log Z$, can be computed by summing all of the variable to factor messages into the root node, y_{root},

$$\log Z = \log \sum_{y_{\text{root}}} \prod_{\alpha \in \delta(\text{root})} m_{\alpha \to \text{root}}(y_{\text{root}}) \tag{1.24}$$

Max-Product Belief Propagation

An algorithm called *Max-Product Belief Propagation (BP)* can be used to compute a MAP assignment in a tree-structured graph using the same divide-and-conquer strategy as the Sum-Product algorithm. However, rather than eliminating variables by summation, it eliminates by maximization. In particular, the factor to variable messages in Max-Product BP are updated as

$$\text{Factor-to-Variable}: \quad m_{\alpha \to i}(y_i) = \max_{\boldsymbol{y}_\alpha \setminus y_i} \left[\psi_\alpha(\boldsymbol{y}_\alpha) \prod_{j \in \delta(\alpha) \setminus i} m_{j \to \alpha}(y_j) \right], \tag{1.25}$$

while the variable to factor messages are as in (1.20). After passing messages forward to the root and then back towards the leaves, we can compute the *max-marginals* of each variable using the formula in (1.22). We use the term max-marginals, to distinguish between the actual single variable marginals computed by the sum-product algorithm. The MAP assignment, $\boldsymbol{y}^\star = (y_1^\star, ..., y_m^\star) = \arg\max p(\boldsymbol{y})$, can then be determined by decoding each variable's assignment along the forward message passing order as follows. First, determine the assignment $y_{\text{root}}^\star$ that maximizes the max-marginal at the root node. Then find the maximizing assignment $y_i^\star$ to each variable y_i, dependent upon the assignment to all previously assigned variables.

Loopy Belief Propagation

The BP algorithm exactly solves the MAP inference and marginal inference tasks in tree-structured factor graphs. When the factor graph is not tree-structured, then BP is no longer exact. The main difficulty is that in a loopy graph we can no longer order our message computations so that all dependencies are resolved prior to updating each message. However, if we initialize all of the messages in our system to some fixed value, e.g. $m_{i\to\alpha}(y_i) = m_{\alpha\to i}(y_i) = 1$, then the message updates in (1.20) and (1.21) are still valid. Iteratively applying these message updates on a factor graph yields an approximate inference method known as *Loopy Belief Propagation (BP).*

Loopy BP is an iterative method that is not guaranteed to converge. And, when the algorithm does converge, the resulting beliefs may poorly approximate the true marginals, in the case of the marginal inference task, or give a poor approximation to the MAP assignment, in the case of the MAP inference task. Despite the lack of theoretical guarantees, the algorithm performs quite well in practice [68, 63]. Much progress has been made in characterizing the fixed points of the iterative message-passing algorithm [102, 88, 42], improving and understanding its convergence properties [88, 26, 46, 66], and designing convergent alternatives [96, 103, 33, 32, 40].

The Bucket Elimination Algorithm

The BP algorithm performs exact marginal inference in tree-structured factor graphs, but is an approximate method in graphs containing cycles. A natural idea for restoring the exactness of BP is to somehow transform a cyclic factor graph into a new tree-structured graph and pass BP-like messages on the new graph structure. This is the core idea behind exact inference methods like Bucket Elimination[19] and the Junction Tree algorithm[58]: they create a tree-structured graph by grouping factors and variables into clusters and pass messages between these clusters. We focus on the Bucket Elimination algorithm in this section, but the junction tree algorithm operates in a similar fashion. A review of different message passing architectures can be found in [18, 53].

Bucket Elimination (BE) is an algorithm that groups factors into clusters that are referred to as *buckets*. It operates along a particular variable *elimination order*, which specifies the order in which variables are summed out. A bucket is associated with each variable to be eliminated and each bucket is assigned a collection of functions. The assigned functions are either: 1) the factors defining our model; or 2) messages generated by the algorithm. Each bucket is processed by combining it's set of assigned functions and then eliminating the bucket variable. The resulting message is passed to some bucket that has yet to be processed and buckets are processed sequentially along the elimination order.

The BE algorithm is illustrated for the pairwise model in Figure 1.9, where it uses the elimination order $\pi = (2, 3, 4, 5, 1)$, eliminating variable y_2 first, y_3 second, ... In doing so, the algorithm performs the following sequence of computations:

$$\begin{aligned}
p(y_1) &\propto \sum_{y_2,y_3,y_4,y_5} \psi_{12}(y_1,y_2)\psi_{14}(y_1,y_4)\psi_{15}(y_1,y_5)\psi_{23}(y_2,y_3)\psi_{34}(y_3,y_4)\psi_{45}(y_4,y_5) \qquad (1.26)\\
&\propto \sum_{y_3,y_4,y_5} \psi_{14}(y_1,y_4)\psi_{15}(y_1,y_5)\psi_{34}(y_3,y_4)\psi_{45}(y_4,y_5) \underbrace{\sum_{y_2} \psi_{12}(y_1,y_2)\psi_{23}(y_2,y_3)}_{m_{2\to3}(y_1,y_3)}\\
&\propto \sum_{y_4,y_5} \psi_{14}(y_1,y_4)\psi_{15}(y_1,y_5)\psi_{45}(y_4,y_5) \underbrace{\sum_{y_3} \psi_{34}(y_3,y_4) m_{2\to3}(y_1,y_3)}_{m_{3\to4}(y_1,y_4)}\\
&\propto \sum_{y_5} \psi_{15}(y_1,y_5) \underbrace{\sum_{y_4} \psi_{14}(y_1,y_4)\psi_{45}(y_4,y_5) m_{3\to4}(y_1,y_4)}_{m_{4\to5}(y_1,y_5)}\\
&\propto \sum_{y_5} \psi_{15}(y_1,y_5) m_{4\to5}(y_1,y_5)
\end{aligned}$$

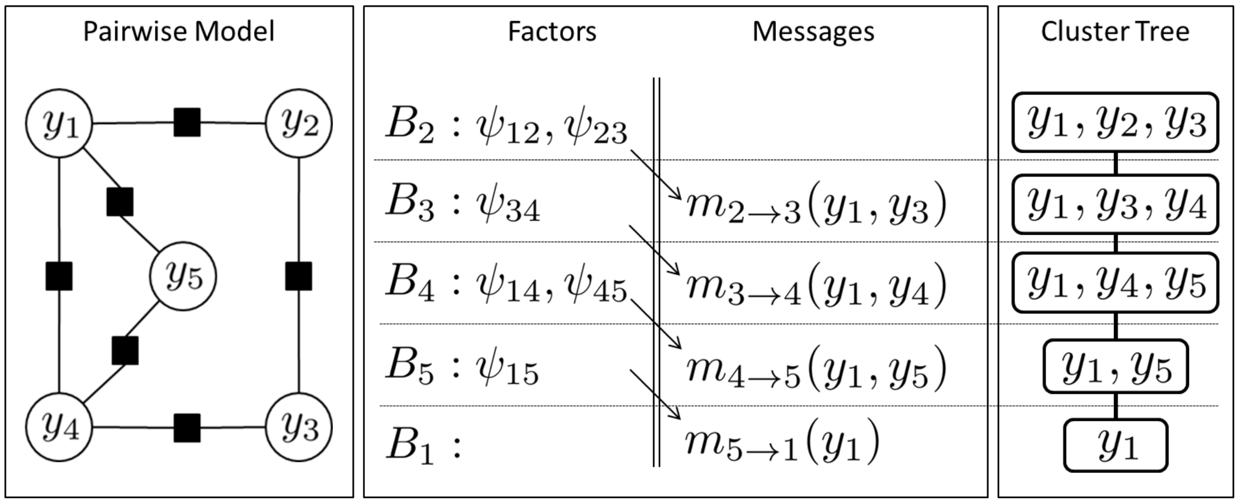


Figure 1.9: Illustration of the Bucket Elimination Algorithm.

Note that each summation produces a message that is passed to the next bucket in the ordering. For example, by eliminating variable y_2 in the first bucket, B_2, we produce a message defined over variables y_1 and y_3. This message is passed to the second bucket, B_3, associated with variable y_3. We denote this message as $m_{2\to3}(y_1, y_3)$ to indicate that it was sent from bucket B_2 to bucket B_3. We will refer to bucket B_3 as being the parent of bucket B_2 and bucket B_2 as being a child of bucket B_3. We note that a bucket may have multiple children, but will only have one parent. The middle column of Figure 1.9 shows how the original model factors and messages generated by the algorithm are assigned to each bucket, where we have used ψ_{ij} as shorthand for $\psi_{ij}(y_i, y_j)$.

Execution of the BE algorithm induces a tree structure, $CT = (V, E)$, known as a cluster tree or bucket tree. In particular, each bucket, B_i, is associated with a node $i \in V$ in the cluster tree and an edge $e = (i, j) \in E$ is drawn between nodes i and j if the message produced by processing bucket B_i is used in the computation of bucket B_j. Each node i in the cluster tree is associated with a set of variables, $\boldsymbol{y}_i \subset \boldsymbol{y}$, that appear in bucket B_i. This subset of of variables is referred to as the cluster's *scope*. Finally, we associate a set of *separator* variables with each edge $(i, j) \in E$ that is equal to the intersection of the scope of cluster i and the scope of cluster j, $\boldsymbol{y}_{ij} = \boldsymbol{y}_i \cap \boldsymbol{y}_j$. An example of a cluster tree is shown in the right pane of Figure 1.9, where the scope of each cluster is shown inside of each node. For example, the first cluster – associated with bucket B_2 – has scope $\{y_1, y_2, y_3\}$. Since each bucket is associated with a cluster, we will use the two words interchangeably.

We can compute marginals over a set of variables in the cluster tree by passing messages in a manner analogous to the Sum-Product BP algorithm. A message from cluster node i to cluster j is computed as

$$\text{CT-Update}: \quad m_{i\to j}(\boldsymbol{y}_{ij}) = \sum_{\boldsymbol{y}_i \setminus \boldsymbol{y}_j} \psi_i(\boldsymbol{y}_i) \prod_{k\in\delta(i)\setminus j} m_{k\to i}(\boldsymbol{y}_{ik}), \tag{1.27}$$

where $\psi_i(\boldsymbol{y}_i)$ is the combination of the factors assigned to cluster (bucket) i and $m_{k\to i}$ is a message from some cluster k to cluster i. Note that the updates in (1.27) are dependent upon previously computed messages, much like the Sum-Product BP updates. These dependencies can be resolved, however, by ordering our message computations as follows: messages are first passed forward, along the elimination order and then backwards, along the reverse elimination order. The log-partition function, $\log Z$, can be computed by summing over all messages into the final, root cluster.

The complexity of message passing on a cluster tree (and of BE) depends on the size of the largest cluster in the tree, where a cluster's size is simply the number of variables in its scope. For example, the complexity of BE in Figure 1.9 is $\mathcal{O}(K^3)$ since the largest cluster contains 3 variables. The complexity of BE thus depends crucially on the elimination order used. One can verify, for example, that using the order $\pi = (1, 2, 3, 4, 5)$ has complexity $\mathcal{O}(K^4)$ in our sample problem. Finding an elimination order that yields small clusters is thus central to efficient inference when using BE[49]. If we let $C^{\max}(\pi) = \max_i |\text{scope}_i|$ denote the size of the largest cluster produced using elimination order π, then in principle we can locate the elimination order whose largest cluster is as small as possible: $C^\star = \min_\pi C^{\max}(\pi)$. The quantity $C^\star$ is referred to as the *treewidth* of our model[3] and the complexity of performing exact inference in our model is $\mathcal{O}(K^{C^\star})$[9]. Thus, when $C^\star$ is large exact inference is generally infeasible.

Mini-Bucket and Weighted Mini-Bucket Elimination

The treewidth of some problems is large enough to make exact inference infeasible. The main difficulty in such problems is the existence of one or more large clusters on which elimination cannot be performed efficiently. In this section, we consider an approximate inference method that partitions such large clusters into several smaller clusters and performs elimination on each mini-cluster independently.

Mini-Bucket Elimination (MBE) is an approximation algorithm based on BE that partitions the set of functions assigned to each bucket[21]. Each partition is called a *mini-bucket* and, as in standard BE, a mini-bucket is processed by first combining the functions in that mini-bucket and then eliminating the bucket variable. The complexity of MBE is controlled by a parameter known as the *iBound*, which limits the size of each mini-bucket to at most iBound variables. When iBound $= 1$ each function is placed in its own mini-bucket and processed independently; and when iBound $\geq C^{\max}(\pi)$, no partitioning occurs and MBE is equivalent to BE.

Execution of the MBE algorithm with an iBound of 2 is illustrated in Figure 1.10 for the pairwise model considered in the previous section. Notice that the factors ψ_{12} and ψ_{23} are placed into two separate mini-buckets in bucket B_2 as indicated by the brackets. This occurs because their combination would produce a factor over the 3 variables $\{y_1, y_2, y_3\}$, which is greater than the iBound of 2. Each of these mini-buckets is processed separately: processing the first mini-bucket containing factor ψ_{23} produces a message $m_{2\to3}(y_3)$ sent to bucket B_3;

[3]Technically, the treewidth is equal to $C^\star - 1$.

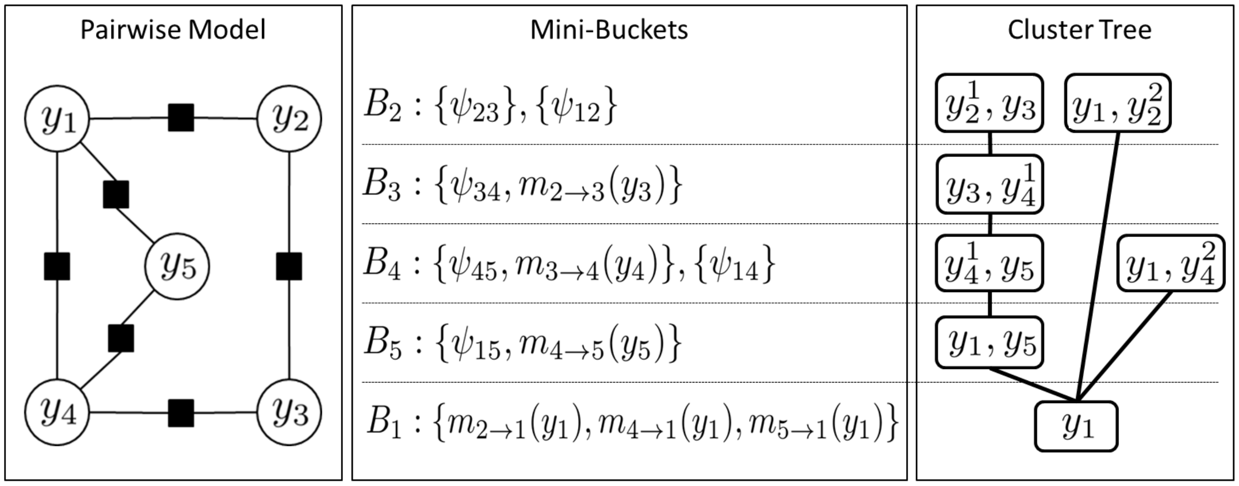

Pairwise Model
y_1
y_2
y_5
y_4
y_3
Mini-Buckets
$B_2 : \{\psi_{23}\}, \{\psi_{12}\}$
$B_3 : \{\psi_{34}, m_{2\to3}(y_3)\}$
$B_4 : \{\psi_{45}, m_{3\to4}(y_4)\}, \{\psi_{14}\}$
$B_5 : \{\psi_{15}, m_{4\to5}(y_5)\}$
$B_1 : \{m_{2\to1}(y_1), m_{4\to1}(y_1), m_{5\to1}(y_1)\}$
Cluster Tree
y_2^1, y_3
y_1, y_2^2
y_3, y_4^1
y_4^1, y_5
y_1, y_4^2
y_1, y_5
y_1

Figure 1.10: Illustration of the Mini-Bucket Elimination Algorithm.

and processing the second mini-bucket containing factor ψ_{12} produces a second message $m_{2\to1}(y_1)$ sent to bucket B_1. Processing continues sequentially along the elimination order as in BE, but we now partition the functions assigned to each bucket if needed.

Execution of the MBE algorithm induces a tree-structure much like the BE algorithm: each mini-bucket corresponds to a cluster associated with a set of variables appearing in the mini-bucket and an edge is drawn between cluster i and cluster j if mini-bucket i's message is used in the computation of mini-bucket j. It will be convenient to think of cluster j as being the parent of cluster i in such cases.

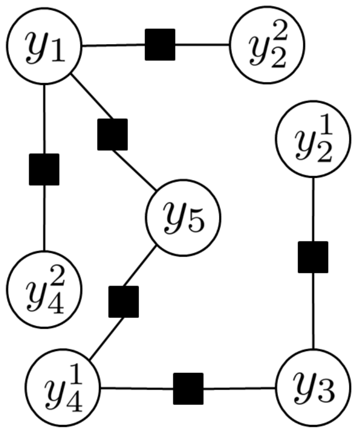


Figure 1.11: Splitting Semantics of MBE.

The cluster tree for the MBE approximation to our sample problem is shown in the right pane of Figure 1.10. As desired, each cluster in the tree has a scope containing at most iBound variables (in this case 2). Notice, however, that the cluster tree contains copies of some of the variables, e.g. y_2^1 and y_2^2 in bucket B_2. This is because eliminating the bucket variable from each mini-bucket separately, effectively splits a variable into one or more replicates [20, 15]. In particular, the execution of MBE illustrated in Figure 1.10 will generate the same messages as executing BE on the *split* pairwise model shown in Figure 1.11.

MBE can be used to provide an upper bound on the log-partition function by summing out the bucket variable from one of the mini-buckets and maximizing the bucket variable in the remaining mini-buckets. For example, summing out y_2^1 for each value of y_3 and separately maximizing y_2^2 for each value of y_1, gives the following inequality in bucket B_2,

$$\sum_{y_2} \psi_{23}\psi_{12} \leq \sum_{y_2^1} \psi_{23} \max_{y_2^2} \psi_{12}. \tag{1.28}$$

An upper bound on $\log Z$ follows by repeating this process in each bucket.

Weighted Mini-Bucket Elimination (WMB) [61] is a generalization of MBE that can be used to iteratively tighten MBE's bound. As its name suggests, the WMB bound is parameterized by a set of weights that can be optimized to find the tightest bound, given a particular elimination order and partitioning of functions into mini-buckets. Before discussing how to optimize the weights, we first introduce the WMB bound and discuss a forward-backward message-passing procedure for approximating $\log Z$ and the marginals.

WMB builds its bound using the weighted summation operator

$$\sum_{y}^{w} f(y) \overset{\text{def}}{=} \left(\sum_{y} f(y)^{1/w} \right)^{w} \tag{1.29}$$

and Hölder's inequality

$$\sum_{y} \prod_{i} f_i(y) \leq \prod_{i} \sum_{y}^{w_i} f_i(y), \tag{1.30}$$

where $\{f_i(y)\}_{i=1}^{m}$ are a collection of positive functions over the discrete variable y and $\{w_i\}_{i=1}^{m}$ is a corresponding set of positive weights such that $\sum_i w_i = 1$. Applying Hölder's inequality to bucket B_2 gives the following inequality,

$$\sum_{y_2} \psi_{23}\psi_{12} \leq \sum_{y_2^1}^{w_1} \psi_{23} \sum_{y_2^2}^{w_2} \psi_{12}. \tag{1.31}$$

We note that the inequality used by the MBE algorithm in (1.28) is a specific case of the WMB inequality in (1.31) – namely, $\lim_{w_2 \to 0^+} \sum_{y_2^2}^{w_2} \psi_{12} = \max_{y_2^2} \psi_{12}$.

We can compute a bound on the log-partition function and marginals over sets of variables in each mini-bucket by passing messages in a manner analogous to the Sum-Product BP and BE algorithms. These updates differ from the BE update rule in (1.27), however, due to the weighted summation operator. A forward message from some cluster (mini-bucket) i to its parent pa_i is computed as

$$\text{WMB-Forward :} \quad m_{i \to pa_i} = \sum_{y_i}^{w_i} \psi_i(\boldsymbol{y}_i) \prod_{j\,:\,i=pa(j)} m_{j \to i} = \left(\sum_{y_i} \left(\psi_i(\boldsymbol{y}_i) \prod_{j\,:\,i=pa(j)} m_{j \to i} \right)^{\frac{1}{w_i}} \right)^{w_i}, \tag{1.32}$$

where w_i is the weight assigned to cluster i. A backward message from cluster i to one of its children j is computed as

$$\text{WMB-Backward :} \quad m_{i \to j} = \left(\sum_{\boldsymbol{y}_i \backslash \boldsymbol{y}_j} \left(\psi_i(\boldsymbol{y}_i) \prod_{k \in \delta(i)} m_{k \to i} \right)^{1/w_i} m_{j \to i}^{-1/w_j} \right)^{w_j}, \tag{1.33}$$

where $m_{k \to i}$ is a message from any neighbor of cluster i, $\delta(i)$, including its parent and w_i and w_j are the weights of cluster i and cluster j, respectively. Note that a forward message depends only on messages from a node's children, while a backward message depends on messages from both a node's parent and children. These dependencies can be resolved by passing messages forward along the elimination order first and then backwards along the reverse elimination order.

The upper bound on $\log Z$ is computed at the root cluster as

$$\log Z \leq \log Z(\boldsymbol{w}) = \log \sum_{\boldsymbol{x}_{\text{root}}}^{w_{\text{root}}} \psi_{\text{root}}(\boldsymbol{x}_{\text{root}}) \prod_{i \in \delta(\text{root})} m_{i \to \text{root}} \tag{1.34}$$

and approximate marginals over each cluster are computed as

$$q(\boldsymbol{x}_i) \propto \left(\psi_i(\boldsymbol{y}_i) \prod_{j \in \delta(i)} m_{j \to i} \right)^{1/w_i} . \tag{1.35}$$

Liu and Ihler discuss at length how to tighten the WMB bound in their paper [61]. The key idea is to recognize that the weighted log-partition function in (1.34) is actually a jointly convex function of both the weights used by the weighted summation operator and the factors (or parameters) defining our model. They exploit this joint convexity to derive efficient message passing updates for both the weights and model parameters, which can be interleaved nicely with the forward-backward message updates in (1.32) and (1.33). We refer the interested reader to their paper for further detail.

Generalized Belief Propagation

Recall that the Loopy BP algorithm was derived by directly applying the Sum Product BP update rules to a loopy graph. While exactness guarantees were lost by applying the updates to a loopy factor graph, we nonetheless gained a powerful and efficient approximate inference method. Generalized Belief Propagation (GBP) is a class of approximate inference methods with an analogous motivation: directly apply the cluster tree message passing updates to a cluster graph, rather than a cluster tree. GBP is thus a generalization of Loopy BP because it passes messages between clusters rather than on a factor graph.

We begin our discussion of GBP by providing a definition of a cluster graph. A *cluster graph* for some model $p(\boldsymbol{y}) \propto \prod_{\alpha \in F} \psi_\alpha(\boldsymbol{y}_\alpha)$ is a graph $CG = (V, E)$ where:

- Each node $i \in V$ is associated with a cluster of variables, $\boldsymbol{y}_i \subseteq \boldsymbol{y}$;
- Each factor $\alpha \in F$ can be *assigned* to a cluster, where by assigned we mean that there exists a cluster $i \in V$ such that $\boldsymbol{y}_\alpha \subseteq \boldsymbol{y}_i$ for each factor $\alpha \in F$;
- Each edge $e = (i, j) \in E$ is associated with a set of variables $\boldsymbol{y}_{ij} \subseteq \boldsymbol{y}_i \cap \boldsymbol{y}_j$ referred to as a *separator*; and
- If a variable y appears in two clusters i and j, then there must be some path connecting cluster i and j such that y is in every cluster and separator on that path. In other words, the set of clusters $\{v \in V \mid y \in \boldsymbol{y}_v\}$ must induce a connected subgraph of CG.

There is a considerable amount of flexibility in choosing both the clusters and edge structure comprising a cluster graph. A factor graph, for example, is a specific type of cluster graph, where a cluster is added for each factor and variable and edges are drawn between variable clusters and factor clusters. A cluster tree is also a special type of tree-structured cluster graph, where the separators are $\boldsymbol{y}_{ij} = \boldsymbol{y}_i \cap \boldsymbol{y}_j$.

Figure 1.12 contains an example of a cluster graph for a pairwise MRF model. A simplified[4] cluster tree for this model is shown in the middle pane and a cluster graph is shown in the right pane. The variables in each cluster are drawn inside the nodes and the variables in each separator are denoted on each edge. Notice that the cluster graph structure contains an undirected cycle. As a result we cannot choose a root node and order our message computations so that all dependencies are resolved prior to updating. We can, however, initialize our messages to some fixed value, e.g. $m_{i \to j}(\boldsymbol{y}_{ij}) = \mathbf{1}$, and then iteratively apply the cluster tree message-passing updates in (1.27).

[4]By simplified we mean that non-maximal clusters have been removed.

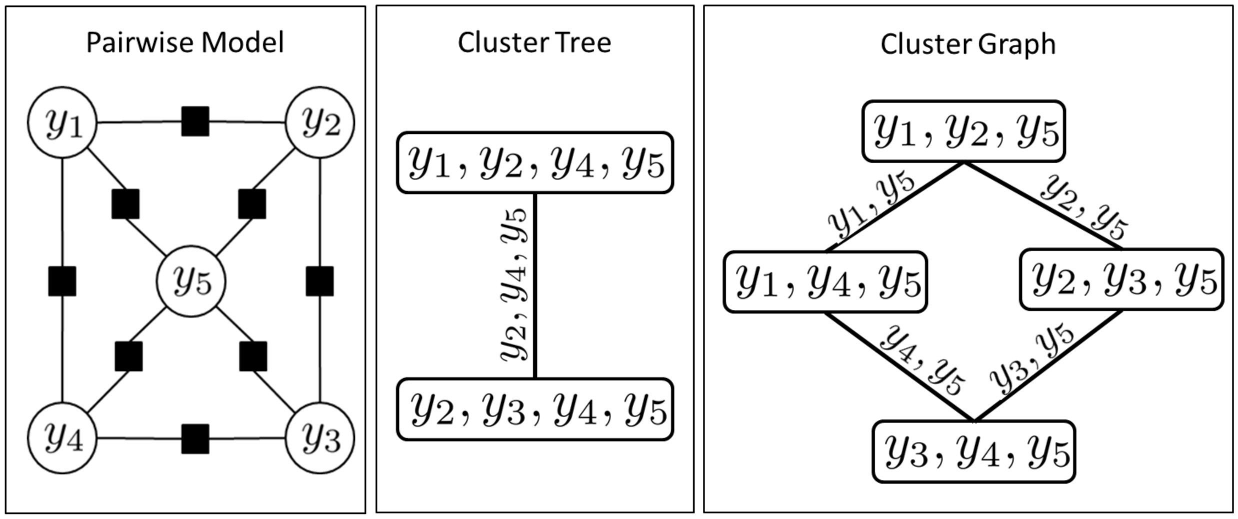


Figure 1.12: Illustration of a Cluster Graph.

GBP is an iterative method with complexity that depends on the size of the largest cluster in the cluster graph. Like Loopy BP, it is not guaranteed to converge, much less converge to a solution providing accurate marginal approximations. However, executing GBP is not as straightforward as executing Loopy BP: while Loopy BP is a well-defined algorithm once given a factor graph, executing GBP requires first specifying the cluster graph structure upon which messages will be passed. As we shall in Chapter 3, the choice of clusters and edges comprising a cluster graph profoundly impact both the convergence of message passing and the accuracy of GBP approximations. Interesting connections between the fixed points of GBP and extrema of an object from statistical physics, known as the Kikuchi free energy, have been established that have helped guide the selection of clusters and separators[102, 94, 95]. We will review these concepts and other related approaches to selecting clusters in Section 1.4.2.

1.4.2 Inference: A Variational Perspective

The previous section introduced Loopy BP and GBP as two approximations that result from a common principle: take message-passing updates that are exact on tree-structured graphs and apply them directly to loopy graphs. In this section, we will see that these algorithms result from applying similar approximations to a common optimization problem. We will also see that the WMB algorithm can be understood as approximating the common optimization problem, albeit in a manner that provides a bound on the log-partition function.

We begin this section by reviewing the relationship between the marginals and log-partition function of an exponential family distribution discussed in Section 1.2.1. Recall from (1.14) that the first derivative of the log-partition function, $\frac{d}{d\boldsymbol{\theta}} \log Z(\boldsymbol{\theta})$, is equal to the mean parameters, $E_{\boldsymbol{\theta}}\left[\boldsymbol{s}(\boldsymbol{y})\right] = \boldsymbol{\mu}(\boldsymbol{\theta})$, of the model $p(\boldsymbol{y};\boldsymbol{\theta}) = \exp\left(\boldsymbol{s}(\boldsymbol{y})\cdot\boldsymbol{\theta} - \log Z(\boldsymbol{\theta})\right)$. We treat $\boldsymbol{\mu}(\boldsymbol{\theta})$ as a mapping from canonical parameters to mean parameters. The reverse mapping from mean parameters to canonical parameters, $\boldsymbol{\theta}(\boldsymbol{\mu})$, is not unique unless $p(\boldsymbol{y};\boldsymbol{\theta})$ is in a minimal representation.

Up to this point, computing the log-partition function has been viewed as an intractable summation task: $\log Z(\boldsymbol{\theta}) = \log \sum_{\boldsymbol{y}} p(\boldsymbol{y};\boldsymbol{\theta})$. The framework of variational inference converts the problem of computing the log-partition function to an optimization problem [90]:

$$\log Z(\boldsymbol{\theta}) = \max_{\boldsymbol{\mu}\in\mathcal{M}} \left[\boldsymbol{\mu}\cdot\boldsymbol{\theta} + H(\boldsymbol{\mu})\right], \tag{1.36}$$

where $\mathcal{M} = \{\boldsymbol{\mu}' \in \mathbb{R}^d \mid \exists \boldsymbol{\theta} \text{ s.t. } \boldsymbol{\mu}' = \boldsymbol{\mu}(\boldsymbol{\theta})\}$ is the marginal polytope, which is the set of mean vectors $\boldsymbol{\mu}'$ that can arise from some joint distribution $p(\boldsymbol{y};\boldsymbol{\theta})$, and

$$H(\boldsymbol{\mu}) = -\sum_{\boldsymbol{y}} p(\boldsymbol{y};\boldsymbol{\theta}(\boldsymbol{\mu})) \log p(\boldsymbol{y};\boldsymbol{\theta}(\boldsymbol{\mu})), \tag{1.37}$$

is the entropy computed using the distribution $p(\boldsymbol{y};\boldsymbol{\theta}(\boldsymbol{\mu}))$ that results from finding parameters $\boldsymbol{\theta}$ that yield the mean vector $\boldsymbol{\mu}$. In addition, from Danskin's theorem the setting of $\boldsymbol{\mu}$ that optimizes (1.36) yields the marginals of our distribution,

$$\frac{d}{d\boldsymbol{\theta}} \log Z(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \left[\boldsymbol{\mu}\cdot\boldsymbol{\theta} + H(\boldsymbol{\mu})\right] \quad \Rightarrow \quad \boldsymbol{\mu}(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \left[\boldsymbol{\mu}\cdot\boldsymbol{\theta} + H(\boldsymbol{\mu})\right]. \tag{1.38}$$

Thus, both the partition function and marginals can be computed from a common optimization problem. Appendix A provides an intuitive development of the variational optimization problem in (1.36) in terms of finding a parameter setting that minimizes the KL-divergence between two distributions.

Unfortunately, simply casting the log-partition function as an optimization problem does not cause the intractability of computing the log-partition function to disappear. The optimization in (1.36) remains intractable for the following two reasons. First, the entropy $H(\boldsymbol{\mu})$ is intractable to compute in general. Second, the marginal polytope, $\mathcal{M}$, is difficult to characterize, requiring an exponential number of constraints in general.

To derive a tractable inference algorithm, one must approximate the optimization in (1.36). Let $\tilde{H}(\boldsymbol{\mu})$ be some approximation to the entropy and $\tilde{\mathcal{M}}$ be an approximation to the marginal polytope. Now define the approximate log-partition function, $\log \tilde{Z}(\boldsymbol{\theta})$, as

$$\log \tilde{Z}(\boldsymbol{\theta}) = \max_{\boldsymbol{\mu} \in \tilde{\mathcal{M}}} \left[\boldsymbol{\mu} \cdot \boldsymbol{\theta} + \tilde{H}(\boldsymbol{\mu}) \right]. \tag{1.39}$$

Approximate marginals, $\tilde{\boldsymbol{\mu}}(\boldsymbol{\theta})$, can then be computed by taking the first derivative of $\log \tilde{Z}(\boldsymbol{\theta})$ as in (1.38). In the following sections, we show how Loopy BP, GBP and WMB arise from different choices of $\tilde{H}(\boldsymbol{\mu})$ and $\tilde{\mathcal{M}}$.

Loopy Belief Propagation

Loopy BP has, to this point, been described as an algorithm that resulted from applying the Sum-Product updates, which are exact on a tree-structured graph, to a graph containing loops. This arc of starting from a tree-structured graph and moving to a loopy graph is also useful when introducing the variational perspective of Loopy BP. In particular, the approximations to the entropy and marginal polytope made by Loopy BP follow from considering a convenient alternate factorization of distributions on tree-structured factor graphs.

Consider some model $p(\boldsymbol{y}) \propto \prod_{\alpha \in F} \psi_\alpha(\boldsymbol{y}_\alpha)$ defined on a tree-structured factor graph $T = (V, F, E)$. Since $p(\boldsymbol{y})$ is tree-structured it can be expressed, or re-parameterized, as [88]:

$$p(\boldsymbol{y}) = \prod_{i \in V} \mu_i(y_i) \prod_{\alpha \in F} \frac{\mu_\alpha(\boldsymbol{y}_\alpha)}{\prod_{j \in \delta(\alpha)} \mu_j(y_j)}, \tag{1.40}$$

where, as described in Section 1.2.1, $\mu_i(y_i) = p(y_i)$ and $\mu_\alpha(\boldsymbol{y}_\alpha) = p(\boldsymbol{y}_\alpha)$ are the marginal probabilities of y_i and $\boldsymbol{y}_\alpha$, respectively. The entropy of a tree-structured distribution in this alternate factorization can be written concisely as:

$$H_T(\boldsymbol{\mu}) = -\sum_{\alpha \in F} \sum_{\boldsymbol{y}_\alpha} \mu_\alpha(\boldsymbol{y}_\alpha) \log \mu_\alpha(\boldsymbol{y}_\alpha) + \sum_{i \in V} (1 - |\delta(i)|) \sum_{y_i} \mu_i(y_i) \log \mu_i(y_i), \tag{1.41}$$

where $|\delta(i)|$ is the number of factors neighboring variable i in the factor graph.

In addition, the marginal polytope for tree structured models, which we denote as $\mathcal{M}(T)$ to be distinct from $\mathcal{M}$, can be compactly expressed as [88]:

$$\mathcal{M}(T) = \mathcal{M}_L = \left\{ \boldsymbol{\mu} \geq 0 \;\middle|\; \begin{array}{lll} \text{Normalization :} & \sum_{y_i} \mu_i(y_i) = 1, & \forall i \in V \\ \text{Consistency :} & \sum\limits_{\boldsymbol{y}_\alpha \setminus y_i} \mu_\alpha(\boldsymbol{y}_\alpha) = \mu_i(y_i), & \forall \alpha \in F, i \in \delta(\alpha), y_i \end{array} \right\}. \tag{1.42}$$

The collection of linear constraints defining $\mathcal{M}_L$ are commonly referred to as *local consistency* constraints because they ensure that any two factors α_1 and α_2 will have marginals that are consistent on their shared variables – i.e. $\sum_{\boldsymbol{y}_{\alpha_1} \setminus y_i} \mu_\alpha(\boldsymbol{y}_{\alpha_1}) = \mu_i(y_i) = \sum_{\boldsymbol{y}_{\alpha_2} \setminus y_i} \mu_\alpha(\boldsymbol{y}_{\alpha_2})$ for any $i \in \delta(\alpha_1), i \in \delta(\alpha_2)$.

Plugging (1.41) and (1.42) into (1.36) gives

$$\log Z(\theta) = \max_{\boldsymbol{\mu} \in \mathcal{M}_L} \left[\boldsymbol{\mu} \cdot \boldsymbol{\theta} + H_T(\boldsymbol{\mu}) \right], \tag{1.43}$$

which is an expression for the exact log-partition function of a tree-structured distribution. The first term in (1.43) is linear in $\boldsymbol{\mu}$ and the entropy is a concave function of $\boldsymbol{\mu}$. We can solve this constrained optimization problem by exploiting Lagrange multiplier theory to write down an expression for the stationary point conditions that hold at the maxima of (1.43). With a little bit of algebraic manipulation, one can derive a fixed-point iteration from these conditions that is equivalent to the Sum-Product BP updates in (1.20) and (1.21) [102].

The Loopy BP approximation to the log-partition function, $\log \tilde{Z}^{\text{BP}}(\boldsymbol{\theta})$, is revealed by solving the optimization problem in (1.43) for non-tree-structured graphs. In a loopy graph, the expression for the entropy, $H_T(\boldsymbol{\mu})$, becomes approximate and is known as the *Bethe* entropy approximation. In addition, the local consistency constraints are no longer sufficient to ensure that $\mathcal{M} = \mathcal{M}_L$ in a loopy graph. In fact, the local consistency constraints provide an outer bound $\mathcal{M}_L \supseteq \mathcal{M}$ on the marginal polytope, which means that there exist vectors $\boldsymbol{\mu} \in \mathcal{M}_L$ that do not correspond to any actual joint distribution. As a result, the vector $\boldsymbol{\mu}$ is commonly referred to as a vector of *pseudomarginals* or *beliefs* to make the distinction from the true marginals clear. Even though the expression for $\log \tilde{Z}^{\text{BP}}(\boldsymbol{\theta})$ is an approximation to $\log Z(\boldsymbol{\theta})$, we can optimize it via the same fixed-point iteration scheme, which results in the Loopy BP algorithm discussed in 1.4.1. We note, however, that the Bethe entropy approximation is typically not a concave function of $\boldsymbol{\mu}$, which means that if iterative message passing converges we may not converge to the global maximum of (1.43). And, even if we do find the global maximum of (1.43), we still only have an approximation to the marginals and log-partition function.

Generalized Belief Propagation

Generalized Belief Propagation (GBP) [101, 102, 43] was described in Section 1.4.1 as an algorithm that directly applied the cluster-tree message passing updates to a loopy cluster graph. In this section, we will see that GBP includes a much broader set of approximations. In particular, the variational perspective allows us to consider rich approximations to the entropy that cannot be realized by a cluster graph structure alone.

The GBP approximation is specified by a collection of regions $\mathcal{R}$. A *region* $\gamma \in \mathcal{R}$ is analogous to a cluster in Section 1.4.1 and is simply a subset of variables $\boldsymbol{y}_\gamma \subseteq \boldsymbol{y}$. We will assume that the collection of regions $\mathcal{R}$ are chosen so that each factor can be assigned to some region: meaning that there exists a $\gamma \in \mathcal{R}$ such that $\boldsymbol{y}_\alpha \subseteq \boldsymbol{y}_\gamma$ for all $\alpha \in F$.

GBP approximates the true entropy through a combination of marginal entropies

$$H(\boldsymbol{\mu}) = -\sum_{\boldsymbol{y}} p(\boldsymbol{y}; \theta(\boldsymbol{\mu})) \log p(\boldsymbol{y}; \theta(\boldsymbol{\mu})) \approx -\sum_{\gamma \in \mathcal{R}} c_\gamma \sum_{\boldsymbol{y}_\gamma} p(\boldsymbol{y}_\gamma; \theta(\boldsymbol{\mu})) \log p(\boldsymbol{y}_\gamma; \theta(\boldsymbol{\mu})) = \tilde{H}^{\text{GBP}}(\boldsymbol{\mu}), \tag{1.44}$$

where $c_\gamma \in \mathbb{R}$ is a parameter referred to as the *over-counting number* for region γ. This is commonly referred to as the Kikuchi approximation to the entropy [52, 67] and different collections of regions and settings of the counting numbers lead to different entropy approximations. For example, the Bethe entropy approximation used by Loopy BP is a specific instance of (1.44) with $\mathcal{R}$ equal to the union of the factors and variables, $\mathcal{R} = F \cup V$, the over-counting numbers for the factor regions equal to 1, $c_\gamma = 1$ for $\gamma \in F$, and the over-counting numbers for the inner regions set as $c_\gamma = 1 - |\delta(\gamma)|$ for $\gamma \in V$.

Like Loopy BP, GBP approximates the marginal polytope via local consistency constraints,

$$\mathcal{M}_{\text{GBP}} = \left\{ \boldsymbol{\mu} \geq 0 \;\middle|\; \begin{array}{lll} \text{Normalization :} & \sum_{\boldsymbol{y}_\gamma} \mu_\gamma(\boldsymbol{y}_\gamma) = 1, & \forall \gamma \in \mathcal{R} \\ \text{Consistency :} & \sum\limits_{\boldsymbol{y}_\gamma \setminus \boldsymbol{y}_\beta} \mu_\gamma(\boldsymbol{y}_\gamma) = \mu_\beta(\boldsymbol{y}_\beta), & \forall \gamma, \beta \in \mathcal{R}, \beta \subset \gamma, \boldsymbol{y}_\beta \end{array} \right\}, \quad (1.45)$$

where $\beta \subset \gamma$ means that $\boldsymbol{y}_\beta \subset \boldsymbol{y}_\gamma$. These local consistency constraints also provide an outer bound on the marginal polytope, $\mathcal{M}_{\text{GBP}} \supseteq \mathcal{M}$; however, they are often tighter than Loopy BP's local consistency constraints, $\mathcal{M}_L \supseteq \mathcal{M}_{\text{GBP}}$.

Plugging (1.44) and (1.45) into (1.39) gives the GBP approximation to the log-partition function

$$\log Z(\boldsymbol{\theta}) \approx \log \tilde{Z}^{\text{GBP}}(\boldsymbol{\theta}) = \max_{\boldsymbol{\mu} \in \mathcal{M}_{\text{GBP}}} \left[\boldsymbol{\mu} \cdot \boldsymbol{\theta} + \tilde{H}^{\text{GBP}}(\boldsymbol{\mu}) \right]. \quad (1.46)$$

We are often interested in GBP approximations with bounded computational complexity. Following our discussion of Weighted Mini-Bucket Elimination in Section 1.4.1, we use the *iBound* control parameter to limit the number of variables appearing in a region. We use $\mathcal{R}(i)$ to denote a collection of regions $\mathcal{R}$, such that $|\boldsymbol{y}_\gamma| \leq i$ for all $\gamma \in \mathcal{R}$ for an iBound of i. We also use $\log \tilde{Z}^{\text{GBP}}(\boldsymbol{\theta}, i)$ to indicate that the collection of regions $\mathcal{R}(i)$ defining the log-partition function approximation were limited by an iBound of i.

The accuracy of the GBP approximation depends mainly on the collection of regions used and the setting of the over-counting numbers. Many different methods exist for selecting collections of regions and counting numbers. We briefly review a few of the most important methods here. A far more detailed discussion of the issues related to region choice will follow in Chapter 3.

1. Cluster Variation Method [52, 67]: In his 1951 paper, Kikuchi proposed the Cluster Variation Method (CVM) which works as follows. We first choose a collection of *outer*

regions, $\mathcal{O}$. These outer regions can be the original factors in our model, $\mathcal{O} = F$, or regions over larger sets of variables. After identifying $\mathcal{O}$, we construct a collection of *inner regions*, $\mathcal{I}$, by taking the intersection of any k of the regions in $\mathcal{O}$: $\mathcal{I} = \{\beta \mid \beta = \cap_k \gamma_k, \gamma_k \in \mathcal{O}\}$. In other words, we take the intersection of all of the outer regions, the intersection of their intersections, ... The collection of regions used in the approximation is the union of the outer and inner regions: $\mathcal{R} = \mathcal{O} \cup \mathcal{I}$. After identifying $\mathcal{R}$, the counting numbers are set via the recursion:

$$c_\beta = 1 - \sum_{\gamma \supset \beta} c_\gamma, \tag{1.47}$$

where it follows that $c_\gamma = 1$ for all $\gamma \in \mathcal{O}$.

2. Junction-Graphs [2]: A *junction graph* is a specific type of cluster graph. Recall from Section 1.4.1 that a cluster graph $CG = (V, E)$ is a graph where each node $i \in V$ is associated with a subset of variables $\boldsymbol{y}_i \subset \boldsymbol{y}$ referred to as its scope and each edge $e = (i, j) \in E$ is associated with a subset of variables $\boldsymbol{y}_{ij} \subseteq \boldsymbol{y}_i \cap \boldsymbol{y}_j$ referred to as a separator. A junction-graph requires, for each variable y, that the sub-graph of CG consisting only of the nodes and edges with variable y in their scope and separators be a tree. The collection of regions used in a junction-graph approximation is, $\mathcal{R} = \mathcal{O} \cup \mathcal{I}$, where $\mathcal{O} = \{\boldsymbol{y}_i \mid i \in V\}$ is collection of outer regions corresponding to each cluster node and $\mathcal{I} = \{\boldsymbol{y}_{ij} \mid (i, j) \in E\}$ is a collection of inner regions on each separator. Over-counting numbers are set by the recursion in (1.47), which means that $c_\gamma = 1$ for each $\gamma \in \mathcal{O}$ and $c_\beta = -1$ for each $\beta \in \mathcal{I}$.

3. Join-Graphs [20, 62]: A join graph is equivalent to our definition of a cluster graph. And a junction-graph, in which the sub-graph induced by each variable is required to be tree-structured, is referred to as an *edge-minimal* join-graph. Dechter et al.[20] propose a novel scheme for constructing join-graphs with cluster scopes limited by an iBound of i. The *Join-Graph-Structuring* procedure works as follows. First, one applies the

mini-bucket partitioning strategy (discussed in Section 1.4.1), which induces a cluster tree over the collection of mini-buckets. To this cluster tree over mini-buckets we add edges connecting all of the mini-buckets in a bucket along a chain. This construction is illustrated in Figure 1.13, where the inter-mini-bucket edges are in bold red. The collection of regions $\mathcal{R}$ and over-counting numbers used in a join-graph approximation are identified as in the case of a junction-graph.

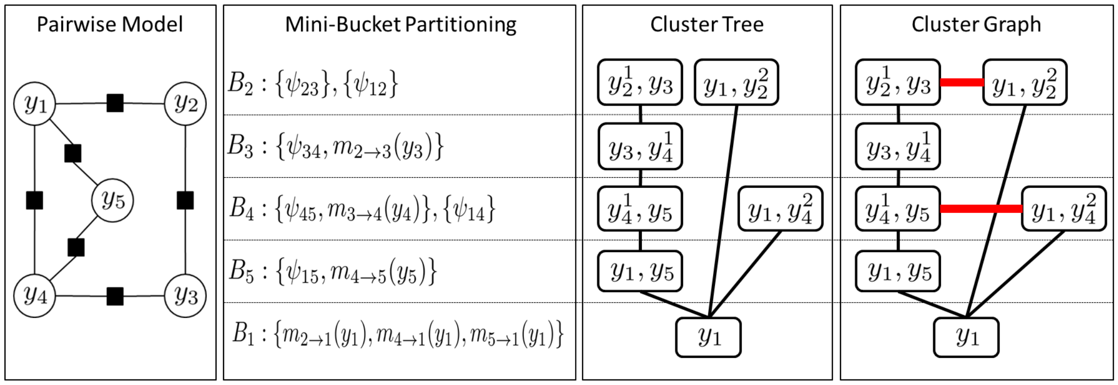

Pairwise Model
Mini-Bucket Partitioning
Cluster Tree
Cluster Graph
y_1
y_2
y_3
y_4
y_5
$B_2 : \{\psi_{23}\}, \{\psi_{12}\}$
$B_3 : \{\psi_{34}, m_{2\to3}(y_3)\}$
$B_4 : \{\psi_{45}, m_{3\to4}(y_4)\}, \{\psi_{14}\}$
$B_5 : \{\psi_{15}, m_{4\to5}(y_5)\}$
$B_1 : \{m_{2\to1}(y_1), m_{4\to1}(y_1), m_{5\to1}(y_1)\}$
y_2^1, y_3
y_1, y_2^2
y_3, y_4^1
y_4^1, y_5
y_1, y_4^2
y_1, y_5
y_1

Figure 1.13: Illustration of *Join-Graph-Structuring* procedure.

The differences between these alternate constructions are most easily visualized using a data structure known as a region graph. Given a collection of regions $\mathcal{R}$, a *region graph* (or Hasse diagram) is simply a directed acyclic graph whose nodes correspond to regions in $\mathcal{R}$ and an edge is drawn from region γ to region β if region γ *covers* region β. By cover, we mean that $\boldsymbol{y}_\gamma \supset \boldsymbol{y}_\beta$ and there exists no $\tau \in \mathcal{R}$ such that $\boldsymbol{y}_\gamma \supset \boldsymbol{y}_\tau \supset \boldsymbol{y}_\beta$. Figure 1.14 illustrates region graphs for both the CVM and junction-graph constructions with outer regions of $\mathcal{O} = \{\{1,2,5\},\{2,3,5\},\{3,4,5\},\{1,4,5\}\}$. Notice that the CVM region graph for this collection of regions has three levels. The depth of a CVM region graph is dictated by the overlap between the outer regions and may be much deeper than three levels in some problems. In contrast, the region graph for a junction-graph or join-graph will only have two levels: a top level for all of the cluster nodes and a bottom level for all of the edge separators.

The over-counting numbers for each region graph in Figure 1.14 are shown in red above each

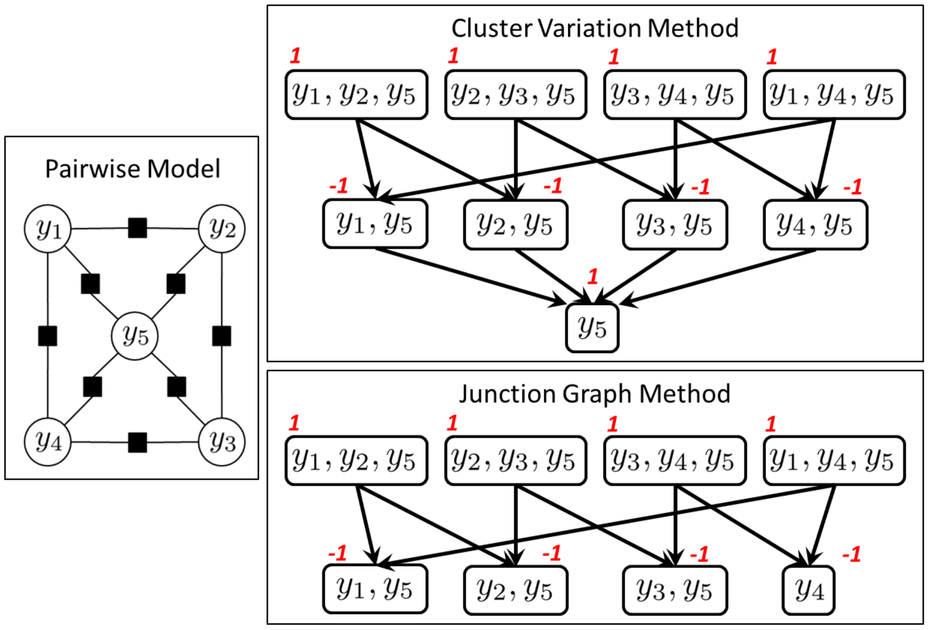


Figure 1.14: Illustration of Region Graphs for the CVM and Junction-Graph methods.

region graph node. Notice that the CVM region graph has inner regions with both positive and negative over-counting numbers. In contrast, inner regions of a junction or join graph will only ever have negative over-counting numbers. This is why at the beginning of this section we stated that the class of GBP approximations are broader than the approximations that can be realized by passing messages on a cluster graph structure. In particular, rich entropy approximations can be formed by both adding and subtracting *inner* marginal entropies in (1.44).

A region graph is also useful when organizing the computations needed to solve the constrained optimization problem in (1.46). Once again, we can use the theory of Lagrange multipliers to write down stationary point conditions and derive a fixed-point iteration. These fixed point updates can be viewed as messages passed on the region graph structure[102]. It is possible to write down updates on the region graph structure that are similar to the standard belief propagation updates – i.e., the belief at some region γ is a product of all factors assigned to γ and all messages passed into region γ. However, the updates that result for such *two-way* message-passing are far more complicated than the standard BP updates as they must properly account for the information sent by all ancestors and descendants of a region. As a result, we focus on a parent-child form of the algorithm, where messages are sent from a parent region γ to some child region β. Let

$$m_{\gamma \to \beta}(\boldsymbol{y}_\beta) = \frac{\sum_{\boldsymbol{y}_\gamma \setminus \boldsymbol{y}_\beta} \mu_\gamma(\boldsymbol{y}_\gamma)}{\mu_\beta(\boldsymbol{y}_\beta)} \tag{1.48}$$

be a "correction" message and update the region beliefs as

$$\mu(\boldsymbol{y}_\tau) \propto \mu(\boldsymbol{y}_\tau) \cdot m_{\gamma \to \beta}(\boldsymbol{y}_\beta), \tag{1.49}$$

for all regions $\tau \in \Delta(\beta) \setminus \Delta(\gamma)$, where $\Delta(i) = \{j \in \mathcal{R} \mid \boldsymbol{y}_j \supseteq \boldsymbol{y}_i\}$ is the set of ancestors of region i in the region graph and region i itself. The Kikuch entropy is typically not a concave function of $\boldsymbol{\mu}$, which means that if these fixed point updates converge they may not yield the vector of beliefs which maximize (1.46).

Weighted Mini-Bucket

Weighted Mini-Bucket Elimination (WMB) was described in Section 1.4.1 as an algorithm that computed an upper bound on the log-partition function. It computed this bound by passing *weighted* messages forward along the cluster tree induced by the mini-bucket partitioning scheme. The messages were weighted because WMB used the weighted summation operator (1.29) and Hölder's inequality (1.30) to build its bound. In this section, we will see that the WMB upper bound can be interpreted from a variational perspective as first approximating the marginal polytope constraint by a marginal polytope constraint on the partition-induced cluster tree and then approximating (actually bounding) the exact entropy using a weighted conditional entropy.

We begin by recalling the form of the WMB bound. Let π denote some elimination order

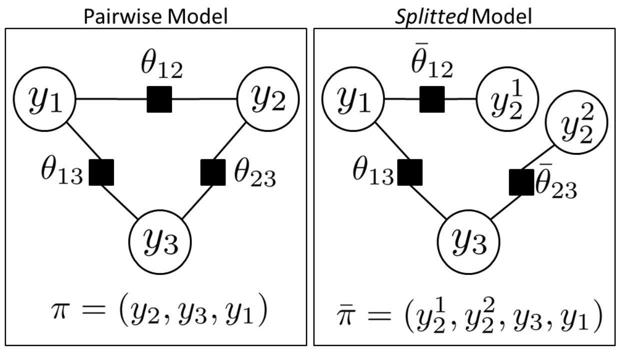


Figure 1.15: Demonstration of WMB notation.

and recall that the process of partitioning the functions assigned to a mini-bucket can be interpreted as replicating a variable (see e.g. Figure 1.11). Let $\bar{\boldsymbol{y}}_i = \{y_i^r\}_{r=1}^{R_i}$ be the collection of R_i replicates of variable y_i. Let $\bar{\boldsymbol{w}}_i = \{w_i^r\}_{r=1}^{R_i}$ be a corresponding collection of weights for each replicate, such that $\sum_{r=1}^{R_i} w_i^r = 1$. Let $\bar{\boldsymbol{y}} = \{\bar{\boldsymbol{y}}_1, ..., \bar{\boldsymbol{y}}_m\}$ and $\bar{\boldsymbol{w}} = \{\bar{\boldsymbol{w}}_1, ..., \bar{\boldsymbol{w}}_m\}$ be the collection of all replicates and weights, respectively. Let $\bar{\pi}$ be the extension of elimination order π to the *split* graph $\bar{G}$. Given an ordering $\bar{\pi} = (1, ..., \bar{m})$, the primal WMB bound is

$$\log Z(\boldsymbol{\theta}) \leq \log Z(\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{w}}) = \log \sum_{\bar{\boldsymbol{y}}_{\bar{m}}}^{\bar{\boldsymbol{w}}_{\bar{m}}} \cdots \sum_{\bar{\boldsymbol{y}}_1}^{\bar{\boldsymbol{w}}_1} \prod_{\alpha \in F} \exp(\bar{\theta}_\alpha(\bar{\boldsymbol{y}}_\alpha)), \tag{1.50}$$

where $\bar{\theta}_\alpha(\bar{\boldsymbol{y}}_\alpha)$ is the set of factors defined over the set of replicates $\bar{\boldsymbol{y}}$. The notation used to build this bound is demonstrated in Figure 1.15 for a simple pairwise model.

The WMB bound can also be constructed from a variational perspective. First, we approximate (replace) the marginal polytope $\mathcal{M}$ on our original graph G, with the marginal polytope on the split graph $\mathcal{M}(\bar{G})$. As a result, we replace searching for a mean vector $\boldsymbol{\mu} \in \mathcal{M}(G)$ with a searching for some extended mean vector $\bar{\boldsymbol{\mu}} \in \mathcal{M}(\bar{G})$. We then bound the entropy using a weighted sum of conditional entropies on the split graph

$$H(\boldsymbol{\mu}) \leq \tilde{H}_{\bar{\boldsymbol{w}}}(\bar{\boldsymbol{\mu}}) = \sum_{i \in \bar{\pi}} \bar{w}_i H(\bar{y}_i | \bar{\boldsymbol{y}}_{\delta(\bar{\pi}, i)}; \bar{\boldsymbol{\mu}}), \tag{1.51}$$

where the sum is over all variables i in the extended ordering $\bar{\pi}$, $\bar{w}_i$ is the weight assigned to variable i in the extended ordering and $\bar{\boldsymbol{y}}_{\delta(\bar{\pi},i)}$ is the set of neighbors subsequent to variable i in ordering $\bar{\pi}$ (e.g. $\delta(\bar{\pi}, y_2^1) = y_1$, $\delta(\bar{\pi}, y_2^2) = y_3$ and $\delta(\bar{\pi}, y_3) = y_1$ in Figure 1.15). The conditional entropy $H_w(\bar{y}_i|\bar{\boldsymbol{y}}_{\delta(\bar{\pi},i)}; \bar{\boldsymbol{\mu}})$ is computed using the marginals $\bar{\boldsymbol{\mu}}$. Plugging the marginal polytope approximation and conditional entropy bound into (1.39) gives the dual WMB bound to the log-partition function

$$\log Z(\boldsymbol{\theta}) \leq \max_{\bar{\boldsymbol{\mu}} \in \mathcal{M}(\bar{G})} \left[\bar{\boldsymbol{\mu}} \cdot \bar{\boldsymbol{\theta}} + \tilde{H}_{\bar{\boldsymbol{w}}}(\bar{\boldsymbol{\mu}}) \right]. \tag{1.52}$$

The optimization problem in (1.52) is equivalent to the dual problems encountered when using Tree Reweighted (TRW) Belief Propagation [86] or the Conditional Entropy Decomposition (CED) approach [31]. A variety of methods could in principle be used to directly optimize (1.52). However, as discussed in Section 1.4.1 and in [61] the primal WMB bound in (1.50) can be optimized efficiently via simple message passing updates, which is the route we follow in our later experiments involving WMB.

Chapter 2

A Bottom-Up Approach to Solving the Weighted Matching Problem using Belief Propagation

Many combinatorial optimization problems can be formulated as MAP inference problems in a graphical model. By formulating in this way, we can not only bring to bear a different set of algorithms for tackling these computationally challenging problems, but also leverage the vast literature on combinatorial optimization to inform our understanding of the performance of different MAP inference algorithms. In fact, many interesting connections between MAP inference and combinatorial optimization have been established over the years, including, for example, that MAP inference in ferromagnetic Ising models can be solved exactly by formulating it as a min-cut network optimization problem[3, 36].

In this chapter, we focus on a specific combinatorial optimization problem – the weighted matching problem – and a particular MAP inference algorithm – max-product belief propagation (BP). We focus on the matching problem for several reasons. First, it is a combinatorial optimization that can be expressed as an Integer Linear Program (ILP). Second, it is a classic problem with many known properties that can be utilized to provide a crisp characterization of the performance of an algorithm like max-product BP. Finally, many applications in wireless networking require the distributed computation of matchings – meaning that there is also a practical benefit to studying a distributed algorithm like BP in this setting[84, 16].

Our focus on the max-product BP algorithm is due to some recent results showing that BP is provably exact for certain types of graphical models containing many loops. These surprising results were shown for models corresponding to some well known combinatorial optimization problems, including matchings [5, 76, 44], perfect matchings [4], independent sets [77] and network flows[28]. The performance of BP in these problems can be characterized by properties of the LP relaxation of their ILP formulation – namely, equivalence to the LP optimized by BP, as well as uniqueness and tightness of the LP optima.

Unfortunately, the natural LP relaxation of the matching problem is often not tight in general (non-bipartite) graphs. As a result, BP is no longer a provably exact solver for many matching problems. The LP relaxation can be made tight, however, by iteratively adding *cutting plane* constraints that remove non-integral optima from consideration, while retaining all feasible solutions to the original ILP. So-called cutting plane methods are a popular approach to solving ILPs [17, 38, 37] and have also received attention in the graphical models community [83, 98, 65].

The main contributions of this chapter are:

1. We develop a cutting plane algorithm for the weighted matching problem that uses BP as its LP solver.

2. As we show in Section 2.4, the addition of tightening constraints in our cutting plane procedure ruins the convergence and correctness properties of BP, even when the relaxation is made tight. Another contribution of this chapter is thus the introduction

of a “fix” to the max-product algorithm that makes it exact for certain collections of cutting plane constraints.

3. Finally, the proposed “fix” to max-product BP increases the set of weighted matching problems that are provably solvable by the max-product algorithm and also adds to our understanding of the conditions needed for BP to be provaby exact.

2.1 Road Map of Chapter

In the remainder of this chapter we discuss several inter-related optimization problems. To aid the reader’s navigation of this chapter, we provide a road map in Figure 2.1.

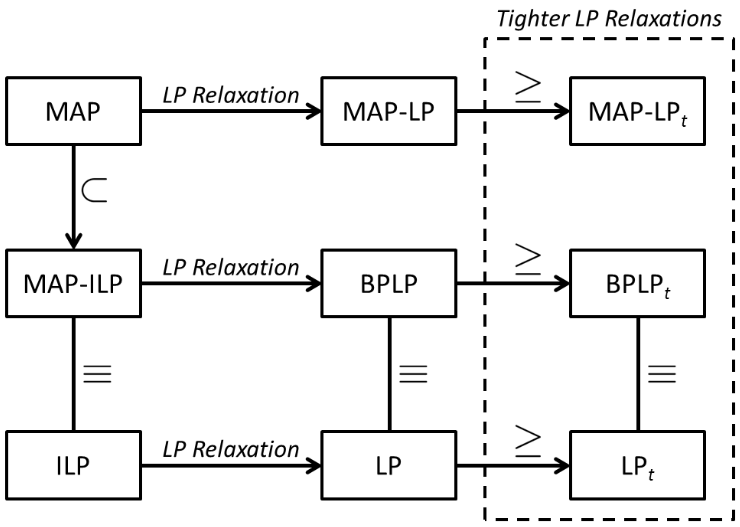


Figure 2.1: Road-map of inter-related combinatorial optimization problems.

We begin in Section 2.2 by reviewing the MAP inference problem in general graphical models and discuss the *standard* LP relaxation of the MAP problem, MAP-LP, which replaces the intractable marginal polytope with the local (pairwise) consistency polytope. We then introduce the ILP problem and discuss its natural LP relaxation, denoted as LP in the road map. Next, we introduce the MAP-ILP problem, which is nothing but the MAP problem for a specific class of graphical models encoding an ILP. We then discuss when the LP relaxation of MAP-ILP, denoted BPLP, is equivalent to the LP relaxation of the ILP problem.

After introducing these problems and their LP relaxations, in Section 2.3 we consider the cutting plane approach to tighten these relaxations and, ultimately, recover the global optima. In particular, we contrast approaches that tighten the bound on the marginal polytope in MAP-LP with approaches that remove non-integral solutions in LP and discuss how the BPLP relaxation can profit from both perspectives.

Last, in Section 2.4 we introduce the weighted matching problem and discuss how the max-product BP algorithm can be used in a cutting plane approach to solve matching problems. We conclude with some experimental results showing the efficacy of the cutting plane procedure and then discuss some exciting problems exposed by the work in this chapter.

2.2 MAP inference, ILPs and their LP Relaxations

2.2.1 MAP and MAP-LP

We begin by restating the MAP inference problem first introduced in Section 1.3. Given a factor graph, $G = (V, F, E)$, defining the joint probability distribution

$$p(\boldsymbol{x}) \propto \prod_{i \in V} \psi_i(x_i) \prod_{\alpha \in F} \psi_\alpha(\boldsymbol{x}_\alpha) \propto \exp\left(\sum_{i \in V} \theta_i(x_i) + \sum_{\alpha \in F} \theta_\alpha(\boldsymbol{x}_\alpha)\right),$$

find the state $\boldsymbol{x}^\star$ of maximum total probability

$$\text{MAP :} \qquad \boldsymbol{x}^\star = \arg\max_{\boldsymbol{x}} \; p(\boldsymbol{x}) = \arg\max_{\boldsymbol{x}} \; \sum_{i \in V} \theta_i(x_i) + \sum_{\alpha \in F} \theta_\alpha(\boldsymbol{x}_\alpha). \tag{2.1}$$

We now introduce the LP relaxation to the MAP problem. It was first proposed in [78] and has subsequently been studied by many others, including [90, 97, 98, 93]. We first express our distribution in the *standard overcomplete representation* introduced in Section 1.2.1. Let $\boldsymbol{b}(\boldsymbol{x})$ be a vector containing an indicator for every state of each variable and an indicator for every configuration of each factor:

$$\boldsymbol{b}(\boldsymbol{x}) = \{I[X_i = x_i] \mid i \in V, x_i\} \cup \{I[\boldsymbol{X}_\alpha = \boldsymbol{x}_\alpha] \mid \alpha \in F, \boldsymbol{x}_\alpha\} \tag{2.2}$$

The MAP problem in (2.1) is equivalent to the following optimization problem:

$$\arg\max \sum_{i \in V} \sum_{x_i} \theta_i(x_i) b_i(x_i) + \sum_{\alpha \in F} \sum_{\boldsymbol{x}_\alpha} \theta_\alpha(\boldsymbol{x}_\alpha) b_\alpha(\boldsymbol{x}_\alpha) \tag{2.3}$$

$$\text{s.t.} \quad b_\alpha(\boldsymbol{x}_\alpha) \in \{0,1\} \; \forall \alpha \in F, \; \boldsymbol{x}_\alpha \tag{2.4}$$

$$\sum_{\boldsymbol{x}_\alpha} b_\alpha(\boldsymbol{x}_\alpha) = 1 \; \forall \alpha \in F, \tag{2.5}$$

$$\sum_{\boldsymbol{x}_\alpha \setminus x_i} b_\alpha(\boldsymbol{x}_\alpha) = b_i(x_i) \; \forall \alpha \in F, \; i \in \delta(\alpha), x_i \tag{2.6}$$

where $b_i(x_i)$ is the component of $\boldsymbol{b}(\boldsymbol{x})$ corresponding the indicator $I[X_i = x_i]$ and $b_\alpha(\boldsymbol{x}_\alpha)$ is the component of $\boldsymbol{b}(\boldsymbol{x})$ corresponding to the indicator $I[\boldsymbol{X}_\alpha = \boldsymbol{x}_\alpha]$. Constraints (2.4) and (2.5) ensure that the indicator for exactly one configuration of each factor is active. The constraint in (2.6) ensures that the single active indicator for each factor is consistent with the active state indicators of each variable $i \in \delta(\alpha)$ in the factor.

The LP relaxation to the MAP problem, MAP-LP, is revealed by relaxing the integrality constraint in (2.4). We express the *MAP-LP problem* compactly as

$$\text{MAP-LP}: \qquad \boldsymbol{b}^\star = \underset{\boldsymbol{b} \in \mathcal{P}_{MAP-LP}}{\arg\max} \; \boldsymbol{b} \cdot \boldsymbol{\theta} \tag{2.7}$$

where $\boldsymbol{b}$ is shorthand for $\boldsymbol{b}(\boldsymbol{x})$, $\boldsymbol{\theta} = \boldsymbol{\theta}(\boldsymbol{x})$ is the corresponding vector of model parameters and $\mathcal{P}_{MAP-LP}$ is the *local consistency polytope*:

$$\mathcal{P}_{MAP-LP} = \left\{ \boldsymbol{b} \in [0,1]^l \;\middle|\; \begin{array}{cc} \sum_{\boldsymbol{x}_\alpha} b_\alpha(\boldsymbol{x}_\alpha) = 1 & \forall \alpha \in F \\ \sum_{\boldsymbol{x}_\alpha \setminus x_i} b_\alpha(\boldsymbol{x}_\alpha) = b_i(x_i) & \forall \alpha \in F, i \in \delta(\alpha), x_i \end{array} \right\}, \tag{2.8}$$

where the dimension of the vector $\boldsymbol{b}$ is $l = K|V| + \sum_{\alpha \in F} K^{|\delta(\alpha)|}$. MAP-LP is also commonly referred to as the *pairwise* LP relaxation[82] because the consistency constraints ensure that pairs of factors having a shared variable will have consistent beliefs on that shared variable.

2.2.2 ILP and LP

Many combinatorial optimization problems involve the optimization of a linear objective function over the integral vectors in some polytope. Such problems are referred to as Integer Linear Programs (ILPs) and take the following form:

$$\text{ILP}: \qquad \max\ \boldsymbol{c} \cdot \boldsymbol{x} \qquad \text{s.t.} \qquad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{d}, \qquad \boldsymbol{x} \in \mathbb{Z}^m, \tag{2.9}$$

where $\boldsymbol{x}$ is an m-dimensional vector of integers, $\boldsymbol{c} \in \mathbb{R}^m$ is a vector of real-valued weights and the bounded polyhedron, $\mathcal{P}$, is defined by the $r \times m$ matrix $\boldsymbol{A}$ and the vector $\boldsymbol{d} \in \mathbb{R}^r$ as

$$\mathcal{P} = \{\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{d}\}. \tag{2.10}$$

In the remainder of this section, we will assume that $\boldsymbol{x} \in \{0,1\}^m$ is a binary vector.

The natural Linear Programming (LP) relaxation of the binary ILP in (2.9) is defined as:

$$\text{LP}: \qquad \max\ \boldsymbol{c} \cdot \boldsymbol{x} \qquad \text{s.t.} \qquad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{d}, \qquad \boldsymbol{x} \in [0,1]^m, \tag{2.11}$$

where the only change from (2.9) is that $\boldsymbol{x}$ is no longer required to be integral. Note that the ILP in (2.9) optimizes over the polytope $\mathcal{P}_{ILP} = \{\boldsymbol{x} \in \{0,1\}^m | \boldsymbol{Ax} \leq \boldsymbol{d}\}$, while the LP relaxation optimizes over $\mathcal{P}_{LP} = \{\boldsymbol{x} \in [0,1]^m | \boldsymbol{Ax} \leq \boldsymbol{d}\}$. This implies that $\mathcal{P}_{LP} \supseteq \mathcal{P}_{ILP}$.

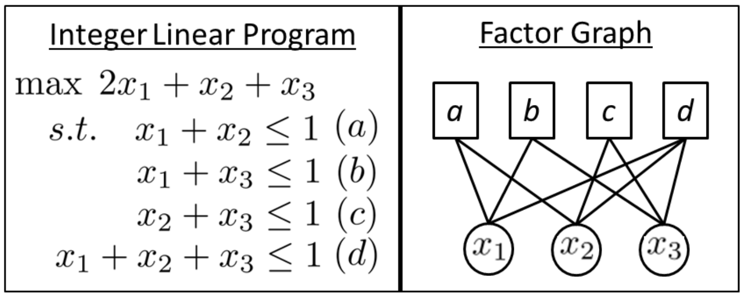
2.2.3 MAP-ILP and BPLP

The ILP in (2.9) can be formulated as a MAP inference problem by constructing a suitable graphical model. Let $\boldsymbol{x} \in \{0,1\}^m$ be a vector of binary random variables associated with each component of the vector $\boldsymbol{x}$ in (2.9). Let $i = 1, ..., m$ index the variables and $j = 1, ..., r$ index the rows of the constraint matrix $\boldsymbol{A}$. Consider the joint probability distribution

$$p(\boldsymbol{x}) \propto \prod_{i=1}^{m} e^{c_i x_i} \prod_{j=1}^{r} \psi_j(\boldsymbol{x}_{S_j}), \tag{2.12}$$

$$\psi_j(\boldsymbol{x}_{\mathcal{S}_j}) = \begin{cases} 1, & \text{if } (\boldsymbol{Ax})_j \leq d_j \\ 0, & \text{otherwise} \end{cases}, \tag{2.13}$$

where every row of matrix $\boldsymbol{A}$ is associated with a factor ψ_j defined over a subset of the variables $\boldsymbol{x}_{S_j}$, where $S_j = \{i : \boldsymbol{A}_{ji} \neq 0\}$. An illustration of this transformation is shown in Figure 2.2, where each factor node (depicted as squares) corresponds to one of the 4 inequalities in the ILP.


Integer Linear Program

$$\begin{aligned} \max\ & 2x_1 + x_2 + x_3 \\ s.t.\ & x_1 + x_2 \leq 1 \ (a) \\ & x_1 + x_3 \leq 1 \ (b) \\ & x_2 + x_3 \leq 1 \ (c) \\ & x_1 + x_2 + x_3 \leq 1 \ (d) \end{aligned}$$

Figure 2.2: Illustrating the transformation of an ILP to a graphical model.

The *MAP-ILP* problem is simply the MAP problem for graphical models of the specific form in (2.12)

$$\text{MAP-ILP} : \boldsymbol{x}^\star = \arg\max_{\boldsymbol{x}} \ \sum_{i=1}^{m} c_i x_i + \sum_{j=1}^{r} \theta_j(\boldsymbol{x}_{S_j}). \tag{2.14}$$

where $\theta_j(\boldsymbol{x}_{S_j}) = \log \psi_j(\boldsymbol{x}_{S_j})$. It is clear that $p(\boldsymbol{x}) \propto \prod_i e^{c_i x_i}$ for any 'feasible' assignment of $\boldsymbol{x}$ and that $p(\boldsymbol{x}) = 0$ otherwise. As a result, the MAP assignment in (2.14) will be equivalent to the vector optimizing the ILP in (2.9)[1].

We now formulate the natural LP relaxation to the MAP-ILP problem, which we refer to as *BPLP* because it is precisely the LP relaxation that max-product BP attempts to solve. Introducing a vector of indicators $\boldsymbol{b}(\boldsymbol{x})$ for each state of every variable and each configuration of every factor, we arrive at the following LP representation:

$$\text{BPLP} : \qquad \max \sum_i c_i b_i(x_i = 1) \tag{2.15}$$

$$\text{s.t.} \quad b_j(\boldsymbol{x}_{S_j}) \in [0,1] \ \forall j = 1..r, \ \boldsymbol{x}_{S_j}, \tag{2.16}$$

$$\sum_{\boldsymbol{x}_{S_j}} b_j(\boldsymbol{x}_{S_j}) = 1 \ \forall j = 1..r, \tag{2.17}$$

$$\sum_{\boldsymbol{x}_{S_j} \setminus x_i} b_j(\boldsymbol{x}_{S_j}) = b_i(x_i) \ \forall j = 1..r, i \in S_j, x_i \tag{2.18}$$

$$b_j(\boldsymbol{x}_{S_j}) = 0 \quad \text{if} \quad \sum_{i \in S_j} \boldsymbol{A}_{ji} x_i > d_j \ \forall j = 1..r. \tag{2.19}$$

The only difference between the *BPLP problem* and the *MAP-LP problem* in (2.7) is the addition of the feasibility constraints (2.19). To make this difference explicit, we absorb the feasiblity contraints into the local consistency polytop $\mathcal{P}_{\text{MAP-LP}}$ and define the *BPLP polytope*

[1]This is true assuming that there is a unique optimal assignment.

as:

$$\mathcal{P}_{BPLP} = \left\{ \boldsymbol{b} \in [0,1]^l \;\middle|\; \begin{array}{cc} \sum_{\boldsymbol{x}_{S_j}} b_j(\boldsymbol{x}_{S_j}) = 1 & \forall j = 1..r \\ \sum_{\boldsymbol{x}_{S_j} \setminus x_i} b_j(\boldsymbol{x}_{S_j}) = b_i(x_i) & \forall j = 1..r,\ i \in S_j,\ x_i \\ b_j(\boldsymbol{x}_{S_j}) = 0 \quad \text{if} \quad \sum_{i \in S_j} \boldsymbol{A}_{ji} x_i > d_j & \forall j = 1..r \end{array} \right\}, \tag{2.20}$$

where the dimensionality of the belief vector $\boldsymbol{b}$ in this formulation is $l = 2m + \sum_{j=1}^{r} 2^{|S_j|}$. We note that the BPLP relaxation optimizes a vector $\boldsymbol{b} \in [0,1]^l$, while the natural LP relaxation of the ILP problem in (2.11) optimizes a vector $\boldsymbol{x} \in [0,1]^m$.

2.2.4 Equivalence of LP and BPLP

The previous sections introduced two different LP relaxations of the binary ILP in (2.9) – namely, *LP* and *BPLP*. Alternative representations of an LP are important for two main reasons. First, the representation of an LP governs the performance of an LP solver. For example, the ellipsoid algorithm has run-time that is polynomial in the number of variables and constraints in an LP [51, 8], suggesting that solutions can be found more efficiently in a compact LP representation. Second, different representations of an LP are also useful for analytical purposes. For example, Feldman, Wainwright and Karger exploit an alternative LP relaxation of the max likelihood decoding problem to bound the error in their decoder[27].

We are primarily interested in alternative LP representations for analytical reasons. In particular, our characterization of the performance of BP will leverage properties of the standard LP relaxation of the matching problem. We now introduce the main result of this section, which is a precise characterization of when the LP problem in (2.11) and the BPLP problem in (2.15) are equivalent. Establishing this equivalence is a prerequisite for our later analysis of max-product BP.

Since $\mathcal{P}_{LP}$ is a polytope defined over variables $\boldsymbol{x} \in \{0,1\}^m$ and $\mathcal{P}_{BPLP}$ is a is a polytope defined over variables $\boldsymbol{b} \in \{0,1\}^l$ with $l > m$, we must first clarify what is meant by equivalence. For the remainder of this section, we will assume that each variable x_i in $\mathcal{P}_{LP}$ corresponds to the variable $b_i(x_i = 1)$ in $\mathcal{P}_{BPLP}$. The remaining $l - m$ variables in $\boldsymbol{b}$ are considered to be *auxiliary* and we denote them by $\bar{\boldsymbol{b}}$ so that $\boldsymbol{b} = (\boldsymbol{x}, \bar{\boldsymbol{b}})$. We now define the projection of the polytope $\mathcal{P}_{BPLP}$ onto $\boldsymbol{x}$ as:

$$\bar{\mathcal{P}}_{BPLP} = \{\boldsymbol{x} \mid \exists\, \bar{\boldsymbol{b}} \ \text{ s.t. } \ (\boldsymbol{x}, \bar{\boldsymbol{b}}) \in \mathcal{P}_{BPLP}\}.$$

We then say that $\mathcal{P}_{BPLP}$ is *equivalent* to $\mathcal{P}_{LP}$ if $\bar{\mathcal{P}}_{BPLP} = \mathcal{P}_{LP}$. A further discussion of polytope equivalence can be found in [27, 100].

Theorem 2.1. *Consider the polytope $\mathcal{P}_{LP} = \{\boldsymbol{x} \in [0,1]^m \mid \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{d}\}$. Then, the following properties hold:*

- *If $\mathcal{P}_{LP}$ has only $0-1$ integral vertices (i.e., extreme points), then $\mathcal{P}_{LP} \subseteq \bar{\mathcal{P}}_{BPLP}$.*
- *$\bar{\mathcal{P}}_{BPLP} \subseteq \mathcal{P}_{LP}$ (without any conditions).*

Proof. We first show that $\mathcal{P}_{LP} \subseteq \bar{\mathcal{P}}_{BPLP}$. To do so, we must show that if a vector $\boldsymbol{x} \in \mathcal{P}_{LP}$, then there exists a vector of beliefs $\boldsymbol{b}$ satisfying the constraints (2.18,2.17,2.19) of BPLP. In other words, given any $\boldsymbol{x} \in \mathcal{P}_{LP}$ we can find a setting of $\bar{\boldsymbol{b}}$ such that $\boldsymbol{b} = (\boldsymbol{x}, \bar{\boldsymbol{b}}) \in \mathcal{P}_{BPLP}$. By Caratheodorys theorem, since the polytope $\mathcal{P}_{LP}$ has only $0-1$ vertices, any point $\boldsymbol{x}$ in the polytope can be expressed as a convex combination of these vertices. The coefficients of the convex combination provide values for the factor beliefs $\{b_j(\boldsymbol{x}_{S_j})\}$ and the beliefs $b_i(x_i = 0)$ correspond to the variables $1 - x_i$ in the LP.

We next show that $\bar{\mathcal{P}}_{BPLP} \subseteq \mathcal{P}_{LP}$. In other words, if $\boldsymbol{b} = (\boldsymbol{x}, \bar{\boldsymbol{b}}) \in \mathcal{P}_{BPLP}$ then $\boldsymbol{x} \in \mathcal{P}_{LP}$. The value of $b_i(x_i = 0)$ is redundant within the BPLP polytope as $b_i(x_i = 0) = 1 - b_i(x_i = 1)$. From this, we derive for each row j of $\boldsymbol{A}$:

$$\begin{aligned}\sum_{i\in\mathcal{S}_j} A_{ji}x_i &= \sum_{i\in\mathcal{S}_j} A_{ji} \sum_{x_{\mathcal{S}_j}:x_i=1} b_j\left(x_{\mathcal{S}_j}\right) = \sum_{i\in\mathcal{S}_j} A_{ji} \sum_{x_{\mathcal{S}_j}} x_i b_j\left(x_{\mathcal{S}_j}\right) = \sum_{x_{\mathcal{S}_j}} \left(\sum_{i\in\mathcal{S}_j} A_{ji}x_i\right) b_j\left(x_{\mathcal{S}_j}\right)\\ &\le \sum_{x_{\mathcal{S}_j}} d_j b_j\left(x_{\mathcal{S}_j}\right) = d_j \sum_{x_{\mathcal{S}_j}} b_j\left(x_{\mathcal{S}_j}\right) = d_j,\end{aligned}$$

where (2.19) was used in the inequality.

□

We note (arguing by contradiction) that the condition in Theorem 2.1 for $\mathcal{P}_{LP} \subseteq \bar{\mathcal{P}}_{BPLP}$ is necessary. For example, consider a matrix $\boldsymbol{A}$ with a single constraint involving all the variables, $S_1 = \{1, \ldots, n\}$, and imagine that the polytope $\mathcal{P}_{LP}$ has a fractional vertex $\boldsymbol{x}$. Then there exists a vector of weights $\boldsymbol{c}$ such that $\boldsymbol{x}$ is the unique solution of the LP in (2.11). However, $\{b_i(x_i = 1) = x_i\}$ cannot satisfy (2.18), (2.17) and (2.19) for any factor $b_1(\boldsymbol{x}_{S_1})$ because $\boldsymbol{x}$ is a fractional vertex of P_{LP}.

This contradiction is illustrated in Figure 2.3. The unique optimum of the LP occurs at $x_1 = 1/2$, $x_2 = 1$, meaning that $b_1(x_1 = 1) = 1/2$ and $b_2(x_2 = 1) = 1$. The normalization constraints imply that $b_1(x_1 = 0) = 1/2$ and $b_2(x_2 = 0) = 0$ and the feasibility constraint requires $b_{12}(x_1 = 1, x_2 = 1) = 0$. However, there is no valid setting of the joint belief $b_{12}(x_1, x_2)$ that can simultaneously satisfy the consistency constraints $b_{12}(x_1 = 0, x_2 = 0) + b_{12}(x_1 = 0, x_2 = 1) = b_1(x_1 = 0) = 1/2$ and $b_{12}(x_1 = 0, x_2 = 1) + b_{12}(x_1 = 1, x_2 = 1) = b_2(x_2 = 1) = 1$ because $b_{12}(x_1 = 1, x_2 = 1) = 0$.

Theorem 2.1 implies the following corollary.

Corollary 2.2. *If the elements of matrix $\boldsymbol{A}$, $A_{ji} \in \{-1, 0, 1\}$ for all i, j, then $\mathcal{P}_{LP} \equiv \bar{\mathcal{P}}_{BPLP}$.*

Proof. Corollary 2.2 is proved using Theorem 2.1 and the fact that each vertex of a polytope

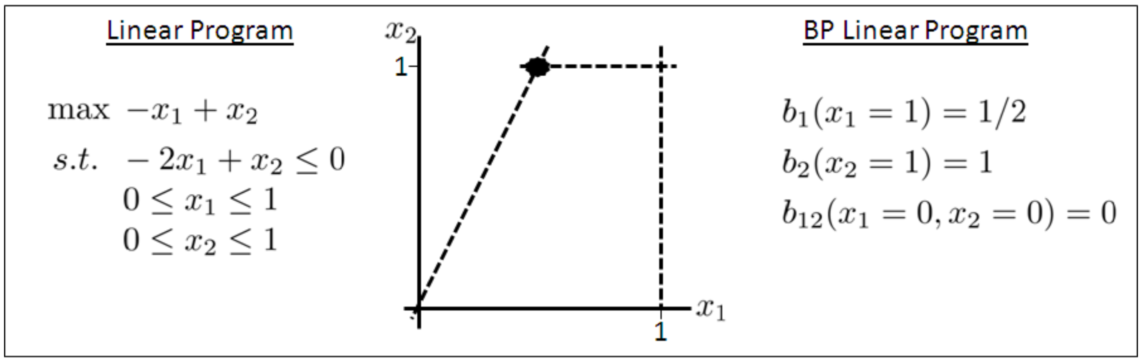


Figure 2.3: Illustration of the necessity of the LP having integral vertices for $\mathcal{P}_{LP} \subseteq \mathcal{P}_{BPLP}$.

can be expressed as the unique solution to a system of 'face' linear equalities (see e.g. [80]).

$\square$

In Section 2.4 we introduce a hierarchy of relaxations for the weighted matching problem that satisfy the conditions in Corollary 2.2 – a condition that aids our analysis of the cutting plane BP algorithm.

2.3 Cutting Plane Methods: A Bottom-Up Approach to Tightening LP Relaxations

The cutting plane method is an iterative approach to solving ILPs originally developed by Dantzig, Fulkerson and Johnson to tackle the Traveling Salesman Problem (TSP)[17]. It begins by solving the natural LP relaxation of the ILP to optimality. If the LP optimum, $\boldsymbol{x}^\star$, is integral, then it is an optimal solution to the ILP and the algorithm terminates; otherwise, there exists a linear constraint $\boldsymbol{a}_1\boldsymbol{x} \leq d_1$ that is satisfied by all solutions to the ILP, but violated by $\boldsymbol{x}^\star$. The inequality $\boldsymbol{a}_1\boldsymbol{x} \leq d_1$ is referred to as a *cutting plane* and we add it to the original LP, giving a new tighter polytope

$$\mathcal{P}_{LP} \supseteq \mathcal{P}_{LP_1} = \{\boldsymbol{x} \in [0,1]^m \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{d},\ \boldsymbol{a}_1\boldsymbol{x} \leq d_1\}. \tag{2.21}$$

This process is repeated by optimizing over $\mathcal{P}_{LP_1}$, checking if its optima is integral and adding a new cut constraint if necessary.

The cutting plane approach is illustrated in Figure 2.4. The solution to the natural LP relaxation in the left-hand plot occurs at a fractional vertex of $\mathcal{P}_{LP}$ (depicted by the dashed blue line) that lies outside of the ILP polytope $\mathcal{P}_{ILP}$ (depicted by the dotted red line). In the right-hand figure, we have added a cut inequality that removes the fractional solution and makes the new LP polytope, $\mathcal{P}_{LP_1}$, equivalent to the ILP polytope. The optima of this new LP is integral and coincides with the optima of the ILP.

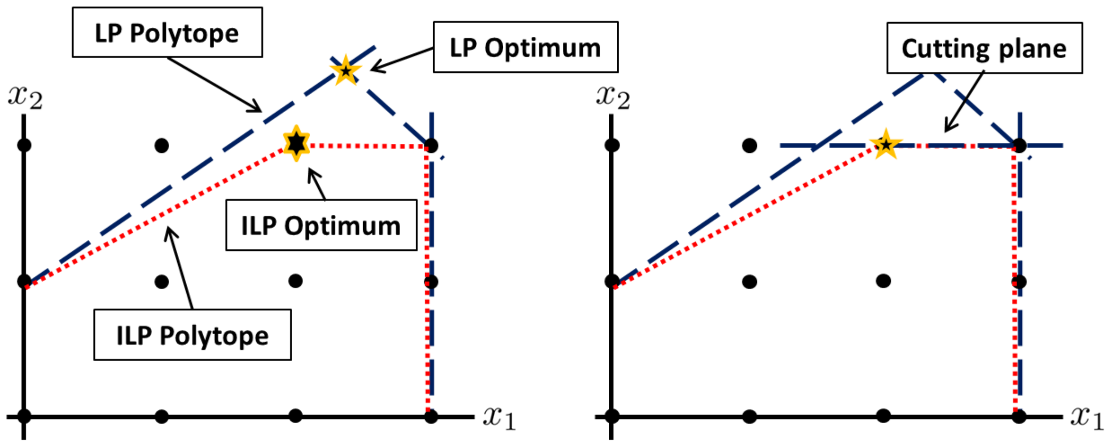


Figure 2.4: Illustration of the cutting plane method.

For the cutting plane method to be a viable approach to solving ILPs, we require:

1. An efficient algorithm for finding cutting planes;
2. An efficient algorithm for solving LPs;
3. A guarantee that the procedure will converge to an integral solution in a bounded (hopefully polynomial) number of cuts.

We discuss each of these requirements in turn.

2.3.1 Cut Generation

Cut generation is a subject that has received a lot of attention in the operations research community. Gomory popularized the use of cutting plane methods by extending the approach of Dantzig, Fulkerson and Johnson to general ILPs[34, 35] and his "rounding" procedure finds use in almost all commercial ILP solvers (e.g. CPLEX[1] and Gurobi[39]). However, far more effective cutting plane methods have been identified for many well-known ILPs. In particular, Padberg and Rao devised a polynomial time separation algorithm for the weighted matching problem and discuss its usage in the ellipsoidal algorithm[70]. The authors in [37, 85] discuss several cut generating heuristics for the matching problem and also consider their deployment in a Simplex-based cutting plane approach.

Cutting plane methods have also received attention in the graphical models community. In this setting, the cuts are employed to tighten the local (pairwise) relaxation, $\mathcal{P}_{MAP-LP}$, of the marginal polytope. The authors in [65] describe a cut generation procedure that tightens the pairwise relaxation of higher-order factors by enforcing consistency on larger subsets of shared variables. For example, adding the constraints

$$\left\{ \sum_{\boldsymbol{x}_\alpha \setminus x_i, x_j} b_\alpha(\boldsymbol{x}_\alpha) = b_{ij}(x_i, x_j) \; \forall i, j \in \alpha, \quad \sum_{x_i} b_{ij}(x_i, x_j) = b_j(x_j), \quad \sum_{x_j} b_{ij}(x_i, x_j) = b_i(x_i) \right\}$$

to $\mathcal{P}_{MAP-LP}$ will ensure consistency on pairs of shared variables. Sontag and Jaakkola propose a cutting plane algorithm for pairwise models that builds consistency in the opposite direction, by adding constraints to ensure that beliefs along a cycle are consistent with the beliefs on its constituent edges [83, 82]. It is an example of a lift-and-project method because the higher level of consistency is enforced by adding new variables into the relaxation that have no effect on the objective being optimized. Werner proposed a different lift-and-project procedure that iteratively adds constraints which rule out inconsistent partial-configurations in the model[98].

Since the BPLP problem is the relaxation to both an ILP problem and a MAP problem it can profit from both perspectives. That is, we can appeal to the heuristics discussed in [37] for the matching problem or even Gomory's generic procedure and add inequalities, $\boldsymbol{a}_t \boldsymbol{x} \leq d_t$, to grow the set of feasibility constraints in $\mathcal{P}_{BPLP}$; alternatively, we can appeal to the growing literature on tightening the pairwise MAP-LP relaxation and add constraints to ensure that the collection of beliefs $\boldsymbol{b}$ are consistent over larger sets of variables.

2.3.2 LP Solvers

Since the BPLP problem is the relaxation to both an ILP problem and a MAP problem it can seemingly benefit from using LP solvers designed for both types of problems. However, as discussed in Section 2.2.4, the polytope $\mathcal{P}_{BPLP}$ is much larger than the polytope $\mathcal{P}_{LP}$. This means that if we were to employ an off-the-shelf LP solver, we would prefer to optimize over $\mathcal{P}_{LP}$ directly. Instead, we focus our attention on message-passing approaches to solving LP relaxations of the MAP inference problem. For a review of classic LP solvers, such as Dantzig's Simplex method, we refer readers to the texts [79, 99].

We begin our discussion of message-passing algorithms with the max-product algorithm itself. When viewed from a variational perspective, the max-product algorithm can be seen as attempting to solve the LP in (2.15) by passing local messages. It is not guaranteed to converge, however, and even if it does converge, it may converge to a solution that is not the LP optimum[2].

These two difficulties – non-convergence and non-optimality – have both been addressed in the literature. A number of convergent message-passing algorithms that optimize different dual representations of the MAP-LP problem have been proposed, including Tree-Reweighted Max Product [89, 55], MPLP [33] and Max-Sum Diffusion [97]. These algorithms also come

[2]The solution will be optimal, however, with respect to a neighborhood of single-loops and trees[92].

with a nice guarantee upon convergence: if they converge to a unique integral assignment, then the assignment is the optimum. Unfortunately, convergence to such an assignment is not guaranteed (see [47] for a discussion of convergence issues and potential remedies). In fact, global optimality is only guaranteed by the TRW-S algorithm [55] when the model's factors are pairwise and submodular – conditions that are not satisfied by the graphical model formulation of the matching problem.

2.3.3 Convergence Guarantees

The last requirement of a viable cutting plane method is a bound on the number of iterations needed for the method to converge using a specific cut generating routine. In other words, we require a bound on the number of LPs needing to be solved. The first such bound was established by Gomory for binary ILPs[35]. He showed that using his generic "rounding" procedure he could produce an integral solution in 2^m iterations. Though most cutting plane methods come with weak or no convergence guarantees, they are usually quite efficient in practice. For example, Sontag and Jaakkola's cutting plane method is able to find the MAP assignment in all but one instance in a set of 269 very challenging protein side-chain prediction problems[83].

Better convergence guarantees appear to be possible for the matching problem for several reasons. First, Edmond's provided a complete polyhedral description of the matching polytope[25]. While it has an exponential number of inequalities, a naive procedure that simply added these constraints one-by-one would achieve the same guarantee as the Gomory procedure. Better still, is a recent result from Chandrasekaran et al., who showed that a cutting plane method using Edmond's inequalities converges in polynomial time for the weighted perfect matching problem[14]. While their procedure is quite involved and requires both the addition and removal of cutting planes in every iteration, it is, nonetheless, a remarkable result and certainly inspired this investigation into solving matchings via a cutting plane BP approach.

2.4 Finding Max Weight Matchings using BP

2.4.1 The Weighted Matching Problem

We now introduce the maximum weighted matching problem: given a graph $G = (V, E)$ with weights w_e on its edges $e \in E$, find a matching of maximum total weight, where a matching M is a subset of the edges E such that no two edges share a vertex and the weight of a matching is the sum of its constituent edge weights. If we let $\boldsymbol{x} = \{x_e\}_{e \in E}$, then the problem can be expressed as the following ILP:

$$\text{match-ILP:} \qquad \arg\max_{\boldsymbol{x}} \sum_{e \in E} w_e x_e \quad \text{s.t.} \sum_{e \in \delta(i)} x_e \leq 1, \ \ \forall i \in V; \ \ x_e \in \{0, 1\}, \tag{2.22}$$

where $\delta(i) = \{e = (i, j) \in E\}$ is the set of edges adjacent to vertex i.

The natural LP relaxation of the matching ILP, which we refer to as *m-ILP*, is formed by replacing $x_e \in \{0, 1\}$ by $x_e \in [0, 1]$. The resulting relaxation is often referred to as the *bipartite* relaxation because it is provably tight when G is bipartite. We can make the relaxation tight on non-bipartite graphs, as famously shown by Edmonds [25], by adding a set of *blossom* inequalities:

$$\text{match-blossom-LP}: \quad \max_{\boldsymbol{x}} \sum_{e \in E} w_e x_e \tag{2.23}$$

$$\text{s.t.} \quad \sum_{e \in \delta(i)} x_e \leq 1, \ \ \forall i \in V; \qquad \sum_{e \in E(S)} x_e \leq \frac{|S|-1}{2}, \qquad \forall S \in \mathcal{S}; \quad x_e \in [0,1],$$

where $E(S) = \{(i,j) \in E : i,j \in S\}$ is the set of edges with both ends in S and $\mathcal{S} \subset 2^{|V|}$ is the set of all odd-sized sets of vertices in G, referred to as *blossoms* by Edmonds. The blossom inequalities imply that an odd cycle of length $2l+1$ can have at most l edges in a matching. For example, a matching can contain at most 1 edge in any set of 3 vertices.

Figure 2.5 illustrates four simple matching problems with fractional bipartite relaxations. In matching (a), there is a unique fractional LP optimum at $x^\star_{12} = x^\star_{13} = x^\star_{23} = 1/2$. The bipartite relaxation of (a) can be made tight by adding the blossom constraint $x_{12}+x_{13}+x_{23} \leq 1$, in which case we recover the unique integral solution $x^\star_{12} = 1, \ x^\star_{13} = x^\star_{23} = 0$. The bipartite relaxation of problem (b) has many LP optima – namely, $x^\star_{12} = x^\star_{13} = x^\star_{23} = 1/2$; $x^\star_{12} = 1, \ x^\star_{13} = x^\star_{23} = 0$; and any convex combination of the two. By adding the blossom constraint, we recover the unique integral solution at $x^\star_{12} = 1, \ x^\star_{13} = x^\star_{23} = 0$. The bipartite relaxation of problem (c) has a unique fractional LP optimum at $x^\star_{12} = x^\star_{13} = x^\star_{23} = 1/2$, but a non-unique LP optima upon adding the blossom constraint. Finally, the bipartite relaxation of problem (d) is tight and has a unique integral optimum at $x^\star_{12} = 1, \ x^\star_{13} = x^\star_{23} = 0$.

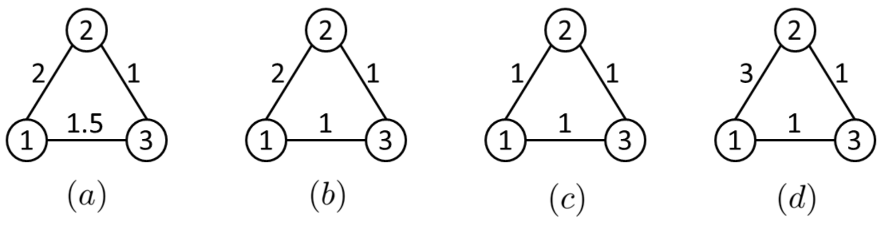


Figure 2.5: Illustration of simple matching problems with fractional and non-fractional LP optima.

2.4.2 Weighted Matching as a MAP problem

The weighted matching problem can be formulated as a MAP problem by associating a random variable with each edge, $\boldsymbol{x} = \{x_e\}_{e\in E}$, and constructing the following model:

$$p(\boldsymbol{x}) \;\propto\; \prod_{e\in E} e^{w_e x_e} \prod_{i\in V} \psi_i(\boldsymbol{x}_i) \prod_{S\in\mathcal{S}} \psi_S(\boldsymbol{x}_S), \tag{2.24}$$

$$\psi_i(\boldsymbol{x}_i) = \begin{cases} 1, & \text{if } \sum_{e\in\delta(i)} x_e \le 1 \\ 0, & \text{otherwise} \end{cases}, \quad \psi_S(\boldsymbol{x}_S) = \begin{cases} 1, & \text{if } \sum_{e\in E(S)} x_e \le \frac{|S|-1}{2} \\ 0, & \text{otherwise} \end{cases}, \tag{2.25}$$

where ψ_i are vertex factors defined over variables $\boldsymbol{x}_i = \{x_e : e \in \delta(i)\}$ and ψ_S are blossom factors defined over $\boldsymbol{x}_S = \{x_e : e \in E(S)\}$. The matching MAP problem is then

$$\text{MAP-match:} \quad \arg\max_{\boldsymbol{x}} \sum_{e\in E} w_e x_e + \sum_{i\in V} \theta_i(x_i) + \sum_{S\in\mathcal{S}} \theta_S(\boldsymbol{x}_S),$$

where $\theta_i(\boldsymbol{x}_i) = \log \psi_i(\boldsymbol{x}_i)$ and $\theta_S(\boldsymbol{x}_S) = \log \psi_S(\boldsymbol{x}_S)$.

Note that this is a specific realization of the MAP-ILP problem in (2.14). Further, notice that the coefficients of the vertex constraints and blossom constraints in (2.23) are either 0 or 1. This means that the conditions of Corollary 2.2 are satisfied and implies that the LP relaxation of the MAP-match problem is equivalent to the match-blossom-LP problem.

2.4.3 Max-Product BP for Weighted Matchings

We now introduce the max-product algorithm for the weighted matching model in (2.24). We present the factor-graph form of max-product. The max-product algorithm updates the set of messages between the edge variables and the vertex and blossom factors using the update rules shown in Figure 2.6, where we assume that edge $e = (i, j)$ and t denotes time.

Variable-to-Vertex: $$m^{t+1}_{e\to i}(x_e) = e^{w_e x_e}\ m^t_{j\to e}(x_e) \prod_{S:e\in E(S)} m^t_{S\to e}(x_e)$$

Variable-to-Blossom: $$m^{t+1}_{e\to S}(x_e) = e^{w_e x_e}\ m^t_{i\to e}(x_e) m^t_{j\to e}(x_e) \prod_{S':e\in E(S')\setminus S} m^t_{S'\to e}(x_e)$$

Vertex-to-Variable: $$m^{t+1}_{i\to e}(x_e) = \max_{\mathbf{x}_i \setminus x_e} \left\{ \psi_i(\mathbf{x}_i) \prod_{e'\in\delta(i)\setminus e} m^t_{e'\to i}(x_{e'}) \right\}$$

Blossom-to-Variable: $$m^{t+1}_{S\to e}(x_e) = \max_{\mathbf{x}_S \setminus x_e} \left\{ \psi_S(\mathbf{x}_S) \prod_{e'\in E(S)\setminus e} m^t_{e'\to S}(x_{e'}) \right\}$$

Figure 2.6: Max-product message updates for the Max Weight Matching problem.

At time $t = 0$ the messages are initialized to 1 and the belief on edge e at time t is

$$b^t(x_e) = e^{w_e x_e}\ m^t_{i\to e}(x_e)\ m^t_{j\to e}(x_e) \prod_{S:e\in E(S)} m^t_{S\to e}(x_e).$$

The algorithm outputs a MAP estimate at time t, $\boldsymbol{x}^{BP}(t) = \{x^{BP}_e(t)\}$, using the beliefs and the decoding rule:

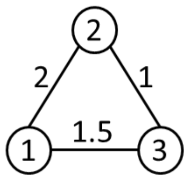
$$x^{BP}_e(t) = \begin{cases} 1 & \text{if } b^t_e(0) < b^t_e(1) \\ ? & \text{if } b^t_e(0) = b^t_e(1) \\ 0 & \text{if } b^t_e(0) > b^t_e(1) \end{cases} \cdot$$

Sanghavi, Malioutov and Willsky proved the following theorem connecting the performance of the max-product algorithm and the bipartite relaxation of the matching ILP [76]:

Theorem 2.3. *If the solution to match-blossom-LP with $\mathcal{S} = \emptyset$ (i.e. the bipartite relaxation) is integral and unique, then the BP MAP estimate, $\boldsymbol{x}^{BP}(t)$, of the model in* (2.24) *will converge to the maximum weight matching, $\boldsymbol{x}^\star$.*

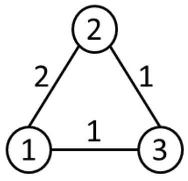
Figure 2.7 shows the MAP estimates produced by max-product for the four matching prob-

(a)



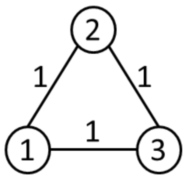
$t =$	1	2	3	4	5	6	7	8	9	10
$x_{12}^{BP}(t)$	0	0	1	1	0	0	1	1	0	0
$x_{13}^{BP}(t)$	0	0	1	1	0	0	1	1	0	0
$x_{23}^{BP}(t)$	0	0	1	1	0	0	1	1	0	0

(b)



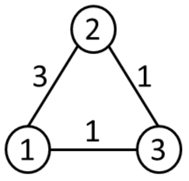
$t =$	1	2	3	4	5	6	7	8	9	10
$x_{12}^{BP}(t)$	?	?	1	1	?	?	?	?	?	?
$x_{13}^{BP}(t)$	0	0	?	?	0	0	?	?	?	?
$x_{23}^{BP}(t)$	0	0	?	?	0	0	?	?	?	?

(c)



$t =$	1	2	3	4	5	6	7	8	9	10
$x_{12}^{BP}(t)$	0	0	1	1	0	0	1	1	0	0
$x_{13}^{BP}(t)$	0	0	1	1	0	0	1	1	0	0
$x_{23}^{BP}(t)$	0	0	1	1	0	0	1	1	0	0

(d)



$t =$	1	2	3	4	5	6	7	8	9	10
$x_{12}^{BP}(t)$	1	1	1	1	1	1	1	1	1	1
$x_{13}^{BP}(t)$	0	0	0	0	0	0	0	0	0	0
$x_{23}^{BP}(t)$	0	0	0	0	0	0	0	0	0	0

Figure 2.7: BP MAP estimates for four sample matching problems.

lems discussed in the previous section. Notice that BP converges to the maximum weight matching only in matching problem (d); in the other problems BP either converges to an uncertain estimate, or fails to converge entirely.

One would hope that the result of Theorem 2.3 would extend beyond the bipartite relaxation because the bipartite relaxation can be made tight by adding blossoms. However, even when the match-blossom-LP is made tight and has a unique solution, max-product will still fail when $\mathcal{S} \neq \emptyset$. For example, one can observe that BP run on a model with an additional factor encoding the lone blossom constraint will fail on both matching problems (a) and (b) in Figure 2.5. This is true, despite the fact that the blossom constraint makes match-blossom-LP tight!

We address this shortcoming in the remainder of this chapter by first proposing a transformation to the model (2.24) that restores the convergence and correctness of BP. We then introduce our cutting plane approach to solving matching problems.

2.4.4 A Graphical Transformation for Exact BP

The loss of convergence and correctness when the match-blossom-LP is tight (and unique) but $\mathcal{S} \neq \emptyset$ motivates the work in this section. We begin by first imposing a restriction on the set $\mathcal{S}$ of blossom constraints that can be used to tighten the relaxation. In the remainder of this chapter, we only consider sets of odd-sized cycles $\mathcal{C} \subset \mathcal{S}$ that are non-intersecting in edges – i.e., $E(C_1) \cap E(C_2) = \emptyset$ for $C_1, C_2 \in \mathcal{C}$. Under this restriction, we propose a new graphical model that is equivalent to the model in (2.24), such that when BP is run on this new model it will converge to the max weight matching assignment whenever match-blossom-LP is tight (and unique).

The motivation for our graphical transformation comes in part from Edmond's matching algorithm [25, 54]. In that algorithm one collapses or contracts all of the vertices in a blossom into a single vertex. The collapsed blossom vertex is then treated as an ordinary vertex because of the following key property: if graph G' is a graph formed from graph G by collapsing some blossom C, then a matching in G' can always be extended to a matching in G. The basic idea is that if we matched some edge into the collapsed blossom vertex, then we can match the remaining even number of nodes to each other using only edges internal to the blossom. In our proposed transformation, we do not collapse the entire blossom to a single vertex; instead, we introduce a new vertex for the cycle blossom C, remove the edges along the cycle C and then add edges from the blossom vertices to the newly added cycle vertex. As we shall see, this transformation has the same effect as the previous collapsing transformation in that any matching in the new graph G' can be extended to a matching in the original graph G.

The new graphical model is defined on an auxiliary graph $G' = (V', E')$ with new edge weights $\{w'_e : e \in E'\}$ defined as follows:

$$V' = V \cup \{i_C : C \in \mathcal{C}\}, \qquad E' = E \cup \{(i_C, j) : j \in V(C), C \in \mathcal{C}\} \setminus \{e : e \in \cup_{C \in \mathcal{C}} E(C)\}$$

$$w'_e = \begin{cases} \frac{1}{2}\sum_{e' \in E(C)} (-1)^{d_C(j,e')} w_{e'} & \text{if } e = (i_C, j) \quad \text{for some } C \in \mathcal{C} \\ w_e & \text{otherwise} \end{cases}.$$

Here $d_C(j, e)$ is the graph distance between vertex j and edge e in cycle C. For example, if $C = (j_1, j_2, \ldots, j_5)$ is a cycle on 5 vertices and $e = (j_2, j_3)$, then $d_C(j_1, e) = 1$, $d_C(j_4, e) = 1$ and $d_C(j_5, e) = 2$.

Figure 2.8 illustrates the graphical transformation. The simple cycle $C = (1, 2, 3, 4, 5)$ in graph G on the left is *collapsed* and replaced by a single vertex i_C represented as a square in graph G' on the right. Notice that each edge along the cycle, e.g. $(1, 2)$, is replaced by an edge into the newly introduced vertex, e.g. $(1, i_C)$, while the edges not in the cycle, e.g. $(2, 4)$, are unaltered. Importantly, while the transform introduces new vertices, meaning that $|V'| > |V|$, the number of edges in G and G' remain the same $|E'| = |E|$. This transformation also explains our restriction to sets of blossoms that are odd-sized cycles not intersecting in edge.

In the example in Figure 2.8, the maximum weight matching is $M^\star = \{(1, 2), (3, 4)\}$ with a total weight of 12. However, note that the bipartite relaxation is not tight as setting $x_{24} = x_{35} = 0$ and $x_{12} = x_{23} = x_{34} = x_{45} = x_{15} = 1/2$ gives a total weight of 12.5. The addition of a blossom constraint on cycle $C = (1, 2, 3, 4, 5)$ makes the relaxation tight. In the new graph G', edge $(1, i_C)$ is given weight $w'_{1C} = 1/2(w_{12} - w_{23} + w_{34} - w_{45} + w_{15}) = 9/2$. The remaining cycle edge weights are $w'_{2C} = 3/2$, $w'_{3C} = 5/2$, $w'_{4C} = 7/2$ and $w'_{5C} = 1/2$, while the weights of edges $(2, 4)$ and $(3, 5)$ unchanged.

We now define a new graphical model that makes use of this graphical transformation. We

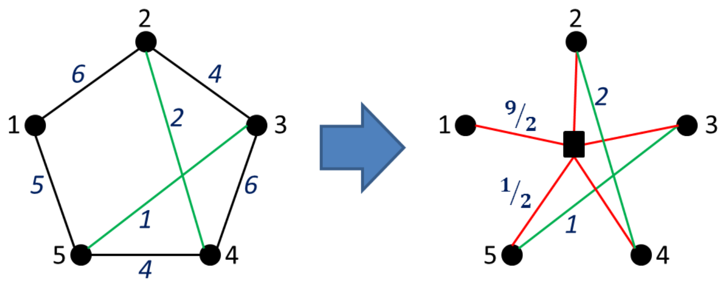


Figure 2.8: Illustration of the graphical transformation from G (left) to G' (right).

will use $\boldsymbol{y} = \{y_e \mid e \in E'\}$ to denote edge variables in the new model in order to provide a distinction from the variables of the original matching model in (2.24).

We begin by noting that the blossom factor, $\psi_C(\boldsymbol{x}_C)$, ensured that the set of edges $e \in E(C)$ constituted a matching on cycle C. Our graphical transformation eliminated each edge in $E(C)$ and replaced them with a new set of edges $\{(i, i_C) \mid i \in V(C)\}$. As a result, we must define a new blossom factor over the new edge variables, $\boldsymbol{y}_C$. Consider the following linear mapping between the original and new edge variables:

$$y_e = \begin{cases} \sum_{e^\dagger \in E(C) \cap \delta(i)} x_{e^\dagger} & \text{if } e = (i, i_C) \\ x_e & \text{otherwise} \end{cases}, \quad x_e = \begin{cases} \frac{1}{2} \sum_{j \in V(C)} (-1)^{d_C(j,e)} y_{i_C, j} & \text{if } e \in \bigcup_{C \in \mathcal{C}} E(C) \\ y_e & \text{otherwise} \end{cases}.$$

This mapping can be used to uniquely decode matchings from edges in G to edges in G' and vice-versa. For example, consider the set of edges in the optimal matching of graph G, $x_{12} = x_{34} = 1$. Using the mapping, we see that this corresponds to setting edges $y_{1,i_C} = y_{2,i_C} = y_{3,i_C} = y_{4,i_C} = 1$ in G'. Now, consider the assignment $y_{3,i_C} = y_{4,i_C} = y_{5,i_C} = 1$ in G'. Using the mapping, we see that this corresponds to setting $x_{34} = x_{45} = 1$ in G, which is clearly not a valid matching in G. As another example, consider the setting $y_{3,i_C} = 1$ in graph G'. Using the mapping, we see that this corresponds to setting $x_{23} = x_{34} = 1/2$ in G, which is also not a valid (integral) matching in G. One can easily verify that the following factor definition is non-zero only for settings of $\boldsymbol{y}_C$ that map to valid, integral matchings on cycle C in graph G:

$$\psi_C(\boldsymbol{y}_C) = \begin{cases} 0 & \text{if } \sum_{e\in\delta(i_C)} y_e > |C|-1 \\ 0 & \text{if } \sum_{j\in V(C)} (-1)^{d_C(j,e)} y_{i_C,j} \notin \{0,2\} \text{ for some } e \in E(C) \\ 1 & \text{otherwise} \end{cases} \cdot \tag{2.26}$$

With this mapping established, we now introduce the new graphical model:

$$p(\boldsymbol{y}) \;\propto\; \prod_{e\in E'} e^{w'_e y_e} \prod_{i\in V} \psi_i(\boldsymbol{y}_i) \prod_{C\in\mathcal{C}} \psi_C(\boldsymbol{y}_C), \tag{2.27}$$

where $\psi_C(\boldsymbol{y}_C)$ is as defined in (2.26) and $\psi_i(\boldsymbol{y}_i) = \begin{cases} 1 \text{ if } \sum_{e\in\delta(i)} y_e \le 1;\ 0 \text{ otherwise}\end{cases}$. Under the linear mapping between $\boldsymbol{x}$ and $\boldsymbol{y}$ one can easily verify that $\sum_{e\in E} w_e x_e = \sum_{e\in E'} w'_e y_e$. And, since the factor $\psi_C(\boldsymbol{y}_C)$ imposes the same blossom constraint as the factor $\psi_C(\boldsymbol{x}_C)$, we can conclude that the new graphical model (2.27) is *equivalent* to the original model (2.24) – every feasible assignment of $\boldsymbol{y}$ corresponds to a feasible assignment of $\boldsymbol{x}$ of the same total weight and vice-versa.

It is not hard to check that the number of operations required to update messages at each round of BP under the new GM is $O(|V||E|)$, as messages updates involving factor ψ_C require solving a matching problem on a simple cycle – which can be done efficiently via dynamic programming in time $\mathcal{O}(|C|)$ – and the summation of the numbers of edges of non-intersecting cycles is at most $|E|$. We are now ready to state the main result of this section.

Theorem 2.4. *If the solution of match-blossom-LP is integral and unique, then the BP MAP estimate $\boldsymbol{y}^{BP}(t)$ under the model (2.27) converges to the MAP assignment $\boldsymbol{y}^*$. Furthermore,* *the max weight matching assignment* $\boldsymbol{x}^*$ *is recoverable from* $\boldsymbol{y}^*$ *as:*

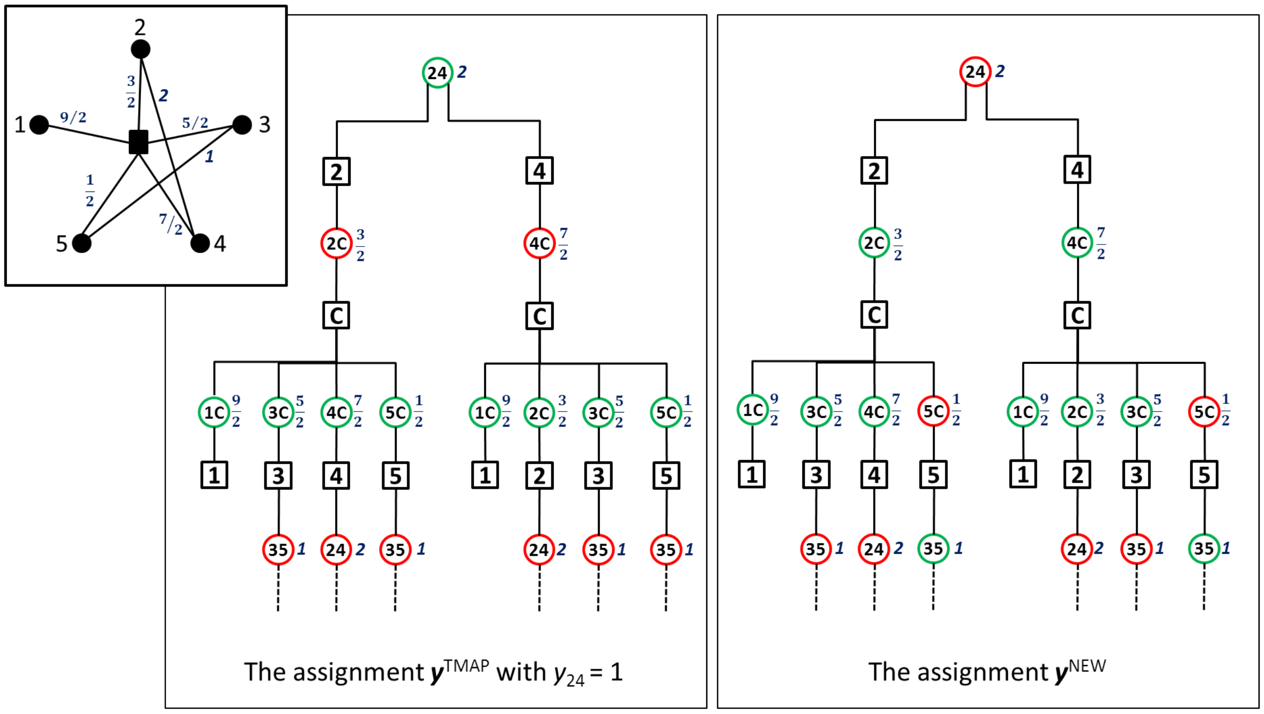
$$x_e^* = \begin{cases} 1/2 \sum_{j \in V(C)} (-1)^{d_C(j,e)} y^*_{i_C,j} & \text{if } \; e \in \bigcup_{C \in \mathcal{C}} E(C) \\ y_e^* & \text{otherwise} \end{cases}. \tag{2.28}$$

The proof of Theorem 2.4 is quite involved and is provided in Appendix B. We also establish the convergence time of the BP algorithm under the model in (2.27). We devote the remainder of this section to a sketch of the main ideas involved in the proof.

To prove the theorem, we need to show that when t is large enough the BP MAP estimate $\boldsymbol{y}^{\mathrm{BP}}(t)$ can be used to recover the edges $e \in M^\star$ of the max weight matching. Since the new matching model is equivalent to the original matching model, the MAP assignment $\boldsymbol{x}^\star$ in (2.24) is recoverable from the MAP assignment $\boldsymbol{y}^\star$ in (2.27). We therefore focus on showing that BP recovers the MAP assignment $\boldsymbol{y}^\star$ when match-blossom-LP has a unique integral optima.

The main tool utilized by the proof is the computation tree [91, 92]. Computation trees "unwrap" a factor graph in order to show the messages passed by BP – the basic idea being that the messages sent at time t depend, on the messages received at time $t-1$, which in turn depend on the messages received at time $t-2$, etc... Figure 2.9 illustrates the computation tree rooted at edge variable y_{24} out to a depth of $t = 6$. As in a regular factor graph, variables are depicted by circles and vertex and blossom factors by squares in the computation tree.

Our proof derives a contradiction via the following fact relating BP and computation trees: the BP MAP estimate at time t is the MAP assignment of the computation tree of depth t[91]. We start by constructing a computation tree $T_e(t)$ of depth t rooted at some variable y_e, where the root variable e is chosen so that it is in the true MAP assignment – i.e. $y_e^\star = 1$. To derive a contradiction, we assume that $b_e^t(0) \geq b_e^t(1)$, which implies that the MAP assignment $\boldsymbol{y}^{TMAP}$ on the computation tree has the root assignment $y_e^{TMAP} = 0$. We



The assignment $\boldsymbol{y}^{\mathrm{TMAP}}$ with $y_{24} = 1$
The assignment $\boldsymbol{y}^{\mathrm{NEW}}$

Figure 2.9: Computation tree for matching model in Figure 2.8.

use the assignment $\boldsymbol{y}^{TMAP}$ to construct a *new* assignment $\boldsymbol{y}^{NEW}$ by "flipping" some of the variables in $\boldsymbol{y}^{TMAP}$. We then compare the total weight of the new matching, $w'(\boldsymbol{y}^{NEW})$, to the total weight of the MAP matching on the computation tree, $w'(\boldsymbol{y}^{TMAP})$, and show that $w'(\boldsymbol{y}^{NEW}) > w'(\boldsymbol{y}^{TMAP})$. This contradicts the optimality of $\boldsymbol{y}^{TMAP}$ and proves that $b^t_e(0) < b^t_e(1)$ and $y^{BP}_e(t) = 1$.

Figure 2.9 illustrates the core contradiction. We start with a weighted graph G' and construct a computation tree to depth $t = 6$ rooted at some variable in the graph. We root the tree at edge $(2, 4)$. Edge $(2, 4)$ does not appear in the optimal matching $M^\star$, meaning that $y^\star_{24} = 0$. To derive a contradiction, we assume that $b^t_{24}(1) \geq b^t_{24}(0)$ so that the root variable $y^{TMAP}_{24} = 1$ appears in a MAP assignment on the computation tree. The left-hand tree illustrates $\boldsymbol{y}^{TMAP}$, the max weight assignment on the computation tree assuming that the root variable appears in the MAP assignment (variables in the MAP assignment are shown in green; variables not in the MAP are shown in red). The right-hand tree illustrates a new assignment $\boldsymbol{y}^{NEW}$ on the computation tree constructed from $\boldsymbol{y}^{TMAP}$ by flipping some of the variables in $\boldsymbol{y}^{TMAP}$. For example, the root variable is flipped from 1 to 0 indicated by the change in color from green to red. The total weight of the left-hand assignment is $w'(\boldsymbol{y}^{TMAP}) = 24$, while the total weight of the right-hand assignment is $w'(\boldsymbol{y}^{NEW}) = 26$. We thus have a contradiction as $\boldsymbol{y}^{TMAP}$ cannot be the actual MAP assignment on the computation tree. Thus, our assumption that $b_{24}^t(1) \geq b_{24}^t(0)$ is incorrect.

Of course, showing that $w'(\boldsymbol{y}^{NEW}) > w'(\boldsymbol{y}^{TMAP})$ in general is not trivial. The two main ingredients for proving this are: 1) establishing that the set of variables that are "flipped" in going from $\boldsymbol{y}^{TMAP}$ to $\boldsymbol{y}^{NEW}$ induces a path on the computation tree $T_e(t)$; and 2) using a perturbation argument to show that the total weight of the edges "flipped" from $0 \to 1$ along the path is greater than the total weight of the edges "flipped" from $1 \to 0$. Our proof strategy is thus similar to the one adopted in [75], but is complicated considerably by the incorporation of blossom factors.

2.4.5 A Cutting Plane BP Algorithm

The previous section established that max-product BP can be used to solve the max-weight matching problem when the matching LP relaxation, match-blossom-LP, is tight and only uses blossoms defined over non-intersecting odd-sized cycles. However, finding a collection of non-intersecting odd-sized cycles that tighten the match-blossom-LP remains a challenging task.

It was recently shown that the cutting plane method is polynomial time for the matching problem [14]. One would thus hope that the LP solver could be replaced by max-product BP in order to produce a distributed, polynomial time cutting plane algorithm for finding max weight matchings. Unfortunately, BP cannot be utilized in the procedure of [14] for a couple of reasons. First, their cut generation procedure requires solving an auxiliary LP to identify blossoms that should be added and removed. Second, and perhaps more important, the set of blossoms used to tighten match-blossom-LP are not guaranteed to be non-intersecting cycles. Due to these difficulties, we provide a heuristic cutting plane algorithm, which we call cutting-plane using BP (or CP-BP), that employs a heuristic to identify sets of odd-sized cycles to tighten the LP relaxation.

Algorithm 2.1 Cutting Plane BP (CP-BP) Algorithm

```
 1: Set $\mathcal{C} = \emptyset$
 2: Run max-product BP on the graphical model in (2.27) for $T$ iterations.
 3: For each edge $e \in E'$, set $y_e = \begin{cases} 1 & \text{if } b_e^T(1) > b_e^T(0) \text{ and } b_e^{T-1}(1) > b_e^{T-1}(0) \\ 0 & \text{if } b_e^T(1) < b_e^T(0) \text{ and } b_e^{T-1}(1) < b_e^{T-1}(0) \\ 1/2 & \text{otherwise} \end{cases}$.
 4: Compute $\boldsymbol{x} = (x_e)$ from $\boldsymbol{y} = (y_e)$ as per (2.28).
 5: if $\boldsymbol{x} \notin \{0, 1/2, 1\}^{|E|}$ then
 6:     Terminate.
 7: else if there is no edge $e \in E$ with $x_e = 1/2$ then
 8:     Return $\boldsymbol{x}$ and Terminate.
 9: else
10:     Find a non-intersecting, odd-sized cycle of half-integral edges $C = \{e : x_e = 1/2\}$
11:     if no such cycle exists (i.e. $C = \emptyset$) then
12:         Terminate.
13:     else
14:         Add $C$ to $\mathcal{C}$ and go to step 2
15:     end if
16: end if
```

The cutting plane BP algorithm is presented in Figure 2.1. We note that the cut recognition heuristic in Step 10 was also employed by [37, 85]. We also note that BP can be replaced by an off-the-shelf LP solver to obtain the LP optima $\boldsymbol{x}$ in Step 4. This results in a "traditional" cutting-plane LP method for the max-weight matching problem, which refer to as CP-LP. The reason why CP-BP terminates when $\boldsymbol{x} \notin \{0, 1/2, 1\}^{|E|}$ is because the solution $\boldsymbol{x}$ of match-blossom-LP with non-intersecting cycles is half-integral. In other words, $\boldsymbol{x} \notin \{0, 1/2, 1\}^{|E|}$ occurs when BP fails to find the solution to the current match-blossom-LP problem. A proof of $\frac{1}{2}$-integrality, which we did not find in the literature, can be found in the supplementary material of [81].

2.4.6 Experiments

The CP-BP procedure described in Algorithm 2.1 is a heuristic and it is not guaranteed to find the max weight matching of a general weighted matching problem for several reasons. First, it may be that match-blossom-LP requires overlapping and/or non-cyclic blossom constraints to be made tight for a given problem, in which case, CP-BP will terminate in Step 12. Second, it may be that our heuristic of finding cuts by searching for non-intersecting cycles on half-integral edges cannot be used to generate a violated cut, in which case, CP-BP will also terminate in Step 12. Last, BP is not guaranteed to find the LP optima when match-blossom-LP is not tight, in which case, CP-BP will terminate in Step 6.

We conducted a variety of experiments to gauge the effectiveness of using the CP-BP heuristic to solve weighted matching problems. We ran experiments on two types of synthetically generated problems: 1) Sparse Graph instances; and 2) Triangulation instances. The sparse graph instances were generated by forming a complete graph on $|V| = \{50, 100, 200\}$ nodes and independently eliminating edges with probability $p = 0.5$ or 0.9. Integral weights, drawn uniformly in $[1, 2^{20}]$, were assigned to the edges that remained. The Triangulation instances were generated by randomly placing $|V| = \{100, 200\}$ points in the $2^{20} \times 2^{20}$ square and computing a Delaunay triangulation on this set of points. Edge weights were set to the rounded Euclidean distance between two points. A set of 100 instances were generated for each setting of $|V|$ and CP-BP was run in all cases for $T = 100$ iterations.

We compared our CP-BP heuristic to a CP-LP heuristic that used an off-the-shelf, simplex based LP-solver rather than BP. This comparison allowed us to distinguish between those failures caused by being unable to locate a tightening cycle constraint and those failures due to the non-exactness of BP on non-tight relaxations. The results from our experiments are presented in Tables 2.1 and 2.2. Columns *# CP-BP* and *# CP-LP* indicate the percentage of instances in which the cutting plane methods found the maximum weight matching. The column *# Tight LPs* indicates the percentage of instances for which the bipartite relaxation was tight (i.e. match-blossom-LP with $\mathcal{C} = \emptyset$). The column *# Correct* indicates the number of correct matchings found by the solvers, while the column *# Converged* indicates the number of instances in which CP-BP converged upon termination, but we were unable to find a non-intersecting, odd-sized cycle using our heuristic. Finally, the *Time* column indicates the *mean [min,max]* run-times of the different solvers.

Table 2.1: Evaluation of CP-BP and CP-LP on Sparse graph instances.

50 % sparse graphs				90 % sparse graphs			
$\lvert V\rvert$ / $\lvert E\rvert$	# CP-BP	# Tight LPs	# CP-LP	$\lvert V\rvert$ / $\lvert E\rvert$	# CP-BP	# Tight LPs	# CP-LP
50 / 490	94 %	65 %	98 %	50 / 121	90 %	59 %	91 %
100 / 1963	92 %	48 %	95 %	100 / 476	63 %	50 %	63 %
200 / 7864	76 %	64 %	84 %	200 / 1902	87 %	53 %	93 %

Table 2.2: Evaluation of CP-BP and CP-LP on Triangulation instances.

	Triangulation, $\lvert V\rvert = 100$, $\lvert E\rvert = 285$		Triangulation, $\lvert V\rvert = 200$, $\lvert E\rvert = 583$	
Algorithm	# Correct / # Converged	Time (sec)	# Correct / # Converged	Time (sec)
CP-BP	33 / 36	0.2 [0.0,0.4]	11 / 12	0.9 [0.2,2.5]
CP-LP	34 / 100	0.1 [0.0,0.3]	15 / 100	0.8 [0.3,1.6]

These results seem to indicate a few things. First, CP-BP is nearly as good as CP-LP for solving the max weight matching problem – there is less than a 10% degradation in correctness in going from CP-LP to CP-BP. In addition, many of the failures are a result of being unable to find a tightening cut constraint $C \in \mathcal{C}$, rather than due to the inexactness of BP for non-tight relaxations (e.g. CP-BP is correct on 33 of the small triangulation instances, while CP-LP is correct on only 34 of those instances). Second, our graphical transformation allows us to use max-product BP to solve significantly more max weight matching problems than were provably solvable by BP prior to our result. This is clear by noting the increase from the *# Tight LPs* column to the *# CP-BP* column in Table 2.1.

Once again, our comparison between CP-BP and CP-LP was to distinguish between those failures caused by being unable to locate a tightening cycle constraint and those due to inexactness of BP on non-tight relaxations. As expected, CP-LP finds the optimal matching more frequently than CP-BP, but the difference between the algorithms is slight. We note, however, that the run-time complexity of CP-BP is $\mathcal{O}(c\ w^{\max}\ |V|^3))$ [75], which is smaller than the exponential time complexity of CP-LP [3]. In addition, CP-BP can be implemented as a parallel and distributed algorithm meaning that there are practical reasons to prefer CP-BP over CP-LP when solving matchings.

2.5 Discussion

We began this chapter by introducing the MAP-ILP problem, which is the MAP problem for a specific class of graphical model that encode an ILP. We then established conditions under which the LP relaxation to the ILP encoded by the graphical model is equivalent to the pairwise LP relaxation to the MAP problem, MAP-LP. This equivalence suggests that the MAP-LP relaxation can be tightened not only by adding constraints that tighten the outer bound on the marginal polytope, but also by adding constrains that directly tighten the LP relaxation to the original ILP. This finding has important implications for the design of cutting plane algorithms, as one may now be able to leverage existing cut generation heuristics and separation algorithms when stipulating a problem-specific MAP solver.

We then introduced the weighted matching problem and discussed a recent result due to Sanghavi et al. showing that max-product BP is exact when the bipartite relaxation to the matching problem is tight and unique[75]. Unfortunately, as shown in our experiments, the bipartite relaxation of the matching problem is often not tight. The bipartite relaxation can be made tight by introducing blossom constraints, but the addition of such constraints ruins the exactness results of Sanghavi et al. We thus introduced a “fix” to BP that overcomes this limitation by transforming the original match-MAP problem to a MAP problem on an auxiliary graphical model for which BP is provably convergent. We incorporated our “fixed”

[3]CP-LP has exponential worst case complexity as it employs the simplex method to solve LP relaxations to the matching problem.

BP algorithm into a cutting plane BP approach that performed well when used to find the maximum weight matching on several synthetic problem instances.

Several questions and directions for future research were exposed by the work in this chapter. One important open problem is: what are the properties common to convergent and correct max-product BP algorithms? We now are aware of several combinatorial optimization problems for which BP is exact, including matchings [5, 76, 44], perfect matchings [4], independent sets [77] and network flows[28]. Necessary conditions for exactness of BP in these problems seem to be uniqueness and tightness of the corresponding LP relaxation. However, these conditions are not sufficient and it appears that structural properties of the factor graph and of the factors themselves play an important role. It is thus worth exploring the underlying mechansim that our graphical transformation restored and, ultimately, identifying the core principles governing the exactness of BP.

Chapter 3

A Bottom-Up Approach to Constructing GBP Approximations

In this chapter, we turn our attention to the task of computing marginal probabilities. Sum-Product Belief Propagation (BP) is an algorithm originally invented by Pearl for performing inference in Bayesian networks[72]. BP is a simple, distributed algorithm that passes messages between the nodes of a graphical model. When the model is tree-structured, the BP algorithm requires just two iterations to compute the exact marginals of the input distribution, $p(\boldsymbol{y})$. When the network is not tree-structured, then BP is no longer exact. We can nonetheless apply the BP message-passing updates to the loopy model. The result is an iterative, approximate inference algorithm known as Loopy BP that has proven to be quite effective in practice (see e.g. Section 1.4.1).

Generalized Belief Propagation (GBP) is a broad class of approximate inference algorithms that extends BP by grouping the network's nodes into clusters and (iteratively) passing messages between the clusters of variables. The main motivation behind clustering the nodes of a model is that if we can convert a loopy graphical model into an equivalent tree-structured graphical model, then we can directly utilize the BP updates to compute exact marginals. In fact, this is precisely the aim of the Bucket Elimination[19] and Junction Tree[58] algorithms (see e.g. Section 1.4.1). They seek a re-parameterization of the model into factors on some cluster tree that are the marginals of the distribution.

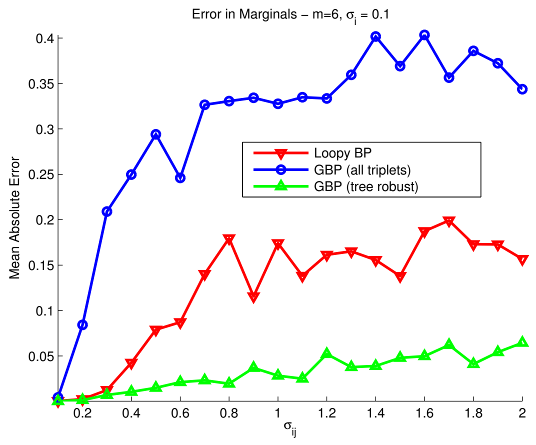
Unfortunately, we cannot always find an equivalent tree-structured graphical model for which the marginal computations are tractable. GBP relaxes the requirement of finding an equivalent tree-structured model and instead finds an equivalent model on a loopy, cluster graph. By grouping the variables into clusters we can find an equivalent model that is more tree-like than the model on which Loopy BP operates, but maintain tractability by limiting the size of the largest cluster considered. In fact, by raising or lowering the size of the largest cluster considered one can imagine a spectrum of message-passing algorithms: Loopy BP, which does no clustering, is at one end of the spectrum; Junction Tree (or Bucket Elimination), which finds an equivalent tree-structured model, is at the other end of the spectrum; and GBP approximations lie between these two extremes.

By passing messages between clusters, rather than the nodes of the model, GBP offers the promise to produce estimates that are far more accurate than Loopy BP without adding that much computation. However, GBP is quite difficult to apply in practice. While Loopy BP is a procedure that is fully specified given a graphical model, GBP requires first identifying a collection of clusters and then specifying the messages that will be passed between the clusters. Moreover, GBP is very sensitive to the choice of clusters used: a bad choice can lead to poor convergence and marginals estimates that are worse than Loopy BP, while a good choice can lead to rapid convergence and accurate estimates.

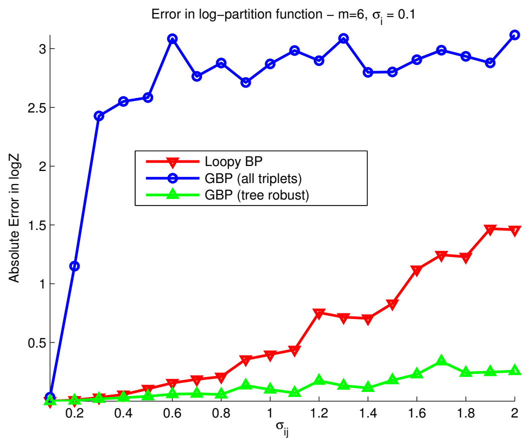
Figure 3.1 illustrates the sensitivity of GBP to cluster choice for a simple pairwise model on 6 variables. Figure 3.1a (3.1b) plots the error in the marginal estimates (log-partition function), where inference in the model becomes harder as we move from left to right[1].

[1]The reported errors are actually an average over 50 randomly generated models with unary interactions

The GBP approximation shown in blue utilizes all 20 clusters of size 3 in the model of 6 variables, while the GBP approximation in green utilizes a specific subset of 10 of these clusters. Clearly, utilizing all of the clusters is a bad choice and leads to estimates that are actually worse than Loopy BP's estimates shown in red. In contrast, the green line shows how a good choice of clusters can lead to far more accurate estimates than Loopy BP.



(a) Error in marginal estimates.



(b) Error in log-partition function estimates.

Figure 3.1: Error of GBP approximations defined using both good and bad clusters.

Our understanding of how to chose *good* clusters has been aided by the variational perspective of inference discussed in Section 1.4.2. The framework of variational inference converts computation of the log-partition function and marginals from a summation task to an optimization problem. Although this optimization problem is intractable, it suggests a way to create approximations by relaxing the problem's constraints and approximating the entropy term appearing in the problem's objective function. In fact, the collection of clusters defining a GBP approximation specify exactly how the problem is relaxed and the entropy is approximated.

This perspective has led to the development of several criteria guiding the specification of GBP algorithms. A common theme among the criteria is that the collection of clusters chosen yield accurate entropies for specific choices of input distribution (e.g. when the input

drawn from $\mathcal{N}(0, 0.1^2)$ and pairwise interactions drawn from $\mathcal{N}(0, \sigma_{ij}^2)$

distribution $p(\boldsymbol{y})$ is uniform). Our developments in this chapter build upon this existing criteria.

The main contributions of this chapter are:

1. We introduce a new criteria – tree-robustness – for choosing the collections of clusters defining a GBP approximation in a pairwise graphical model. Tree-robustness requires the entropy to be exact on every possible tree embedded in the model, where an embedded tree is obtained by ignoring factors on off-tree edges. We provide a theoretical justification of the tree-robustness criteria and demonstrate its effectiveness empirically.

2. We propose a method for automatically choosing collection of clusters satisfy tree-robustness and existing, "common-sense" criteria for Loop-Structured Region Graphs (SRGs) – a specific family of GBP approximations. Our construction procedure relates choosing clusters in a Loop-SRG to the task of finding Fundamental Cycle Bases in the underlying graph.

3.1 Two Perspectives of GBP

In Chapter 1, we introduced GBP from two different perspectives. The first was as a natural extension to Loopy BP, where messages were passed between the nodes of a cluster graph rather than the nodes of a factor graph. In this section, we will see that such message passing algorithms can be interpreted as finding alternate factorizations of an input distribution. The second perspective arose by making certain approximations to the variational optimization problem. In this section, we briefly review these two different motivations for GBP approximations, before unifying them via the region graph formalism in the following section.

3.1.1 GBP as a Re-parameterization Procedure

A graphical model defines a distribution as a product of factors, $p(\boldsymbol{y}) = \frac{1}{Z}\prod_{\alpha\in F}\psi_\alpha(\boldsymbol{y}_\alpha)$. The factorization of $p(\boldsymbol{y})$ is not unique, however, as the combination of many different factors can yield the same probability distribution. For example, consider the re-parameterization $\tilde{\psi}_\alpha(\boldsymbol{y}_\alpha) = c \cdot \psi_\alpha(\boldsymbol{y}_\alpha)$ for some positive constant c. It follows that $\tilde{p}(\boldsymbol{y}) = \frac{1}{\tilde{Z}}\prod_{\alpha\in F}\tilde{\psi}_\alpha(\boldsymbol{y}_\alpha) = p(\boldsymbol{y})$, where $\tilde{Z} = c^{|F|} \cdot Z$.

An important observation made in [88] is that when the BP algorithm is run on a tree-structured model, it produces an alternate factorization of the distribution $p(\boldsymbol{y})$ as a product of factors that correspond to the marginals of the model,

$$p(\boldsymbol{y}) = \prod_{i\in V} p(y_i) \prod_{\alpha\in F} \frac{p(\boldsymbol{y}_\alpha)}{\prod_{j\in\delta(\alpha)} p(y_j)}, \tag{3.1}$$

where $p(\boldsymbol{y}_\alpha)$ is the joint marginal over variables $\boldsymbol{y}_\alpha$ and $p(y_i)$ is the marginal of variable y_i. Moreover, this factorization into a product of marginals is unique in a tree-structured model.

When the model contains cycles, it is generally not possible to re-parameterize the distribution $p(\boldsymbol{y})$ into a quotient of a product of factor marginals in the numerator and a product of single variable marginals in the denominator as in (3.1). We can nonetheless apply the BP updates introduced in Section 1.4.1, resulting in the Loopy BP algorithm. Upon convergence, the Loopy BP algorithm will have found a re-parameterization of $p(\boldsymbol{y})$ of the form

$$p(\boldsymbol{y}) = \frac{1}{Z'} \prod_{i\in V} b(y_i) \prod_{\alpha\in F} \frac{b(\boldsymbol{y}_\alpha)}{\prod_{j\in\delta(\alpha)} b(y_j)}, \tag{3.2}$$

where $b(\boldsymbol{y}_\alpha)$ is an approximate joint marginal over variables $\boldsymbol{y}_\alpha$ and $b(y_i)$ is an approximate marginal of variable y_i. Note that the normalization constant in this new parameterization, Z', will generally not be equal to the normalization constant of our original factorization, $Z \neq$

Z'. In addition, $Z' \neq 1$ unless the collection of $b(\boldsymbol{y}_\alpha)$'s and $b(y_i)$'s define a joint distribution. Further, computation of Z' after re-parameterization is no easier than computing Z in our original model, which is why we simply take $b(\boldsymbol{y}_\alpha) \approx p(\boldsymbol{y}_\alpha)$ and $b(y_i) \approx p(y_i)$ as approximate marginals and avoid computation of Z'.

When the model is cyclic, it is always possible to group the model's nodes into a cluster tree (see Section 1.4.1 for a definition). Bucket Elimination[19] and Junction Tree [58] are algorithms that pass BP messages on a cluster tree $CT = (V, E)$ to compute marginals on pairs of neighboring clusters. By doing so, they produce a factorization of $p(\boldsymbol{y})$ of the form

$$p(\boldsymbol{y}) = \frac{\prod_{i \in V} p(\boldsymbol{y}_i)}{\prod_{(i,j) \in E} p(\boldsymbol{y}_{ij})}, \tag{3.3}$$

where $p(\boldsymbol{y}_i)$ and $p(\boldsymbol{y}_{ij})$ are cluster marginals and separator marginals, respectively, defined over subsets of variables, $\boldsymbol{y}_i \subseteq \boldsymbol{y}$, where $\boldsymbol{y}_{ij} = \boldsymbol{y}_i \cap \boldsymbol{y}_j$ for each edge $(i, j) \in E$.

In many problems of interest, it is not possible to efficiently compute the cluster marginals that define $p(\boldsymbol{y})$ as an equivalent tree-structured model. We can, however, take the clusters in our cluster-tree that are too large to perform computations on and break them up into computationally tractable sub-clusters. If we add separators between each of these sub-clusters, the result is a collection of clusters and separators forming a loopy, cluster graph, $CG = (V, E)$. If we apply the BP updates on this cluster graph, the result is a generalized form of the Loopy BP algorithm, or GBP. Upon convergence, the GBP algorithm will have found an alternate parameterization of $p(\boldsymbol{y})$ of the form

$$p(\boldsymbol{y}) = \frac{1}{Z^\dagger} \frac{\prod_{i \in V} b(\boldsymbol{y}_i)}{\prod_{(i,j) \in E} b(\boldsymbol{y}_{ij})}, \tag{3.4}$$

where $b(\boldsymbol{y}_i)$ and $b(\boldsymbol{y}_{ij})$ are approximate cluster and separator marginals. As in (3.2), the normalization term $Z^\dagger \neq Z$ and is generally not equal to 1 either. This is precisely the type of re-parameterizations sought by GBP algorithms, such as the Junction-Graph algorithm[2] and Iterative Join Graph Propagation (IJGP)[20], that pass messages on a cluster graph.

3.1.2 GBP as a Variational Approximation

Recall from Section 1.4.2 that variational inference converts the problem of computing the log-partition function to an optimization problem [90]. In particular, given a model $p(\boldsymbol{y}) = \exp\left(\sum_\alpha \theta_\alpha(\boldsymbol{y}_\alpha) - \log Z(\boldsymbol{\theta})\right)$, with $\theta_\alpha(\boldsymbol{y}_\alpha) = \log \psi_\alpha(\boldsymbol{y}_\alpha)$, the log-partition function is the solution to the optimization problem

$$\log Z(\boldsymbol{\theta}) = \max_{\boldsymbol{\mu}\in\mathcal{M}} \left[\boldsymbol{\mu}\cdot\boldsymbol{\theta} + H(\boldsymbol{\mu})\right], \tag{3.5}$$

where $\mathcal{M} = \{\boldsymbol{\mu}' \in \mathbb{R}^d \mid \exists\boldsymbol{\theta} \text{ s.t. } \boldsymbol{\mu}' = \boldsymbol{\mu}(\boldsymbol{\theta})\}$ is the marginal polytope and $H(\boldsymbol{\mu})$ is the entropy computed using the distribution $p(\boldsymbol{y};\boldsymbol{\theta}(\boldsymbol{\mu}))$ that results from finding parameters $\boldsymbol{\theta}$ that yield the mean vector $\boldsymbol{\mu}$. Unfortunately, the marginal polytope $\mathcal{M}$ is difficult to characterize and the entropy is intractable in general.

GBP approximates both $\mathcal{M}$ and $H(\boldsymbol{\mu})$ using a collection of regions $\mathcal{R}$, where a region (or cluster) $\gamma \in \mathcal{R}$ is simply a subset of variables $\boldsymbol{y}_\gamma \subseteq \boldsymbol{y}$ [2]. GBP approximates the true entropy through a combination of marginal entropies

$$H(\boldsymbol{\mu}) \approx \tilde{H}^{\text{GBP}}(\boldsymbol{\mu}) = -\sum_{\gamma\in\mathcal{R}} c_\gamma \sum_{\boldsymbol{y}_\gamma} \mu(\boldsymbol{y}_\gamma;\boldsymbol{\theta}) \log \mu(\boldsymbol{y}_\gamma;\boldsymbol{\theta}), \tag{3.6}$$

where $c_\gamma \in \mathbb{R}$ is the over-counting number for region γ and $\mu(\boldsymbol{y}_\gamma;\boldsymbol{\theta}) = p(\boldsymbol{y}_\gamma;\theta(\boldsymbol{\mu}))$ is the marginal probability of $\boldsymbol{y}_\gamma$.

GBP approximates the marginal polytope, $\mathcal{M}$, via a collection of local consistency con-

[2]Once again, we assume that each factor $\alpha \in F$ can be assigned to some region $\gamma \in R$.

straints,

$$\mathcal{M}_{\text{GBP}} = \left\{ \boldsymbol{\mu} \geq 0 \;\middle|\; \begin{array}{lll} \text{Normalization :} & \sum_{\boldsymbol{y}_\gamma} \mu_\gamma(\boldsymbol{y}_\gamma) = 1, & \forall \gamma \in \mathcal{R} \\ \text{Consistency :} & \sum\limits_{\boldsymbol{y}_\gamma \setminus \boldsymbol{y}_\beta} \mu_\gamma(\boldsymbol{y}_\gamma) = \mu_\beta(\boldsymbol{y}_\beta), & \forall \gamma, \beta \in \mathcal{R}, \beta \subset \gamma, \boldsymbol{y}_\beta \end{array} \right\}, \quad (3.7)$$

where $\beta \subset \gamma$ means that $\boldsymbol{y}_\beta \subset \boldsymbol{y}_\gamma$. Since, $\mathcal{M}_{\text{GBP}} \supseteq \mathcal{M}$, there exist vectors $\boldsymbol{\mu} \in \mathcal{M}_{\text{GBP}}$ that could not have come from any joint distribution $p(\boldsymbol{y}; \boldsymbol{\theta}(\boldsymbol{\mu}))$, which is why the vector of marginals $\boldsymbol{\mu}$ are referred to as beliefs or approximate marginals.

Now, let us make the connection between the re-parameterization perspective of GBP and the variational perspective more concrete. Consider the GBP approximation described by the re-parameterization in (3.4) and make the following correspondence:

$$\boldsymbol{\theta} \leftrightarrow \{\log \psi_\alpha(\boldsymbol{y}_\alpha) \mid \alpha \in F, \boldsymbol{y}_\alpha\} \quad (3.8)$$

$$\boldsymbol{\mu} \leftrightarrow \{b(\boldsymbol{y}_i) \mid i \in V, \boldsymbol{y}_i\} \cup \{b(\boldsymbol{y}_{ij}) \mid (i,j) \in E, \boldsymbol{y}_{ij}\} \quad (3.9)$$

$$\tilde{H}^{\text{GBP}}(\boldsymbol{\mu}) \leftrightarrow -\sum_{i \in V} \sum_{\boldsymbol{y}_i} b(\boldsymbol{y}_i) \log b(\boldsymbol{y}_i) + \sum_{(i,j) \in E} \sum_{\boldsymbol{y}_{ij}} b(\boldsymbol{y}_{ij}) \log b(\boldsymbol{y}_{ij}) \quad (3.10)$$

$$\mathcal{M}_{\text{GBP}} \leftrightarrow \left\{ \begin{array}{ll} \sum_{\boldsymbol{y}_i} b(\boldsymbol{y}_i) = 1 & \forall i \in V \\ \sum_{\boldsymbol{y}_i \setminus \boldsymbol{y}_{ij}} b(\boldsymbol{y}_i) = b(\boldsymbol{y}_{ij}) & \forall (i,j) \in E, \boldsymbol{y}_i j \\ b(\boldsymbol{y}_i) \geq 0 & \forall i \in V, \boldsymbol{y}_i \end{array} \right\}. \quad (3.11)$$

As a result, the optimization problem $\max\limits_{\boldsymbol{\mu} \in \mathcal{M}_{\text{GBP}}} \left[\boldsymbol{\mu} \cdot \boldsymbol{\theta} + \tilde{H}^{\text{GBP}}(\boldsymbol{\mu}) \right]$ is equivalent to

$$\max_{\boldsymbol{b}} \sum_{\alpha \in F} \sum_{\boldsymbol{y}_\alpha} b(\boldsymbol{y}_\alpha) \log \psi_\alpha(\boldsymbol{y}_\alpha) - \sum_{i \in V} \sum_{\boldsymbol{y}_i} b(\boldsymbol{y}_i) \log b(\boldsymbol{y}_i) + \sum_{(i,j) \in E} \sum_{\boldsymbol{y}_{ij}} b(\boldsymbol{y}_{ij}) \log b(\boldsymbol{y}_{ij}), \quad (3.12)$$

subject to the constraints in (3.11).

3.2 Region Graphs and Structured Region Graphs

In Section 1.4.2, we introduced a data structure known as a region graph that served two main purposes. First, it helped visualize the collection of regions $\mathcal{R}$ defining a GBP approximation, which provided a clear distinction between the Cluster Variation Method (CVM) constructions[52], Junction-Graph constructions[2] and Join Graph constructions[62]. Second, it helped to organize message-passing computations, much in the way that a factor graph helps to organize the BP computations. In this section, we will see that the region graph structure can be expanded to include a specification of how each region is factored. This provides an additional level of flexibility when constructing a GBP approximation, which will ultimately be exploited in Section 3.4.

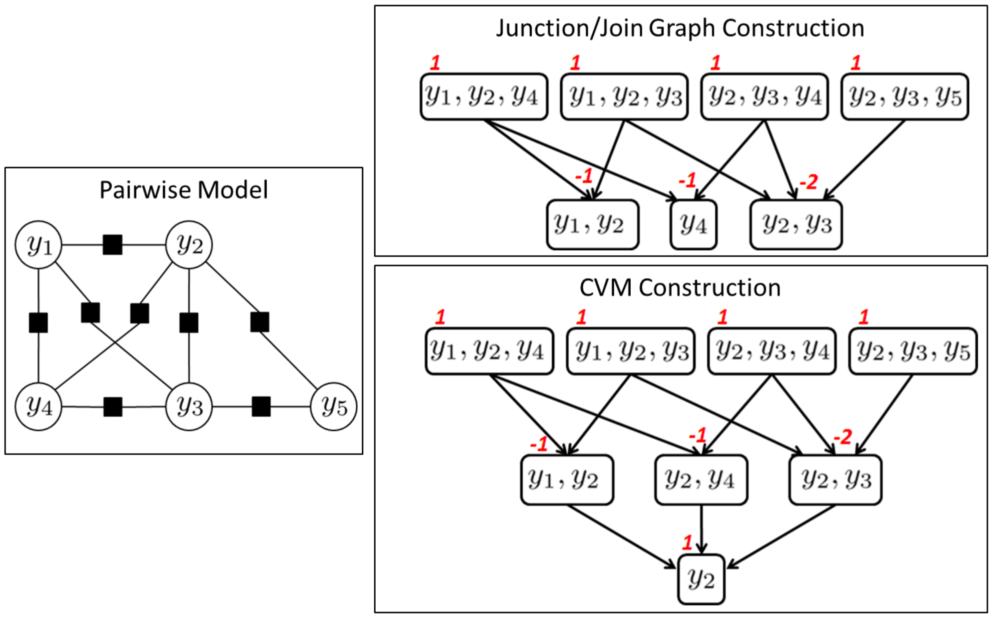


Figure 3.2: Illustration of two different Region Graph constructions.

We begin this section by reviewing the definition of a region graph, before discussing its extension to structured region graphs. Given a collection of regions $\mathcal{R}$, a *Region Graph* is simply a directed acyclic graph whose nodes correspond to regions in $\mathcal{R}$ and an edge is drawn from region γ to region β if region γ *covers* region β. By cover, we mean that $\boldsymbol{y}_\gamma \supset \boldsymbol{y}_\beta$ and there exists no $\tau \in \mathcal{R}$ such that $\boldsymbol{y}_\gamma \supset \boldsymbol{y}_\tau \supset \boldsymbol{y}_\beta$. We associate an over-counting number, c_γ with each region. We also assume that each factor $\alpha \in F$ is assigned to an outer region $\gamma \in \mathcal{O}$, meaning that $\boldsymbol{y}_\alpha \subseteq \boldsymbol{y}_\gamma$ for each $\alpha \in F$. Let $\psi(\gamma)$ denote the collection of factors assigned to outer region γ.

Figure 3.2 contains two different region graph constructions for the pairwise model on 5 variables shown. Both region graphs contain the same set of *outer* regions, which are those regions with no parents. However, the *inner* regions are chosen using different construction procedures – namely, the cluster variation method (CVM) and the Junction/Join Graph method introduced in Section 1.4.2. Note that in the CVM region graph, no edge is drawn from region $\gamma = \{y_1, y_2, y_4\}$ to region $\beta = \{y_2\}$ because there exists a region $\tau = \{y_2, y_4\}$. In other words, region γ does not *cover* region β. In addition, the over-counting numbers of each region are depicted in red above each region and are determined by the recursion in equation (1.47). Finally, we can assign the factors defining the pairwise model to each outer region. For example, for region $\gamma = \{y_1, y_2, y_4\}$, we let $\psi(\gamma) = \{\psi_{12}, \psi_{14}, \psi_{24}\}$.

The region graph's structure makes explicit the set of local consistency constraints defining the approximation, $\mathcal{M}_{\text{GBP}}$, to the marginal polytope. In particular, for every region γ we have that

$$\sum_{\boldsymbol{y}_\gamma \setminus \boldsymbol{y}_\beta} b_\gamma(\boldsymbol{y}_\gamma) = b_\beta(\boldsymbol{y}_\beta),\ \forall \beta \in ch(\gamma),\ \boldsymbol{y}_\beta, \tag{3.13}$$

where $ch(\gamma)$ is the set of children of region γ in the region graph. In other words, the marginal of each parent region's belief must agree with the belief of its children.

The region graph's structure also helps organize the computations needed to optimize the variational optimization problem. As discussed in Section 1.4.2, if the marginal of parent region γ is not consistent with the belief of some child $\beta \in ch(\gamma)$, we can make the child belief consistent by defining a correction message

$$m_{\gamma\to\beta}(\boldsymbol{y}_\beta) = \frac{\sum_{\boldsymbol{y}_\gamma \setminus \boldsymbol{y}_\beta} b_\gamma(\boldsymbol{y}_\gamma)}{b_\beta(\boldsymbol{y}_\beta)} \tag{3.14}$$

and updating the region beliefs as

$$b_\tau(\boldsymbol{y}_\tau) \propto b_\tau(\boldsymbol{y}_\tau) \cdot m_{\gamma\to\beta}(\boldsymbol{y}_\beta)^\alpha, \tag{3.15}$$

for all regions $\tau \in \Delta(\beta) \setminus \Delta(\gamma)$, where $\Delta(i) = \{j \in \mathcal{R} \mid \boldsymbol{y}_j \supseteq \boldsymbol{y}_i\}$ is the set of ancestors of region i in the region graph and region i itself. In these updates, α is a *damping factor* that controls how greedy of an update is made. If $\alpha = 1$ we make a full update and the marginal of parent region γ will be consistent with the belief of child region β, while if $\alpha < 1$ we make partial update. For example, consider the message $m_{\gamma\to\beta}(y_2)$ sent from inner region $\gamma = \{y_1, y_2\}$ to inner region $\beta = \{y_2\}$ in the CVM region graph in Figure 3.2. This message is used to update the belief $b_\beta(y_2)$ at region β, but it is also used to update the beliefs of the following regions as well: $\tau \in \{\{y_2, y_4\}, \{y_2, y_3\}, \{y_2, y_3, y_4\}, \{y_2, y_3, y_5\}\}$.

A *Structured Region Graph* (SRG) is a region graph in which each region γ is also associated with a set of cliques $\mathcal{C}(\gamma)$. Every variable in $\boldsymbol{x}_\gamma$ must appear in some clique or factor. The set of factors, $\psi(\gamma)$, and cliques, $\mathcal{C}(\gamma)$, associated with a region define a *structure* $G(\gamma)$, which is an undirected graph with vertices for each variable in $\boldsymbol{x}_\gamma$ and edges connecting any pair of variables appearing in the same factor or clique. The structure of a region, $G(\gamma)$, defines how a region factors. In particular, the belief of each region γ is represented by a local distribution of the form

$$b_\gamma(\boldsymbol{y}_\gamma) \propto \prod_{\alpha \in \psi(\gamma)} \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{C \in \mathcal{C}(\gamma)} f_C(\boldsymbol{y}_C), \tag{3.16}$$

where $\psi_\alpha(\boldsymbol{y}_\alpha)$ is an original factor and $f_C(\boldsymbol{y}_C)$ is a factor associated with clique C.

The hierarchy conditions of an SRG are different than a regular region graph. In a region graph, an edge is drawn from some region γ to some region β if region γ *covers* region β, meaning that $\boldsymbol{y}_\gamma \supset \boldsymbol{y}_\beta$ and there exists no $\tau \in \mathcal{R}$ such that $\boldsymbol{y}_\gamma \supset \boldsymbol{y}_\tau \supset \boldsymbol{y}_\beta$. In an SRG, an edge is drawn from region γ to region β if the structure of region β is a sub-graph of the structure of region γ, $G(\beta) \subset G(\gamma)$. In other words, every clique in some child region β appears in its parent regions $\gamma \in pa(\beta)$.

Figure 3.3 illustrates both a region graph and an SRG for the same pairwise model on a 3×3 grid. Notice that the outer regions in the SRG are loop structured, where the set of original factors, ψ_α, assigned to each region are denoted with a square and the set of cliques, ψ_C associated with each region are drawn without the square. For example, the graph structure, $G(\gamma)$, for region $\boldsymbol{y}_\gamma = \{y_1, y_2, y_4, y_5\}$ is a single loop which indicates that the belief over region γ factors as

$$b_\gamma(y_1, y_2, y_4, y_5) \propto \psi(y_1, y_2)\psi(y_1, y_4)\psi(y_2, y_5)\psi(y_4, y_5). \tag{3.17}$$

In contrast, the graph structure for region $\boldsymbol{y}_{\tilde{\gamma}} = \{y_5, y_6, y_8, y_9\}$ is also a single loop. However, the lack of squares on two of the edges indicate that the belief over region $\tilde{\gamma}$ factors as

$$b_\gamma(y_5, y_6, y_8, y_9) \propto f(y_5, y_6)f(y_5, y_8)\psi(y_6, y_9)\psi(y_8, y_9). \tag{3.18}$$

The loop structure informs us that we need only $4 \cdot K^2$ values to represent the joint belief of these outer region, as opposed to the K^4 values that would be required if the structure were a complete graph. The structure of a region can also be exploited when computing the message updates in (3.14). In particular, we can exploit the loop structure of outer region $\boldsymbol{y}_\gamma = \{y_1, y_2, y_4, y_5\}$ to efficiently compute the message $m_{\gamma \to \beta}$ to the child edge region $\boldsymbol{y}_\beta = \{y_2, y_5\}$ in $\mathcal{O}(K^3)$ time. While exploiting a region's structure may not be significant in this example, one can imagine a collection of loop-structured outer regions where the

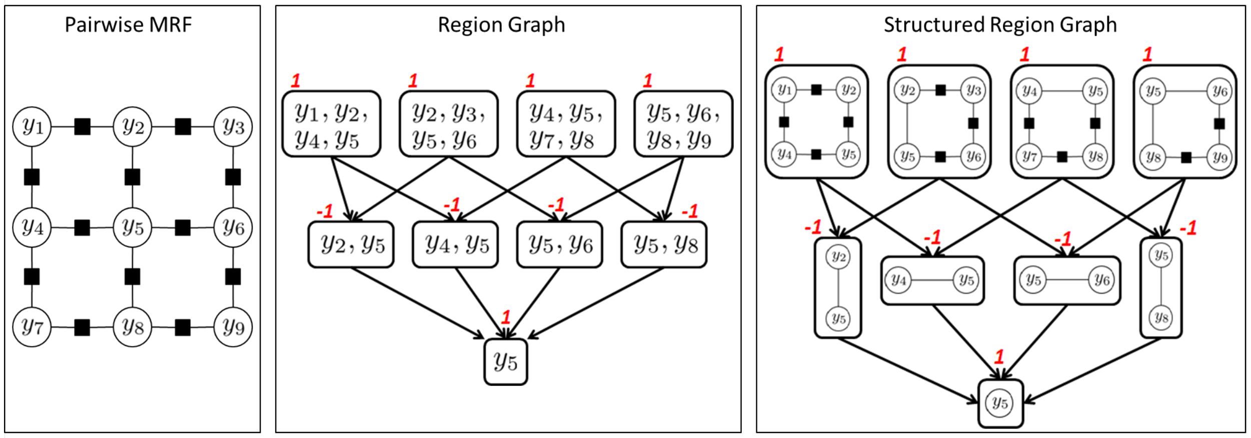


Figure 3.3: Illustration of a Region Graph and Structured Region Graph on a 3×3 grid.

computational savings would be quite substantial. Of course the computational savings comes at the cost of making many additional independence assumptions that are not satisfied by the model. As a result, if the regions and their structures are not chosen carefully, the accuracy of our estimates may degrade.

In addition to making the factorization of a region explicit, the graph structure of a region also allows us to define a richer set of local consistency constraints than the parent-child constraints considered in 3.13. In particular, the beliefs at a parent region γ in an SRG must satisfy

$$\sum_{\boldsymbol{y}_\gamma \setminus \boldsymbol{y}_C} b(\boldsymbol{y}_\gamma) = \sum_{\boldsymbol{y}_\beta \setminus \boldsymbol{y}_C} b(\boldsymbol{y}_\beta), \ \forall \beta \in ch(\gamma), \ C \in \mathcal{C}(\beta), \boldsymbol{y}_C. \tag{3.19}$$

In other words, the marginals of each parent region's belief must agree with the belief of each of its children's cliques. This is why we required the graph structure of a parent region to be a super-graph of its child regions in the definition of an SRG. As an illustration, consider the collection of structured regions in Figure 3.4. Notice that the child edge inner region $\boldsymbol{y}_\beta = \{y_2, y_5\}$ factors into independent marginals $b_\beta(\boldsymbol{y}_\beta) = b_\beta(y_2)b_\beta(y_5)$. As a result, we require the belief of the loop-structured region $\boldsymbol{y}_{\gamma_1} = \{y_1, y_2, y_4, y_5\}$ to satisfy the consistency constraints: $\sum_{y_1,y_4,y_5} b_{\gamma_1}(\boldsymbol{y}_{\gamma_1}) = b_\beta(y_2)$ and $\sum_{y_1,y_2,y_4} b_{\gamma_1}(\boldsymbol{y}_{\gamma_1}) = b_\beta(y_5)$. In addition, the region $\boldsymbol{y}_{\gamma_2} = \{y_2, y_3, y_5, y_6\}$ factors as $b_{\gamma_2}(\boldsymbol{y}_{\gamma_2}) = b_{\gamma_2}(y_2, y_3)b_{\gamma_2}(y_5, y_6)$ and must satisfy the consistency constraints: $\sum_{y_3} b_{\gamma_2}(y_2, y_3) = b_\beta(y_2)$ and $\sum_{y_6} b_{\gamma_2}(y_5, y_6) = b_\beta(y_5)$.

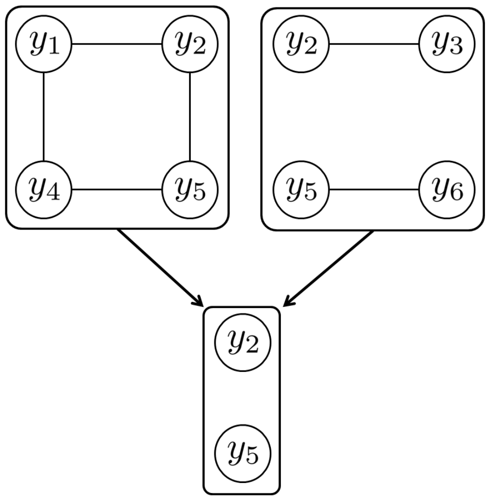


Figure 3.4: Illustration of consistency between structured regions.

In the remainder of this chapter, we will restrict our attention to Loop-SRGs, which are a particular type of SRG, defined as follows.

Definition 3.1. *A* Loop-SRG *is a 3-level SRG consisting of loop outer regions and edge and node inner regions, where a loop outer region has a structure $G(\gamma)$ that forms an (elementary) cycle. Loop outer regions are connected to the set of edge inner regions comprising the loop and edge inner regions are connected to the two node regions comprising the edge.*

The SRG in Figure 3.3 is an example of a Loop-SRG.

3.3 Region Selection Criteria

There is a considerable amount of flexibility in choosing the collection of regions $\mathcal{R}$ defining a GBP approximation. And, as illustrated in Figure 3.1, the choice of regions used has a profound effect on the quality of the estimates produced by GBP. In this section we review some criteria for choosing collections of regions found in the literature, before introducing our new criteria, tree-robustness.

The GBP approximation to the exact variational problem introduces two approximations. First, we relax the constraint that our beliefs correspond to marginals of some joint distribution, i.e. $\boldsymbol{\mu} = \{b_\gamma \boldsymbol{y}_\gamma)\} \in \mathcal{M}_{\text{GBP}} \supseteq \mathcal{M}$. Second, we approximate the true entropy $\tilde{H}^{\text{GBP}}(\boldsymbol{\mu}) \approx H(\boldsymbol{\mu})$. The criteria that follow try to ensure that both of these approximations are accurate. In particular, the *connectedness* criteria considers the quality of the marginal polytope approximation, while the *balance*, *MaxEnt-Normality*, *Over-Counting Number Unity* and *Tree-Robustness* criteria consider the quality of the entropy approximation.

We begin with a list of criteria compiled from [102, 94, 71, 95].

1. <u>Connectedness</u> - Let $RG = (V, E)$ be a region graph, where each vertex $\gamma \in V$ corresponds to a region $\gamma \in \mathcal{R}$. Let $\mathcal{R}(y_i) = \{\gamma \in \mathcal{R} \mid y_i \in \boldsymbol{y}_\gamma\}$ be the collection of regions containing variable y_i. A region graph is 1-*connected* if the sub-graph consisting of regions $\mathcal{R}(y_i)$ is connected for all $y_i \in \boldsymbol{y}$. Since an edge $(\gamma, \beta) \in E$ from region γ to region β in a region graph ensures parent-child consistency in (3.13), 1-connectedness ensures that every region will have consistent single variable marginals. This would of course be true if the collection of beliefs $\{b_\gamma(\boldsymbol{y}_\gamma)\}$ were the marginals of some actual distribution, i.e. $b_\gamma(\boldsymbol{y}_\gamma) = p(\boldsymbol{y}_\gamma)$.

   We can strengthen the notion of 1-connectedness by considering larger sets of variables. In particular, let $\boldsymbol{y}_S \subset \boldsymbol{y}$ be some subset of variables and let $\mathcal{R}(\boldsymbol{y}_S) = \{\gamma \in \mathcal{R} \mid \boldsymbol{y}_S \subseteq \boldsymbol{y}_\gamma\}$ be the collection of regions containing variables $\boldsymbol{y}_S$. Clearly we would like $\mathcal{R}(\boldsymbol{y}_S)$ to be connected to ensure that the collection of beliefs $\{b_\gamma(\boldsymbol{y}_\gamma)\}$ will have consistent marginals on $\boldsymbol{y}_S$.

   Finally, we say that a region graph, is *totally-connected* if it is connected for all sets of variables that are subsets of some outer region, i.e. for any $\boldsymbol{y}_S \subseteq \boldsymbol{y}_\gamma$ where $\gamma \in \mathcal{O}$ [71]. Intuitively, total-connectedness ensures that the collection of beliefs $\{b_\gamma(\boldsymbol{y}_\gamma)\}$ will be consistent on any subset of their variables. Of course, total-connectedness will not guarantee that $\boldsymbol{\mu} = \{b_\gamma(\boldsymbol{y}_\gamma)\} \in \mathcal{M}$, but in practice it results in finding a collection of beliefs that are more like actual marginals than non-totally-connected region graphs. We also note that the CVM method[52] produces a collection of clusters $\mathcal{R}$ that are totally-connected[71].

2. Balance - Once again, let $\mathcal{R}(y_i) = \{\gamma \in \mathcal{R} \mid y_i \in \boldsymbol{y}_\gamma\}$ be the collection of regions containing variable y_i. A region graph is 1-*balanced* if:

$$\sum_{\gamma \in \mathcal{R}(y_i)} c_\gamma = 1 \;\; \forall y_i \in \boldsymbol{y}. \tag{3.20}$$

A motivation for the 1-balance criteria comes from [102]. Let the true joint distribution $p(\boldsymbol{y})$ take each state $\boldsymbol{y} \in \{0, ..., K-1\}^m$ with equal probability and assume that the region beliefs $\{b_\gamma(\boldsymbol{y}_\gamma)\}$ are equal to the exact marginal probabilities $\{p(\boldsymbol{y}_\gamma)\}$. Then a collection of regions that are 1-balanced will return the exact entropy, $H_{\text{exact}} = -\log \frac{1}{K^m} = m \log K$, because

$$\tilde{H}^{\text{GBP}} = -\sum_{\gamma \in \mathcal{R}} c_\gamma \sum_{\boldsymbol{y}_\gamma} b_\gamma(\boldsymbol{y}_\gamma) \log b_\gamma(\boldsymbol{y}_\gamma) = -\sum_{\gamma \in \mathcal{R}} c_\gamma \log \prod_{i \in \gamma} \frac{1}{K} \tag{3.21}$$

$$= \sum_{\gamma \in \mathcal{R}} \sum_{i \in \gamma} c_\gamma \log K \tag{3.22}$$

$$= \sum_{i=1}^{m} \sum_{\gamma \in R(y_i)} c_\gamma \log K \tag{3.23}$$

We can extend the notion of 1-balance to larger sets of variables. In particular, let $\mathcal{R}(\boldsymbol{y}_S) = \{\gamma \in \mathcal{R} \mid \boldsymbol{y}_S \subseteq \boldsymbol{y}_\gamma\}$ be the collection of regions containing some subset of variables $\boldsymbol{y}_S \subset \boldsymbol{y}$. A region graph is balanced with respect to the set $\boldsymbol{y}_S$ if $\sum_{\gamma \in \mathcal{R}(\boldsymbol{y}_S)} c_\gamma = 1$. We say that a region graph, is *totally-balanced* if it is balanced for all sets of variables that are subsets of some outer region[71]. We note that the CVM method[52] produces a collection of clusters $\mathcal{R}$ that are totally balanced[71].

3. <u>MaxEnt-Normality and non-Singularity</u> - In the variational optimization problem, we seek a vector of beliefs $\boldsymbol{\mu} = \{b_\gamma(\boldsymbol{y}_\gamma)\}$ that maximize $\max_{\boldsymbol{\mu}\in\mathcal{M}_{\text{GBP}}} \left[\boldsymbol{\mu}\cdot\boldsymbol{\theta} + \tilde{H}^{\text{GBP}}(\boldsymbol{\mu})\right]$. Since we are maximizing the entropy $\tilde{H}^{\text{GBP}}(\boldsymbol{\mu})$, under the constraint that $\boldsymbol{\mu}\in\mathcal{M}_{\text{GBP}}$, it seems reasonable to focus our attention on ensuring that our choice of entropy approximation $\tilde{H}^{\text{GBP}}$ behaves like the exact entropy near its maximum. We know that a true joint distribution achieves maximum entropy when all of its states are equi-probable, i.e. $p(\boldsymbol{y}) = \frac{1}{K^m}$. As a result, it seems reasonable to require that our approximate entropy also achieve its maximum when all of the beliefs $\boldsymbol{\mu} = \{b_\gamma(\boldsymbol{y}_\gamma)\}$ are uniform.

   More formally, a GBP approximation is considered to be *MaxEnt-Normal* if the entropy approximation $\tilde{H}^{\text{GBP}}(\boldsymbol{\mu})$ is maximized by a collection of uniform beliefs $\boldsymbol{\mu} = \{b_\gamma(\boldsymbol{y}_\gamma)\}$ [102]. The entropy approximation for the all-triplets GBP approximation used in Figure 3.1 is not MaxEnt-Normal, as we now shall see. Figure 3.5 illustrates the all-triplets region graph used in the poor performing GBP approximation in Figure 3.1.

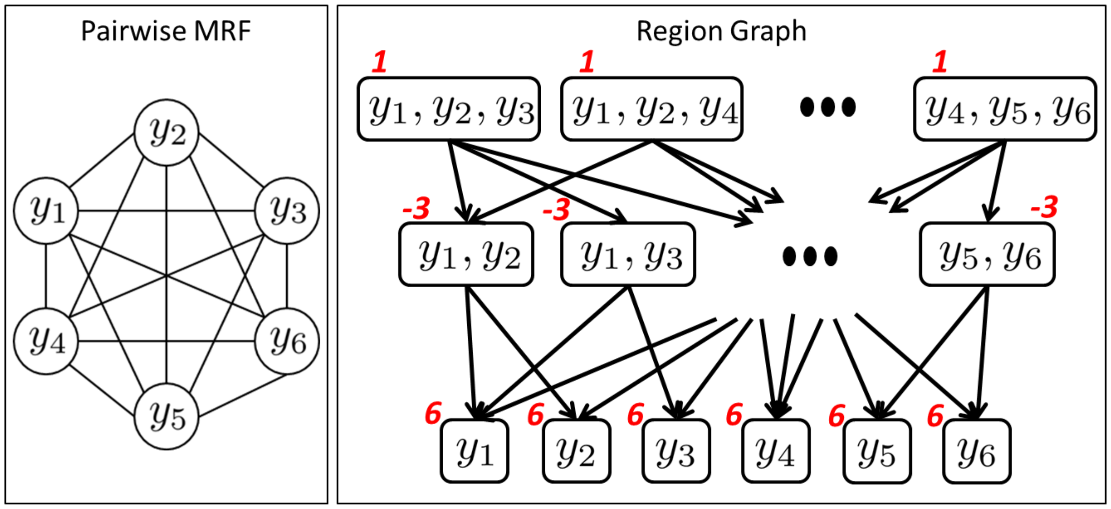


Figure 3.5: Illustration of the all-triplets region graph for a complete pairwise model on 6 nodes.

Assume that the beliefs of all regions $\{b_\gamma(\boldsymbol{y}_\gamma)\}$ are uniform. There are total of 20 outer triplet regions, each with over-counting number 1; a total of 15 inner edge regions, each with over-counting number -3; and a total of 6 inner node regions, each with over-counting number 6. Assuming that all region beliefs are uniform, the entropy of each triplet region is $H_{\text{triplet}} = 3\log K$, the entropy of each edge region is $H_{\text{edge}} = 2\log K$ and the entropy of each node region is $H_{\text{node}} = \log K$. Thus, when all region beliefs are uniform, the total entropy under our approximation is $20H_{\text{triplet}} - 45 \cdot H_{\text{edge}} + 36 \cdot H_{\text{node}} = 6\log K$.

Now consider a situation where our model $p(\boldsymbol{y})$ takes just two states with equal probability, e.g. the all-zeros state and the all-ones state are equi-probable. Assume that our beliefs $\{b_\gamma(\boldsymbol{y}_\gamma)\}$ correspond to the exact marginals under this model. In this case, each triplet region, edge region and node region will have entropy $\log 2$. Plugging, into the expression for the total approximate entropy, gives $20H_{\text{triplet}} - 45 \cdot H_{\text{edge}} + 36 \cdot H_{\text{node}} = 11\log 2$. As a result, for a model with binary variables ($K = 2$), the all-triplets approximate entropy does not achieve its maximum when the beliefs are uniform. Thus, the all-triplets GBP approximation is <u>not</u> MaxEnt-Normal, which may partly explain its poor performance in Figure 3.1.

Another way to think about MaxEnt-Normality is as follows. Imagine, that our distribution $p(\boldsymbol{y}) \propto \prod_{\alpha \in F} \psi_\alpha(\boldsymbol{y}_\alpha)$ is uniform. This can occur, for example, if $\psi_\alpha(\boldsymbol{y}_\alpha) = 1$ for all configurations $\boldsymbol{y}_\alpha$ and all factors $\alpha \in F$. However, any uniform setting of the factors will suffice to make $p(\boldsymbol{y})$ uniform. When all of the factors are uniform, then MaxEnt-Normality ensures that the beliefs that optimize $\max_{\boldsymbol{\mu} \in \mathcal{M}_{\text{GBP}}} \left[\boldsymbol{\mu} \cdot \boldsymbol{\theta} + \tilde{H}^{\text{GBP}}(\boldsymbol{\mu})\right]$ are uniform as well.

Non-Singularity is a criterion that goes beyond requiring the entropy approximation to take its maximum when all beliefs are uniform and thinks about the message-passing procedure used to find a set of optimizing beliefs, in the special case when $p(\boldsymbol{y})$ is uniform. In particular, since we optimize the constrained variational objective using the updates in (3.14) and (3.15), it would be nice if we were assured that this message passing algorithm would return beliefs $\boldsymbol{\mu} = \{b_\gamma(\boldsymbol{y}_\gamma)\}$ that are uniform, whenever $p(\boldsymbol{y})$ is uniform. More formally, a region graph is *non-singular* if message-passing with uniform factors has a unique fixed point at which the beliefs $\boldsymbol{\mu} = \{b_\gamma(\boldsymbol{y}_\gamma)\}$ are uniform[95]. In this way, non-singularity is a stronger condition than MaxEnt-Normality.

4. Over-Counting Number Unity - A region graph satisfies *over-counting number unity* if the sum of all the over-counting numbers is equal to one[102]: $\sum\limits_{\gamma \in \mathcal{R}} c_\gamma = 1$. The motivation for this criteria is similar to that of the *balance* criteria. In particular, consider a situation where our model $p(\boldsymbol{y})$ takes just two states with equal probability. For convenience, assume that $p(0, 0, ..., 0) = p(1, 1, ..., 1) = \frac{1}{2}$ and all other states have probability zero. In this case, the entropy of $p(\boldsymbol{y})$ is just $\log 2$. Further, assume that the beliefs of each region $\{b_\gamma(\boldsymbol{y}_\gamma)\}$ correspond to the exact marginals of $p(\boldsymbol{y})$. In that case, the entropy of every region is also $\log 2$. Over-counting number unity thus ensures that our entropy approximation is exact in the case that $p(\boldsymbol{y})$ takes two states with equal probability.

3.3.1 Tree-Robustness

Both the *balance* criteria and the *over-counting number unity* criteria sought exactness of the approximate entropy when $p(\boldsymbol{y})$ took a specific form: $p(\boldsymbol{y})$ was assumed to be uniform in the case of balance and $p(\boldsymbol{y})$ took two states with equal probability in the case of counting number unity. Tree-robustness takes this idea a little further, by requiring the approximate entropy to be exact when some of the factors defining the model $p(\boldsymbol{y})$ are uniform.

Consider a pairwise model $p(\boldsymbol{y}) \propto \prod_{(i,j)\in E} \psi_{ij}(y_i, y_j)$ defined on some graph $G = (V, E)$, where for notational simplicity, we have absorbed any unary factors into the pairwise terms. Let $T \subseteq E$ be some subset of the edges of G, such that the subgraph, $G' = (V, T)$, induced by the edges in T is tree-structured. Assume that the factors on the off tree edges, $\bar{T} = E \backslash T$,

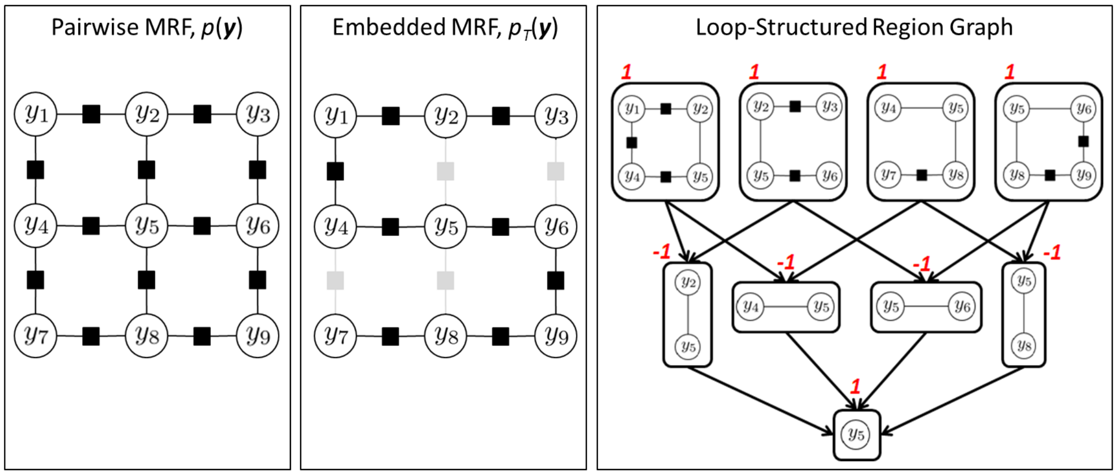


Figure 3.6: Illustration of tree-robustness on a 3×3 grid.

are uniform[3] and let $p_T(\boldsymbol{y})$ denote the distribution induced on tree T. Further, assume that the beliefs on each clique of each region in the SRG are equal to the exact marginals of $p_T(\boldsymbol{y})$. Then an SRG is *Tree-Exact* with respect to tree T if the entropy approximation $\tilde{H}^{\text{GBP}}$ is exact under these two assumptions. We say that an SRG is *Tree-Robust* if it is Tree-Exact with respect to all embedded trees in $p(\boldsymbol{y})$. In other words, a tree-robust SRG ensures that our entropy approximation is exact on any embedded, tree-structured model in $p(\boldsymbol{y})$ when the beliefs on all cliques of the SRG are correct.

Figure 3.6 illustrates the concept of tree-robustness for an MRF on a simple 3×3 grid. The middle pane of Figure 3.6 shows some embedded, tree-structured model, $p_T(\boldsymbol{y})$, in which the grayed-out factors ψ_{25}, ψ_{36}, ψ_{47} and ψ_{58} are assumed to be uniform. The exact entropy for $p_T(\boldsymbol{y})$ is

$$H_T = H_{23} + H_{12} + H_{14} + H_{45} + H_{56} + H_{69} + H_{89} + H_{78} - H_2 - H_1 - H_4 - H_5 - H_6 - H_9 - H_8,$$

where H_{ij} is the joint entropy on (y_i, y_j) and H_i is the entropy of variable i.

The right-pane shows a SRG for this model that, as we will see in the next Section, is

[3]On each edge $(i,j) \in \bar{T}$, we require that $\psi_{ij}(y_i, y_j) = 1$ for all K^2 joint configurations of (y_i, y_j).

tree-robust. Notice that the structure of region $\{y_2, y_3, y_5, y_6\}$ is tree-structured because the factor ψ_{36} is uniform. We do, however, include the clique factor f_{25} because edge region $\{y_2, y_5\}$ is a child of region $\{y_2, y_3, y_5, y_6\}$. The approximate entropy, $\tilde{H}^{\text{GBP}}$, of this SRG is:

$$\tilde{H}^{\text{GBP}} = \tilde{H}_{1245} + \tilde{H}_{2356} + \tilde{H}_{4578} + \tilde{H}_{5689} - \tilde{H}_{25} - \tilde{H}_{45} - \tilde{H}_{56} - \tilde{H}_{58} + \tilde{H}_5. \tag{3.24}$$

Assuming that the beliefs on the cliques of each region correspond to the exact marginals $p_T(\boldsymbol{y})$, then the entropy of each outer region can be re-written as

$$\begin{aligned}
\tilde{H}_{1245} =& H_{1245} = H_{2|1,4,5} + H_{1|4,5} + H_{45} = (H_{12} - H_1) + (H_{14} - H_4) + H_{45} \\
\tilde{H}_{2356} =& \tilde{H}_{3|2} + \tilde{H}_{2|5} + \tilde{H}_{56} = (H_{23} - H_2) + (H_{25} - H_5) + H_{56} \\
\tilde{H}_{4578} =& \tilde{H}_{7|8} + \tilde{H}_{8|5} + \tilde{H}_{45} = (H_{78} - H_8) + (H_{58} - H_5) + H_{45} \\
\tilde{H}_{5689} =& H_{5689} = H_{5|6,8,9} + H_{6|8,9} + H_{89} = (H_{56} - H_6) + (H_{69} - H_9) + H_{89}.
\end{aligned}$$

By substituting these expressions for the entropy of each outer region into (3.24) and simplifying, we can see that $\tilde{H}^{\text{GBP}}$ is indeed exact on the embedded, tree-structured model $p_T(\boldsymbol{y})$. In comparison, one can verify that the Bethe entropy is not exact on this embedded model.

Figure 3.7 illustrates two different Loop-SRGs for a pairwise model on 5 nodes. The left pane of Figure 3.7 shows an embedded, tree-structured model, $p_T(\boldsymbol{y})$, in which all factors are uniform, except ψ_{14}, ψ_{23}, ψ_{35} and ψ_{45}. The entropy of $p_T(\boldsymbol{y})$ is $H_T = H_{1|4} + H_{4|5} + H_{5|3} + H_{23}$. The top SRG is tree-robust. It has approximate entropy

$$\tilde{H}^{\text{top}} = \tilde{H}_{123} + \tilde{H}_{124} + \cdots + \tilde{H}_{135} + \tilde{H}_{145} - 2\tilde{H}_{12} - 2\tilde{H}_{13} - 2\tilde{H}_{14} - 2\tilde{H}_{15} + 3\tilde{H}_1. \tag{3.25}$$

Assuming that the clique beliefs in each region are exact, we can re-write the entropy of the

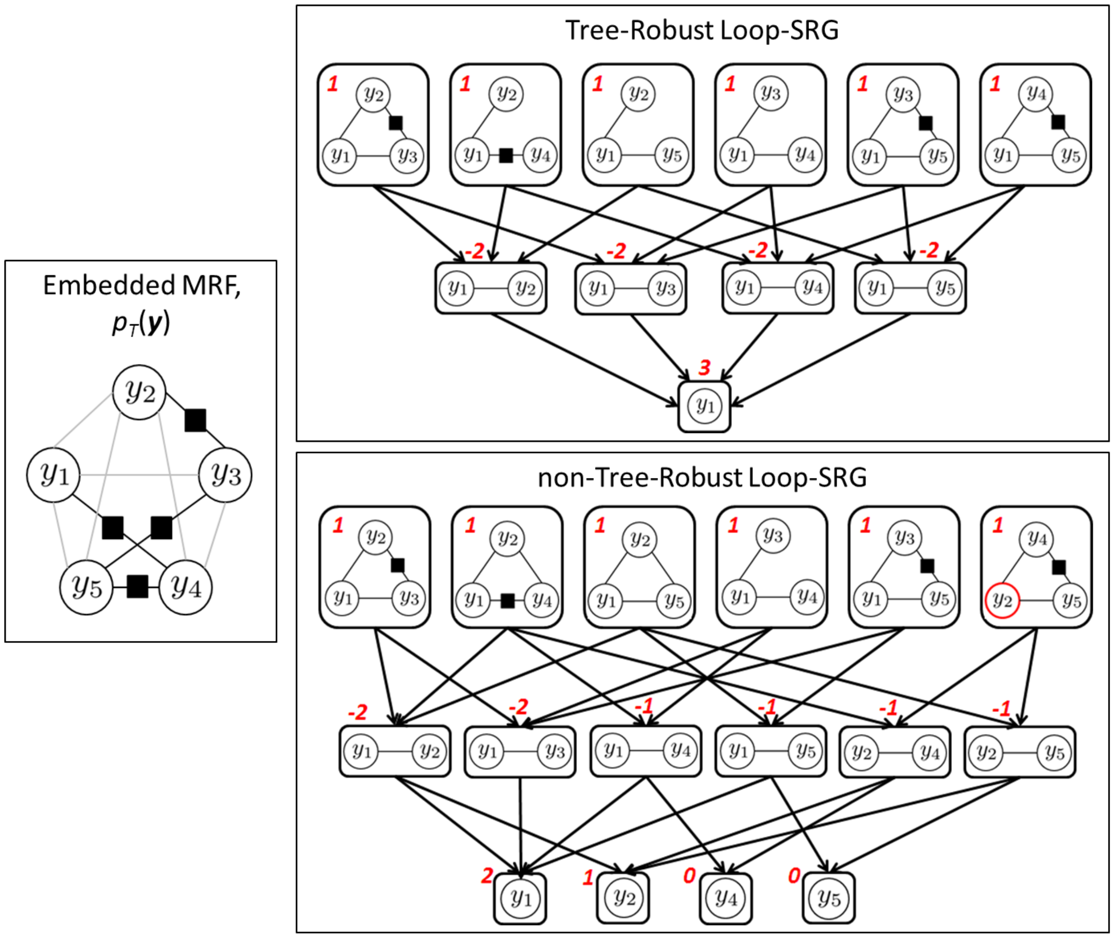


Figure 3.7: Illustration of a tree-robust and non-tree-robust SRG.

outer regions as:

$$\tilde{H}_{123} = H_{123} = H_{2|1,3} + H_{13} = H_{2|3} + H_{13}$$

$$\tilde{H}_{124} = \tilde{H}_{2|1} + \tilde{H}_{14} = H_{2|1} + H_{14}$$

$$\tilde{H}_{125} = \tilde{H}_{2|1} + \tilde{H}_{15} = H_{2|1} + H_{15}$$

$$\tilde{H}_{134} = \tilde{H}_{3|1} + \tilde{H}_{14} = H_{3|1} + H_{14}$$

$$\tilde{H}_{135} = H_{135} = H_{3|1,5} + H_{15} = H_{3|5} + H_{15}$$

$$\tilde{H}_{145} = H_{145} = H_{1|4,5} + H_{45} = H_{1|4} + H_{45}.$$

By plugging these expressions for the outer region entropies into (3.25) and canceling entropy terms for the edge and node inner regions, we can show that the entropy of the top Loop-SRG is exact for $p_T(\boldsymbol{y})$, i.e. $\tilde{H}^{\text{top}} = H_T$.

Now consider the Loop-SRG on the bottom of Figure 3.7, which differs from the top SRG in only a single outer region: region (y_1, y_4, y_5) has been replaced with region (y_2, y_4, y_5). The entropy approximation for the bottom SRG is

$$\tilde{H}^{\text{bottom}} = \tilde{H}_{123}+\tilde{H}_{124}+\cdots+\tilde{H}_{135}+\tilde{H}_{245}-2\tilde{H}_{12}-2\tilde{H}_{13}-\tilde{H}_{14}-\tilde{H}_{15}-\tilde{H}_{24}-\tilde{H}_{25}+2\tilde{H}_1+\tilde{H}_2. \quad (3.26)$$

Once again, assume that the clique beliefs in each region are exact and consider the entropies for the first three outer regions,

$$\begin{aligned}
\tilde{H}_{123} =& H_{123} = H_{2|1,3} + H_{13} = H_{2|3} + H_{13} \\
\tilde{H}_{124} =& H_{124} = H_{1|2,4} + H_{24} = H_{1|4} + H_{24} \\
\tilde{H}_{125} =& H_{125} = H_{1|2,5} + H_{25} = H_{1|5} + H_{25}.
\end{aligned}$$

These are the only three outer regions that are parents of edge region (y_1, y_2) in the region graph. However, none of the entropies of these regions decompose in a fashion that can cancel with the entropy H_{12} in (3.26). As a result, there is no way that $\tilde{H}^{\text{bottom}}$ of the bottom Loop-SRG can be equal to, H_T, the exact entropy of $p_T(\boldsymbol{y})$. This, in turn, implies that the bottom Loop-SRG is not tree-robust.

3.4 A Bottom-Up Approach to Constructing Region Graphs

The previous section identified a set of criteria that *should* be satisfied by the collection of regions $\mathcal{R}$ defining a GBP approximation. However, it is not immediately clear how to satisfy these criteria, given an arbitrary model $p(\boldsymbol{y})$. The brute force strategy of searching over all collections of regions $\mathcal{R} \subset 2^m$ and evaluating each possible collection on each of the criteria is clearly infeasible. In this section, we restrict attention to finding Loop-SRGs of pairwise models $p(\boldsymbol{y})$. We introduce an algorithm for choosing collections $\mathcal{R}$ that works in a bottom-up fashion, by grouping edges into loop-structured regions, that satisfy the 5 criteria discussed in the previous section. This is in contrast to top-down approaches, like the *Join-Graph-Structuring* procedure[62] mentioned in Section 1.4.1, which start from a junction-tree and split large regions into smaller, tractable sub-regions.

Our approach to finding good collections of regions makes explicit use of the form of the Loop-SRG. In particular, we focus on on finding a collection of loop outer regions $\mathcal{O}$ and then fill in the edge and node inner regions using the CVM, which guarantees that our Loop-SRG is totally-connected and totally-balanced[71]. To find the collection of loop outer regions, we map each loop outer region in $\mathcal{O}$ to a cycle in the undirected graph, G, underlying the pairwise model $p(\boldsymbol{y})$ and then search for collections of cycles that satisfy certain structural properties. We can thus appeal to tools from graph theory to find $\mathcal{O}$, rather than searching over subsets of variables.

The remainder of this section is organized as follows. We begin by reviewing some basic theory of cycle spaces and cycle bases. We then show that if our collection of loop outer regions comprise a special type of cycle basis, known as a fundamental cycle basis, then our Loop-SRG will be non-Singular and satisfy counting-number unity. We then introduce and define tree-robust cycle bases, which are a special kind of fundamental basis, and show that if $\mathcal{O}$ comprises a tree-robust bases, our Loop-SRG will be tree-robust as well. We then identify tree-robust bases in planar graphs and complete graphs, two structures that are common in pairwise graphical models. Finally, we introduce an algorithm for finding tree-robust, or nearly tree-robust collections of loop outer regions given some arbitrary pairwise model $p(\boldsymbol{y})$.

3.4.1 Cycle Spaces and Cycle Bases

We begin with some background on cycle bases taken from [50]. Let $G = (V, E)$ be a 2-connected graph. A simple cycle C in G is a connected Eulerian subgraph in which every vertex has degree 2. The cycle space $\mathfrak{C}(G)$ of a graph G consists of all simple and non-simple cycles of G, including the empty cycle $\emptyset$. The dimension of a cycle space for a graph with 1 connected component is $\rho \equiv \rho(G) = |E| - |V| + 1$.

Definition 3.2. *A* *Cycle Basis* *of* $\mathfrak{C}(G)$ *is a set of simple cycles* $\mathcal{B} = \{C_1, ..., C_\rho\}$ *such that for every cycle* C *of* G*, there exists a unique subset* $\mathcal{B}_C \subseteq \mathcal{B}$ *such that the set of edges appearing an odd number of times in* $\mathcal{B}_C$ *comprise the cycle* C*.*

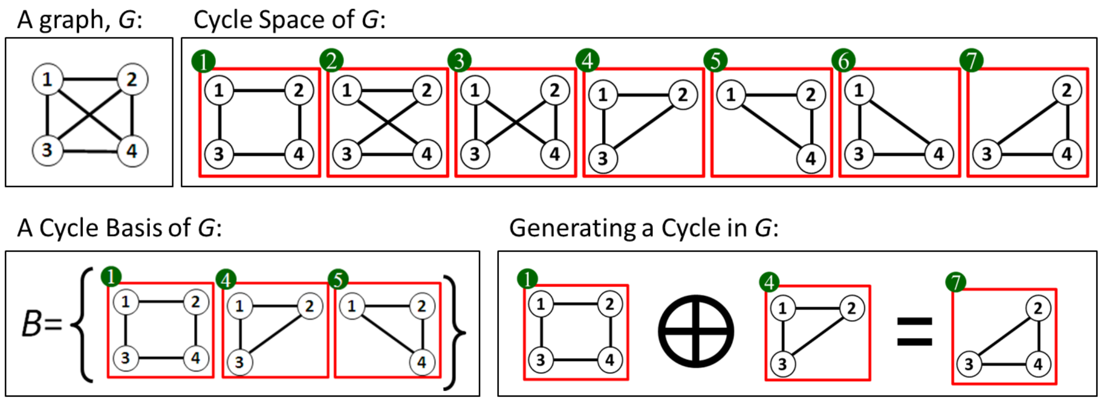


Figure 3.8: Illustration of a graph, its cycle space and a cycle basis.

Figure 3.8 depicts the cycle space for the complete graph of 4 nodes (K_4) and a sample cycle basis. The cycle basis can be used to *generate* any cycle in the cycle space. As an example, the cycle $C_7 = \{3, 2, 4\}$ is generated by combining basis cycle $C_1 = \{1, 2, 3, 4\}$ with basis cycle $C_4 = \{1, 2, 3\}$.

Definition 3.3. *A cycle basis* $\mathcal{B}$ *is a* *Fundamental Cycle Basis (FCB)* *if there exists a permutation* π *of the cycles in* $\mathcal{B}$ *such that* $C_{\pi(i)} \setminus \left\{C_{\pi(1)} \cup \cdots \cup C_{\pi(i-1)}\right\} \neq \emptyset$ *for* $i = 2...\rho$*. In other words, the cycles can be ordered so that cycle* $C_{\pi(i)}$ *has some edge that does not appear in any cycle preceding it in the ordering.*

Figure 3.9 depicts a graph and a cycle basis that is clearly non-fundamental, as every edge in the graph appears in at least 2 cycles in the basis.

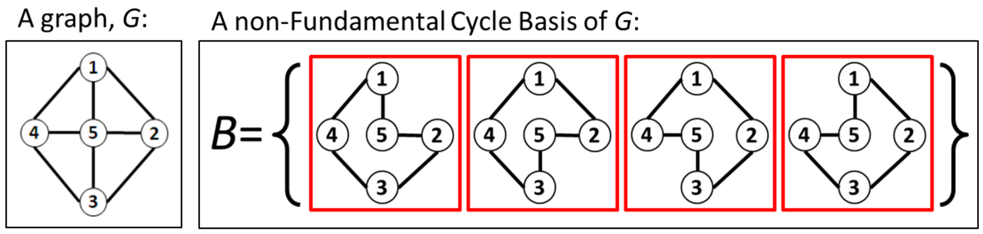


Figure 3.9: Illustration of a graph and a cycle basis that is non-fundamental.

Consider mapping each loop outer region $\gamma \in \mathcal{O}$ to a cycle C in the basis $\mathcal{B}$ of the undirected graph of $p(\boldsymbol{y})$ [4]. Using this mapping, we can claim the following:

Theorem 3.1. *A Loop-SRG is Non-Singular and satisfies Over-Counting Number Unity if its loop outer regions are a Fundamental Cycle Basis (FCB) of G.*

The proof of this theorem appears in Appendix C. It uses a set of *reduction operators* introduced in [95] that modify an SRGs structure while preserving the fixed points of the constrained variational optimization problem. In proving Theorem 3.1, we use the reduction operators to show that a loop outer region with a unique edge (i.e. an edge not shared with any other loop region) can be *reduced* to the set of edges comprising that loop. Since the set of loop outer regions form a FCB, we are guaranteed to find a loop region with a unique edge if we reduce the loops along the order π - i.e. beginning with the loop corresponding to $C_{\pi(\rho)}$ and ending at loop $C_{\pi(1)}$.

Recall the two Loop-SRGs considered in Figure 3.7. They specified two different GBP approximations for a pairwise model on the complete graph of 5 nodes. The cycle space for the graph K_5 has dimension $\rho = |E| - |V| + 1 = 10 - 5 + 1 = 6$. One can verify that the collection of 6 loop outer regions used in both SRGs are fundamental cycle bases. As a

[4]More specifically, the structure $G(\gamma)$ of each outer region $\gamma \in \mathcal{O}$ is chosen to be a unique cycle $C \in \mathcal{B}$.

result, both of the SRGs are non-singular and satisfy over-counting number unity. However, only one of the SRGs was tree-robust.

This result implies that the loop regions in a Loop-SRG should form a FCB. This greatly reduces the set of loops considered when constructing a Loop-SRG. However, a graph may have many fundamental bases, so it is natural to ask if Loop-SRGs formed from certain FCBs are better than others. We now introduce and define a class of Loop-SRGs with loop regions corresponding to tree-robust cycle basis, a specific type of FCB.

3.4.2 Tree Robust Cycle Bases

The previous section established that a Loop-SRG is non-singular and satisfies over-counting number unity if its loops form a FCB. In this section, we first define a tree-robust cycle basis and then show that a Loop-SRG is tree-robust if its loops form a tree-robust cycle basis.

Definition 3.4. *Let T be some spanning tree of G. A cycle basis $\mathcal{B}$ is Tree Exact w.r.t. T if there exists an ordering π of the cycles in $\mathcal{B}$ such that $\left\{C_{\pi(i)} \setminus \{C_{\pi(1)} \cup \cdots \cup C_{\pi(i-1)}\}\right\} \setminus T \neq \emptyset$ for $i = 2...\rho$. In other words, the cycles in the basis can be ordered so that cycle $C_{\pi(i)}$ has some edge that:*

1. *Does not appear in any cycle preceding it in the ordering; and*
2. *Does not appear in the spanning tree T.*

Definition 3.5. *A cycle basis $\mathcal{B}$ is Tree Robust (TR) if it is Tree Exact w.r.t. all spanning trees of G.*

By mapping each loop outer region to a cycle in basis $\mathcal{B}$, we can show that:

Theorem 3.2. *A Loop-SRG is Tree Robust if its loop outer regions are a Tree Robust cycle basis of G.*

The proof of this result appears in Appendix C and is similar in spirit to the proof that loop outer regions forming an FCB are non-singular and satisfy counting number unity. The proof is of course complicated by the spanning tree requirement.

In the remainder of this section, we introduce two theorems that characterize TR cycle bases. These results prove useful in showing that, for example, the faces of a planar graph constitute a TR cycle basis. The ensuing theorems require the following definition.

Definition 3.6. *The Unique Edge Graph of a set of cycles $\mathcal{C} = \{C_1, ..., C_k\}$ is a graph comprised of the set of edges that are in exactly one cycle in $\mathcal{C}$. We will use $I(\mathcal{C})$ to denote the unique edge graph for cycle set $\mathcal{C}$. $I(\mathcal{C})$ is* cyclic *if it contains at least one cycle.*

Using this definition we can now state the following, equivalent characterization of TR cycle bases.

Theorem 3.3. *Let $\mathfrak{B}^{|k|}$ denote all size k subsets of cycles in $\mathcal{B}$. A FCB $\mathcal{B}$ is Tree Robust iff $I(\mathcal{B}_k)$ is cyclic and not empty for all $\mathcal{B}_k \in \mathfrak{B}^{|k|}$ for $1 \leq k \leq \rho$. In other words, the unique edge graph must be cyclic for all pairs of cycles, and all triples of cycles,..., and all of the ρ cycles.*

Corollary 3.4. *An FCB is TR iff $\mathcal{B}_k$ is TR for all $\mathcal{B}_k \in \mathfrak{B}^{|k|}$ for $1 \leq k \leq \rho$.*

The full proof is in Appendix C, but we sketch the main idea here. Sufficiency follows because the unique edge graph will be cyclic no matter what ordering, π, we choose to remove the cycles in. Since the unique edge graph is always cyclic, there can be no spanning tree T that *covers* all of the edges of the unique edge graph. Necessity follows because if there were a subset of cycles for which the unique edge graph is acyclic, then there exists no ordering π that can avoid that subset (or a larger subset with an acyclic unique edge graph). Hence, by choosing the tree T to block all edges of the acyclic unique edge graph under consideration we prove that the basis is not tree robust.

In fact, this is precisely how we showed that the bottom Loop-SRG in Figure 3.7 was not tree-robust. The unique edge graph for that Loop-SRG is acyclic after reducing loop region (y_1, y_3, y_4) and has the set of edges used to define the embedded, tree-structured model, $p_T(\boldsymbol{y})$, in that example.

3.4.3 Planar and Complete Graphs

Using the theorems from the previous section, we now identify TR cycle bases of planar and complete graphs. In the case of planar graphs, a TR basis can be constructed from the set of cycles forming the faces of the planar graph. This supports the observation of previous authors that GBP run on the faces of planar graphs gives accurate results.

Theorem 3.5. *Consider a planar graph G. The cycle basis $\mathcal{B}$ comprised of the faces of G is TR.*

Proof. We use theorem 3.3. Consider the graph formed by any subset of k faces of the planar graph. Consider any of its connected components. The path that traces the circumference of that component is a loop and also consists of unique edges. □

In the case of complete graphs, a TR basis can be constructed by choosing some vertex as a root, creating a spanning tree with edges emanating from that root and constructing loops of length 3 using each edge not in the spanning tree. This is exactly the 'star' construction proposed in [95], which was shown empirically to be superior to other Loop-SRGs on complete graphs.

Theorem 3.6. *Consider a complete graph G on n vertices (i.e. K_n). Construct a cycle basis $\mathcal{B}$ as follows. Choose some vertex v as the root. Create a 'star' spanning tree rooted at v (i.e. with all edges v-u). Now construct cycles of the form v-i-j from each off-tree edge i-j. The basis $\mathcal{B}$ constructed in this way is TR.*

Proof. We use again theorem 3.3. Consider the graph constructed from any subset of k triangles. The edges not on the spanning tree (i.e. not connecting to the root) are all unique and will either form a loop (in which case we are done) or a tree. In case of a tree, consider a path connecting two leaf nodes, which are both also connected to the root, thus forming a loop. Because the two edges connecting to the root are also unique (since they correspond to leaf nodes) we have proven the existence of a cycle in the unique edge graph. $\square$

This result can be extended to partially complete graphs where there exists some vertex v that is connected to all the other vertices in G.

3.4.4 Finding TR SRGs in General Pairwise Graphical Models

The previous section identified TR cycle bases for probability distributions defined over two specific classes of graphs: planar and complete. The prescription for how to construct TR SRGs for MNs on these types of graphs is clear: find a TR basis $\mathcal{B}$ of the underlying graph, make the cycles in $\mathcal{B}$ the loop outer regions and fill in edge and node inner regions using the CVM. This prescription can be generalized in some cases to graphs containing many TR components. More formally,

Theorem 3.7. *Consider a graph G comprised of components (subgraphs) $H_1, ..., H_k$. Let the components $H_1, ..., H_k$ be mutually* <u>*singly connected*</u> *if for any two components H_i and H_j there exist vertices $v_i \in H_i$ and $v_j \in H_j$ that are singly connected (i.e. connected through a single path). Let $\mathcal{B}_{H_1}, ..., \mathcal{B}_{H_k}$ denote the TR cycle bases for each component. Then a TR cycle basis of G is simply the union of the TR cycle bases of each component: $\mathcal{B}_G = \{\mathcal{B}_{H_1} \cup \cdots \cup \mathcal{B}_{H_k}\}$.*

Proof. First note that by singly connecting the components $H_1, ..., H_k$ we do not create any new cycles. Thus $\mathcal{B}_G$ is a cycle basis of G. Every subset of cycles of $\mathcal{B}_G$ is the union of some subset of cycles from the component cycle bases $\{\mathcal{B}_{H_i}\}$. Moreover, for any of these subsets the unique edge graph must be cyclic (by theorem 3.3). Since the component cycle bases do not overlap, it follows that the unique edge graph for every subset of cycles of $\mathcal{B}_G$ must also be cyclic. □

For pairwise models defined over more general graphs, the picture of how to construct a TR SRG is less clear. Since verifying that a basis is TR requires inspecting all subsets of cycles in that basis, searching for a TR basis in a general graph seems difficult. Moreover, not every graph will admit a TR basis. As a result, we now describe a method for constructing Loop-SRGs that are partially TR. In other words, we sacrifice finding a TR basis of G to find a basis that is Tree Exact for many (just not all) spanning trees of G.

The method for finding a partially TR basis works as follows. We first find the largest complete or planar subgraph H of G and construct a TR basis $\mathcal{B}(\mathcal{H})$ for H as described in the previous section. Since the TR core H is a subgraph of G, the $\rho(H)$ cycles in $\mathcal{B}(\mathcal{H})$ will not form a complete basis of G. We choose the remaining $\rho(G) - \rho(H)$ cycles so that the basis of G is fundamental. We do so by finding a sequence of *ears* (simple paths or cycles) in G, such that each new *ear* has some edge not occurring in some previous ear. This process is described in Algorithm 3.1.

Figure 3.10 illustrates Algorithm 3.1 on a 2×3 grid. The pairwise model $p(\boldsymbol{y})$ is shown on the left and its TR core H on the right. We first add the faces $(1, 2, 5, 4)$ and $(2, 3, 6, 5)$ of the TR core H to the basis $\mathcal{B}$. We then mark all of the edges in H as used and all of the vertices as visited. Edge $(1, 6)$ is an ear, or unused edge. The path $6, 3, 2, 1$ connects the *start* vertex, 6, to the *end* vertex, 1, of this ear along used edges. As a result, cycle $(1, 6, 3, 2)$ is added to $\mathcal{B}$. Similarly, cycle $(3, 4, 5, 6)$ is added for ear $(3, 4)$, giving us the partially TR basis $\mathcal{B} = \{(1, 2, 5, 4), (2, 3, 6, 5), (1, 6, 3, 2), (3, 4, 5, 6)\}$.

Algorithm 3.1 Find partially-TR Cycle Basis

Input: A pairwise graphical model, $p(\boldsymbol{y})$, with underlying undirected graph G
Output: A Cycle Basis $\mathcal{B}$

```
1: Find a TR subgraph $H$ of $G$ with TR basis $\mathcal{B}(H)$
2: if $G$ contains no TR subgraph (i.e. $H = \emptyset$) then
3:     Let $H$ be some simple cycle in $G$
4: end if
5: Add cycles $\mathcal{B}(H)$ to $\mathcal{B}$
6: Mark all edges in $H$ as used
7: Mark all vertices in $H$ as visited
8: while $\exists$ unused edge $e = (s, t)$ from a visited vertex $s$ do
9:     If $t$ is visited, then set $p_1 = e$
10:    Else, find an ear $p_1$ from $s$ through edge $e = (s, t)$ to some visited vertex $u$.
11:    Find shortest path $p_2$ from $s$ to $u$ on used edges
12:    Add cycle $C$ consisting of $p_1 \cup p_2$ to the bases $\mathcal{B}$
13:    Mark all edges (vertices) on $C$ as used (visited)
14: end while
```

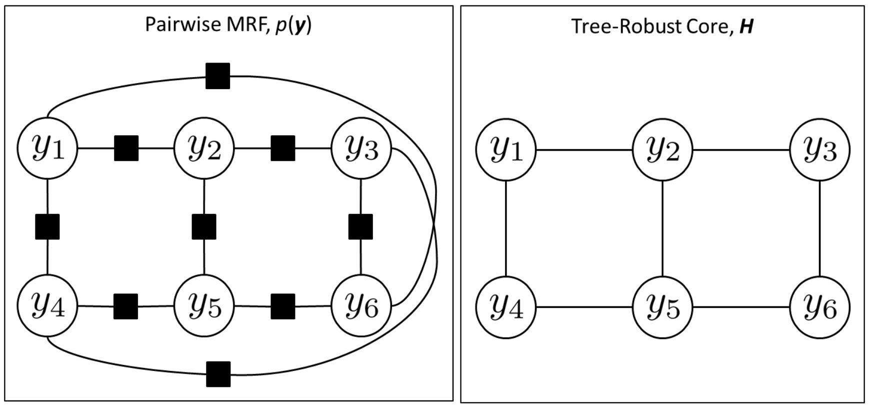

Pairwise MRF, $p(\boldsymbol{y})$
Tree-Robust Core, $\boldsymbol{H}$
y_1
y_2
y_3
y_4
y_5
y_6

Figure 3.10: Illustration of finding a partially-Tree-Robust cycle basis.

3.5 Experiments

In section 3.3 we introduced several region selection criteria, including our new tree-robustness criteria. Tree-robustness guarantees exactness of the GBP entropy approximation when a tree-inducing collection of factors in our model are uniform and the GBP beliefs correspond to the exact marginals of $p(\boldsymbol{y})$. Unfortunately, both of these assumptions are unlikely to hold in practice. Of course, the same can be said of the assumptions motivating the balance, over-counting number unity and MaxEnt-Normality and non-Singularity criteria as well.

We thus conducted a series of experiments to validate the recommendations for constructing Loop-SRGs made in this chapter.

We begin our empirical analysis by justifying the recommendation that the collection of loop outer regions constitute a fundamental cycle basis and thus satisfy both *over-counting number unity* and *non-Singularity.* We then move to justifying the recommendation that the collection of outer regions be tree-robust. Finally, we demonstrate the effectiveness of Algorithm 3.1 in pairwise models that do not have an easily identifiable tree-robust cycle basis.

In the experiments that follow, we focus on pairwise MRFs, $p(\boldsymbol{y}) \propto \prod\limits_{i \in V} \psi_i(y_i) \prod\limits_{(i,j) \in E} \psi_{ij}(y_i, y_j)$, defined over many different underlying graph structures $G = (V, E)$. Every variable $i \in V$ in the model is binary and has an associated unary potential of the form $\psi_i(y_i) = [\exp(\theta_i);\ \exp(-\theta_i)]$, where $\theta_i \sim \mathcal{N}(0, \sigma_i^2)$. Each edge $(i, j) \in E$ in the model has a pairwise potential of the form $\psi_{ij}(y_i, y_j) = [\exp(\theta_{ij})\ \exp(-\theta_{ij});\ \exp(-\theta_{ij})\ \exp(\theta_{ij})]$, where $\theta_{ij} \sim \mathcal{N}(0, \sigma_{ij}^2)$. Two different error measures are reported. $Error_Z = |\log Z - \log \tilde{Z}|$ is the absolute error between the exact ($\log Z$) and approximate ($\log \tilde{Z}$) values of the log partition function. $Error_{L_1}$ measures the mean absolute error in the marginal probability estimates and is computed as $Error_{L_1} \overset{\text{def}}{=} \frac{1}{m} \sum\limits_{i=1}^{m} \sum\limits_{y_i \in \{0,1\}} |b(y_i) - p(y_i)|$, where $m = |V|$ is number of variables in the model, $b(y_i)$ is the estimated marginal (or belief) of variable y_i and $p(y_i)$ is the true marginal of variable y_i.

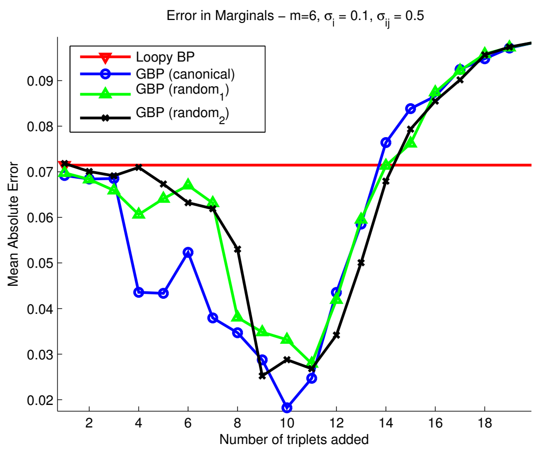
3.5.1 Empirical Justification of Over-Counting Number Unity

We first ran an experiment to validate our recommendation that the collection of loop outer regions satisfy the over-counting number unity criteria. One can verify that the sum of the counting numbers in a Loop-SRG is: $\sum_{\gamma \in \mathcal{R}} c_\gamma = L - E + V$, where L is the number of loop outer regions, E is the number of edge inner regions and V is the number of node inner regions (see e.g. [95]). Importantly, $\sum_{\gamma \in \mathcal{R}} c_\gamma = 1$, when the number of loops is equal to the dimension of the cycle space of G, i.e. $L = \rho(G)$. Further, $\sum_{\gamma \in \mathcal{R}} c_\gamma < 1$ when $L < \rho(G)$ and $\sum_{\gamma \in \mathcal{R}} c_\gamma > 1$ when $L > \rho(G)$.

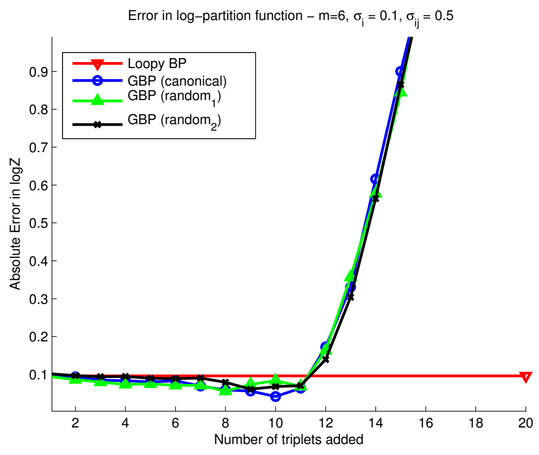
We considered a pairwise model on the complete graph of 6 nodes (K_6) and construct a sequence of Loop-SRGs as follows: starting with the Bethe region graph, which contains no loop outer regions, we successively add triplet loop outer regions to our SRG. There are a total of 20 triplets in the graph K_6 and we add these 20 triplets as outer regions along several different orderings. First, is a canonical ordering, where we begin by adding triplet $(1, 2, 3)$, followed by triplet $(1, 2, 4)$ and end by adding triplet $(4, 5, 6)$. Adding triplets along the canonical order produces an SRG with a tree-robust cycle basis when exactly $\rho(K_6) = 15 - 6 + 1 = 10$ triplets have been added. We also considered adding triplet loop regions along two distinct random orderings.

Figure 3.11 contains a few plots of error as a function of the number of triplet outer regions added to our GBP approximation. The top row contains $Error_{L_1}$ (left) and $Error_Z$ (right) averaged over a collection of 50 random models generated with $\sigma_i = 0.1$ and $\sigma_{ij} = 0.5$. The bottom row contains $Error_{L_1}$ (left) and $Error_Z$ (right) for a collection of 50 random models generated with $\sigma_i = 0.1$ and $\sigma_{ij} = 1.0$. To negate any errors stemming from non-convergence, in these experiments we used a convergent, double-loop, form of GBP discussed in [43].

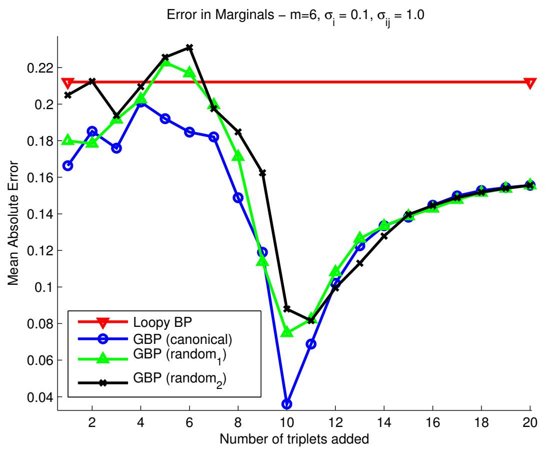
Notice that for both settings of σ_{ij}, we see a strong dip when the collection of outer regions contains $L = \rho(K_6) = 10$ triplets – precisely when the over-counting number unity property is satisfied. While the dip is more pronounced along the canonical ordering (shown in blue), where we know our triplet outer regions form a cycle basis when $L = 10$, the dip exists even along the random orderings where the collection of $L = 10$ regions need not form a cycle basis. After 10 outer regions have been added, the quality of GBP's log-partition function estimate deteriorates rapidly, quickly becoming worse than Loopy BP. These experimental results seem to support our recommendation that the collection of loop outer regions should



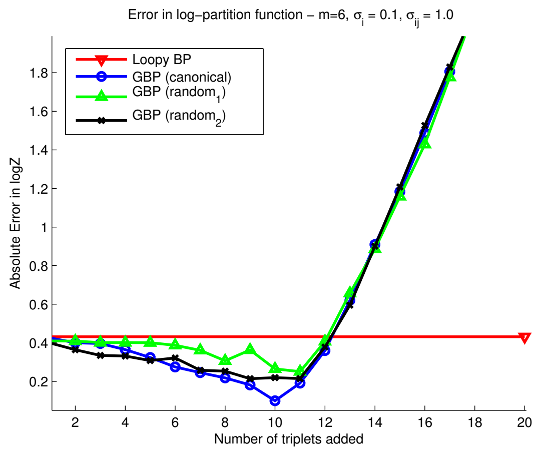
(a) $Error_{L_1}$ for model with $\sigma_i = 0.1$, $\sigma_{ij} = 0.5$.



(b) $Error_Z$ for model with $\sigma_i = 0.1$, $\sigma_{ij} = 0.5$.



(c) $Error_{L_1}$ for model with $\sigma_i = 0.1$, $\sigma_{ij} = 1.0$.



(d) $Error_Z$ for model with $\sigma_i = 0.1$, $\sigma_{ij} = 1.0$.

Figure 3.11: Error as the number of loop outer regions added to a Loop-SRG is increased.

satisfy the over-counting number unity property.

3.5.2 Empirical Justification of non-Singularity

We next ran an experiment to show that choosing loop regions that form a fundamental cycle basis leads to better approximations than choosing a collection of regions that form a cycle basis. In other words, the goal of this experiment is to empirically validate the claim that non-Singularity is desirable characteristic of a GBP approximation.

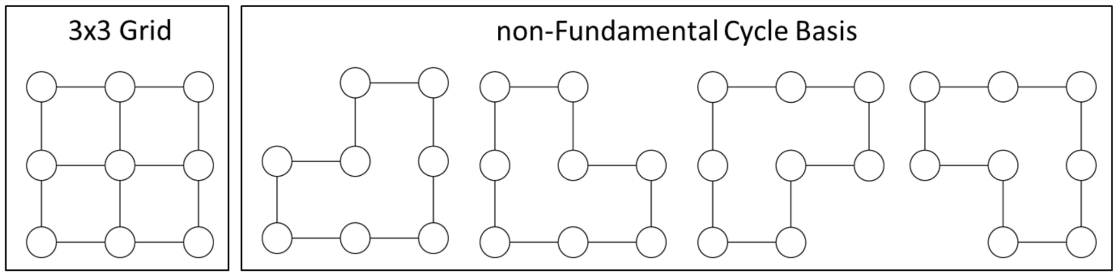
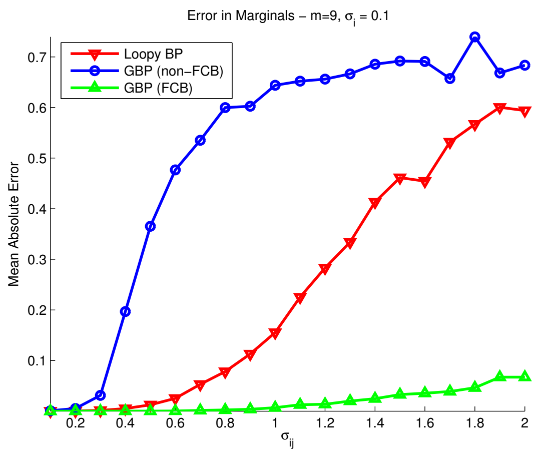


Figure 3.12: A non-Fundamental Cycle Basis on a 3×3 grid.

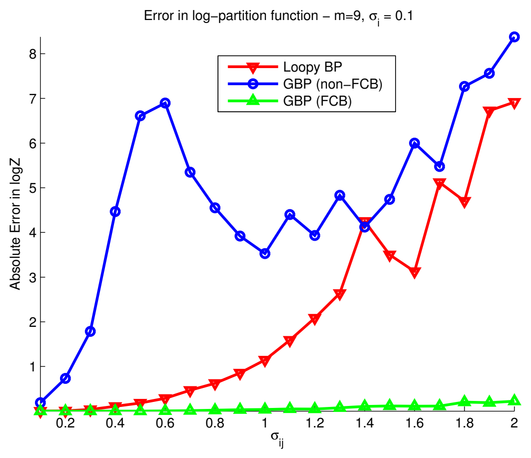
We conducted a simple experiment on a 9×9 grid of variables in standard 4-neighbor connectivity. We constructed a Loop-SRG using a collection of loop outer regions comprised of the 64 faces of the 9×9 grid. As we saw in Section 3.4.3 the faces of a planar graph constitute a tree-robust, and therefore, fundamental cycle basis. We also chose a collection of 64 loop outer regions comprising a non-fundamental cycle basis by repeating the non-FCB construction of the 3×3 grid shown in Figure 3.12 a total of 16 times in the 9×9 grid.

Figure 3.13 shows the error in the marginal estimates and log-partition function estimates of both the FCB construction (green) and the non-FCB construction (blue). In these experiments, we generated 50 random grid instances for each setting of σ_{ij}, holding σ_i fixed at 0.1. We ran a damped version of GBP discussed in Section 3.2, where beliefs were updated as in (3.15) using $\alpha = \frac{1}{2}$. The errors reported are averaged across the 50 instances at each setting of σ_{ij}.

Notice that the error of the non-FCB SRG is much worse than the FCB SRG construction as σ_{ij} is increased and the models become increasingly frustrated. In fact, the error of the non-FCB estimates quickly becomes larger than the error of the Loopy-BP estimates shown in red. Interestingly, both MaxEnt-Normality and non-Singularity only require the entropy approximation to behave sensibly when the factors of the model are uniform. As σ_{ij} is increased, however, the factors become increasingly less uniform. This fact, coupled with the empirical evidence in Figure 3.13 certainly suggest that choosing a collection of regions that are non-singular is a good idea.



(a) $Error_{L_1}$ for models with $\sigma_i = 0.1$ and increasing σ_{ij}.



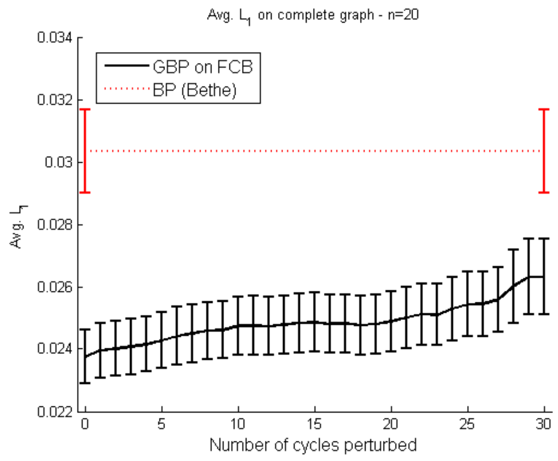
(b) $Error_Z$ for models with $\sigma_i = 0.1$ and increasing σ_{ij}.

Figure 3.13: Comparison of Loop-SRGs formed with Fundamental and non-Fundamental Cycle Bases.

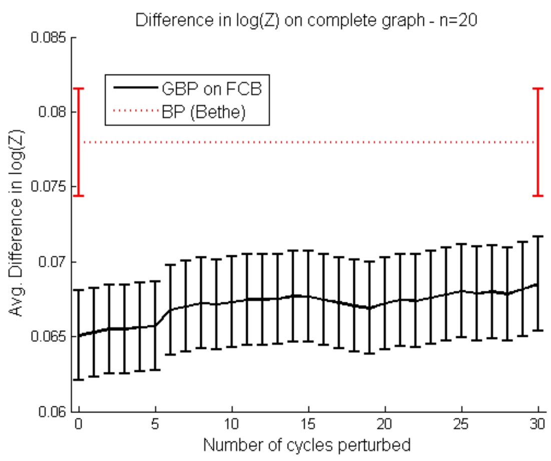
3.5.3 Empirical Justification of Tree-Robustness

We next ran a set of experiments to justify our recommendation that the collection of regions be tree-robust. Since every tree-robust cycle basis is a fundamental basis, and therefore satisfies both non-Singularity and counting number unity, we wanted to show that one gets more accurate estimates when using a Loop-SRG comprised of a tree-robust cyle basis rather than a Loop-SRG comprised of fundamental, but not tree-robust cycle basis.

To test this hypothesis, we generated 31 different Loop-SRGs for a pairwise model on a complete graph with 20 binary variables (K_{20}). We first constructed a tree-robust basis $\mathcal{B}_{\mathrm{TR}}$, using the *star* construction process described in Section 3.4.3. Under this construction our tree-robust basis is comprised of all cycles of length 3 (i.e. all triplets) passing through vertex 1. From this tree-robust basis, a sequence of 30 fundamental, but non-tree-robust cycle bases were created as follows: for $i = 1...30$, we choose a cycle C_i of the form $C_i = (1, u, v)$ from $\mathcal{B}_{\mathrm{TR}}$ and modify it by swapping vertex 1 with some vertex w ($w \neq u \neq v \neq 1$) so that $C_i = (w, u, v)$. At every iteration we choose cycles that have not been modified previously and reject modifications that make the basis non-fundamental. In this way, as i increases the cycle basis is made *less* tree-robust, while remaining fundamental.



(a) $Error_{L_1}$ on increasingly non-TR Loop SRGs.



(b) $Error_Z$ on increasingly non-TR Loop SRGs.

Figure 3.14: Performance of GBP run on a sequence of increasingly non-TR Loop-SRGs.

Figure 3.14 shows $Error_{L_1}$ and $Error_Z$ as the Loop-SRG is made less tree-robust. In this figure, we generated 500 random model instances with $\sigma_i = 1$ and $\sigma_{ij} = 1/\sqrt{20-1}$. Note that by keeping the cycle length at 3 and ensuring that each basis is fundamental, the increase in error can only be explained by the change in the tree-robust core of the basis. Since error increases as the basis is made less tree-robust, these results support our claim that tree robustness is a desirable property. It was also observed that GBP took an increasing number of iterations to converge as the basis was made less tree-robust.

3.5.4 Experiments on Partially Tree-Robust GBP Approximations

The previous experiments considered pairwise models with an underlying graph structure for which a tree-robust basis was known. In this section, we consider more general pairwise models, which are neither planar nor complete, and use Algorithm 3.1 to find a collection of cycles that form a partially tree-robust cycle basis.

Algorithm 3.1 seeks an initial, tree-robust subgraph $H \subset G$ of the graph, G, underlying our pairwise model. The previous experiments indicated that GBP yields accurate approximations when $H = G$. When a graph contains no tree-robust core (i.e. $H = \emptyset$), then Algorithm

3.1 simply finds a fundamental cycle basis of G. We wish to study the performance of GBP between these two extremes - i.e. on *partially* tree-robust SRGs. We conducted experiments on two types of models, where the size of the sub-graph H (relative to G) can be controlled.

The following experiments include a comparison to the Iterative Join Graph Propagation (IJGP) method [20] discussed in Section 1.4.1. Recall that IJGP forms a join (cluster) graph, which is just a two level region graph, in a top-down fashion. Starting from a cluster tree decomposition, it uses the mini-bucket heuristic to divide large, intractable clusters into smaller, computationally manageable sub-clusters. The number of variables appearing in any outer region of an IJGP approximation is less than or equal to a control parameter known as the *iBound*. The *iBound* controls the complexity of message computations in IJGP because each outer region forms a clique over all of the variables in a region. Importantly, since Loop-SRGs assume a loop structure in the outer regions, message computations on Loop-SRGs are equivalent to IJGP with $iBound = 3$.

Partial K-Trees

In these experiments we construct a set of partial K-tree instances via the following procedure[5]. We first build a random K-tree on m vertices using the process described in [29]. The number of neighbors (or degree) of the vertices in K-trees constructed by this procedure follow a power law. This means there will exist a few vertices that are adjacent to most of the vertices in G. As a result, the tree-robust core will comprise a large portion of G. To reduce the size of the tree-robust core, we iteratively remove edges from the K-tree as follows. First choose a vertex v with probability proportional to the current degree of that vertex. Then modify G by removing an edge from v to one of its neighbors, so long as removing that edge does not disconnect G. This process is repeated until the ratio of the maximum

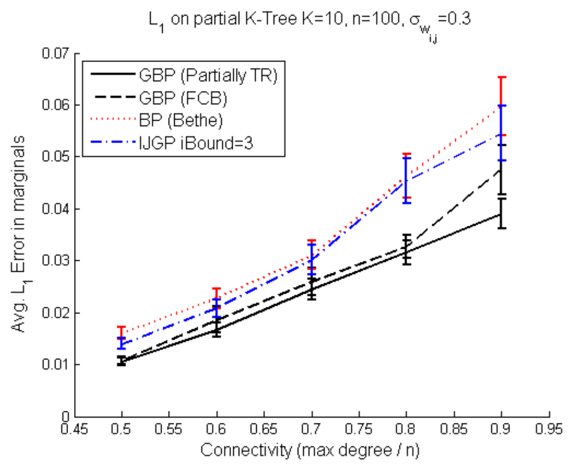
[5]K-trees are chordal graphs with maximal cliques of size K. Partial K-trees are non-chordal graphs with maximal cliques of size K.

degree in G to the number of vertices m falls below some threshold. We refer to this ratio as the connectivity of the graph. A random pairwise MRF is formed over each partial K-tree structure by assigning random unary and pairwise potentials to the vertices and edges.

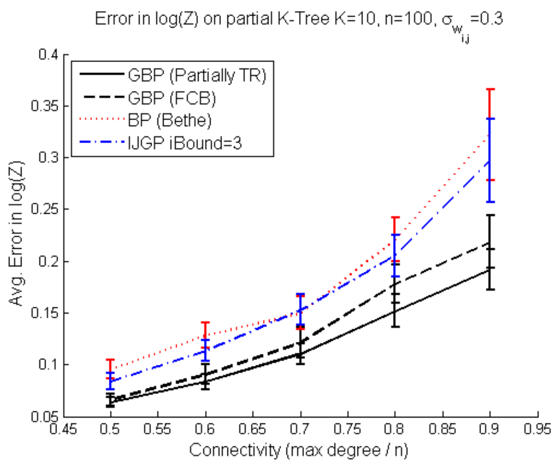
Figure 3.15 shows the performance of GBP as a function of connectivity. In these plots, we generated 100 random models at each level of connectivity, with $K = 10$, $m = 100$, $\sigma_i = 1$ and $\sigma_{ij} = 0.3$ held fixed. The partially tree-robust SRGs are found by first choosing the tree-robust core $H \subset G$ as follows. We find the vertex v_{root} with maximum degree in G and designate it as our root. We then use the *star* construction described in Section 3.4.3 to find all triplets that include the designated root vertex, v_{root}. Cycles are added to this tree-robust core as described in Algorithm 3.1. The partially tree-robust SRGs are compared to Loop-SRGs that are formed by finding a random FCB of each partial K-tree. In particular, we use Algorithm 3.1 with $H = \emptyset$ to find a random FCB. Importantly, these random FCBs do not build upon the tree-robust core, which as we now demonstrate, leads to less accurate estimates.

Figure 3.15 shows that the benefit of the partially tree-robust SRG diminishes as connectivity is decreased. This behavior confirms our belief that the benefit of finding a tree-robust core decreases as the core comprises a smaller proportion of cycles in the fundamental basis. Even so, it is important to note that choosing a Loop-SRG with outer regions forming a FCB yields more accurate approximations than both IJGP and Loopy BP.

Figure 3.16 shows the performance of GBP as a function of *iBound* for a fixed connectivity level. The *iBound* is increased from 2 (which is equivalent to Loopy BP) to 10 (which is exact). These plots show that GBP run on the partial tree-robust SRG is roughly equivalent to IJGP with *iBound* $= 7$, while GBP run on the FCB is equivalent to IJGP with *iBound* $= 6$. The fact that GBP run on both of these Loop-SRGs performs better than IJGP with *iBound* $= 6$ is quite remarkable considering that message passing on a Loop-SRG is computationally equivalent to IJGP with *iBound* $= 3$.

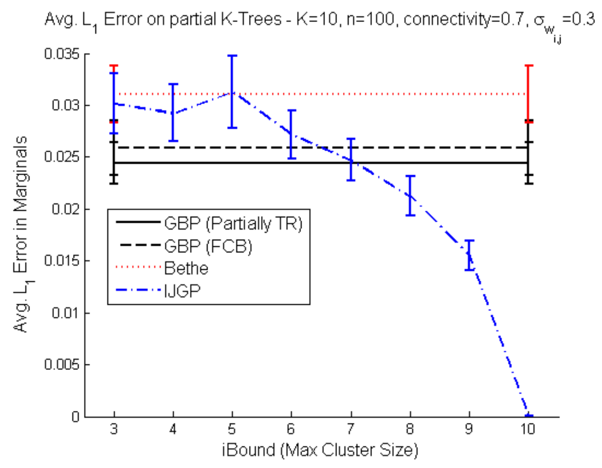


(a) $Error_{L_1}$ as a function of *connectivity*.

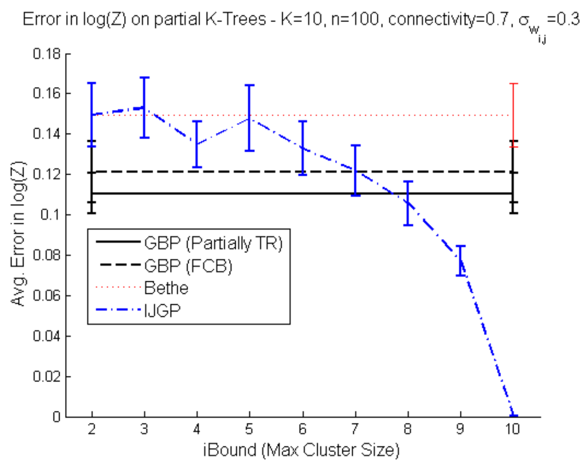


(b) $Error_Z$ as a function of *connectivity*.

Figure 3.15: Performance of GBP on the partial tree-robust construction, FCB construction and IJGP with $iBound = 3$ as a function of *connectivity*.



(a) $Error_{L_1}$ as a function of *iBound*.



(b) $Error_Z$ as a function of *iBound*.

Figure 3.16: Performance of the Loop-SRG constructions as a function of *iBound* with fixed *connectivity* of 0.7.

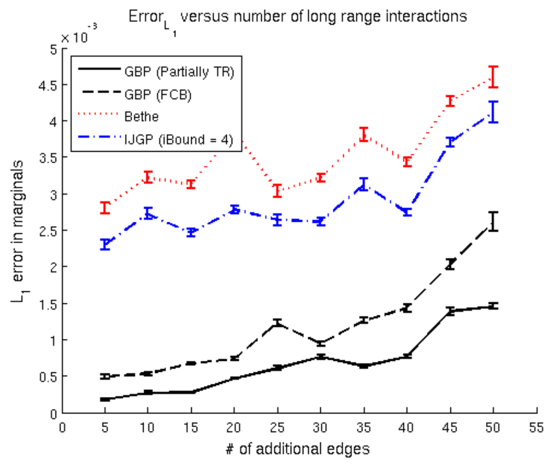
Grids with long range interactions

We also considered grid instances that are made non-planar by adding a number of edges between vertices that are non-adjacent in the standard 4-neighbor connectivity. These additional long range interactions were added via the following procedure. We begin with a 10×10 grid. Let G_0 denote this initial graph. Two vertices u and v are randomly chosen from the grid. If edge (u, v) exists in graph G_{i-1}, new vertices u and v are chosen randomly;

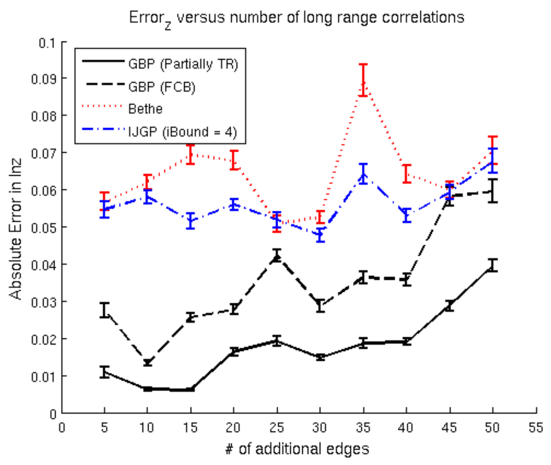
if edge (u, v) is not in G_{i-1}, then graph G_i is created by adding edge (u, v) to G_{i-1}. This process is repeated until a specified number of long range edges have been added to G_0.

Figure 3.17 compares the $Error_{L_1}$ and $Error_Z$ of GBP run on different loop SRG constructions. A total of 25 instances were generated each with $5, 10, ..., 50$ additional edges. In all 250 of these models the unary and pairwise terms were drawn with $\sigma_i = 1$ and $\sigma_{ij} = 0.5$. For the partially tree-robust SRG construction, we take the tree-robust core H to be G_0 and fill out the cycle basis using Algorithm 3.1 (as illustrated in Figure 3.10). As in the partial K-tree experiments, for the FCB construction we choose a fundamental basis that does not build upon the tree-robust core by using Algorithm 3.1 with $H = \emptyset$.

In Figure 3.17, we see that both the partially tree-robust and FCB constructions outperform IJGP with an equivalent $iBound = 3$. Interestingly, when adding 50 additional edges we do not see the $Error_{L_1}$ of the partially TR and the FCB constructions coalesce. This may be explained by the fact that even with 50 additional edges more than 60% of the loop outer regions in the SRG are from the tree-robust core (131 cycles in the basis, 81 of which come from the tree-robust core).



(a) $Error_{L_1}$ on grids with long range interactions.



(b) $Error_Z$ on grids with long range interactions.

Figure 3.17: Performance of the different Loop-SRG constructions as an increasing number of long range interactions are added to a 10×10 grid.

3.6 Discussion

Generalized Belief Propagation (GBP) is a broad class of approximate inference algorithms that extends Loopy BP by grouping the nodes of a graphical model into regions and iteratively passing messages between these regions. In this chapter, we provided some new guidance on how to choose the collection or regions (cliques, or clusters) defining a GBP algorithm. In particular, we connected the problem of choosing the loop outer regions of a Loop-Structured Region Graph (Loop-SRG) to that of finding a fundamental cycle basis of the undirected graph underlying a pairwise graphical model. We showed that choosing a collection of outer regions that comprise a fundamental cycle basis give GBP approximations that behave sensibly when the factors defining a model are extremely weak (non-Singularity) and extremely strong (over-counting number unity) and demonstrated the effectiveness of this recommendation empirically.

We then proposed a new criteria – tree-robustness – which is a refinement to the criterion that the loop regions form a fundamental basis. We offered a graph-theoretic characterization of the class of tree-robust cycle bases and identified tree-robust cycle bases for planar and complete graphs – two commonly occurring classes of pairwise models. This characterization helps explain the success of GBP on the *star* construction of [95] for complete graphs and the *all faces* construction on planar graphs. We also proposed an algorithm to automatically construct a high-performing Loop-SRG in pairwise models on arbitrary graph structures. The algorithm works by first identifying a tree-robust *core* and expanding the core to a full, fundamental cycle basis using an ear construction.

The experiments in this chapter confirm that GBP can yield very accurate approximations when the loop regions of a Loop-SRG form a fundamental basis and that these approximations can be further improved by choosing a fundamental basis that is at least partially tree robust. The criteria proposed in this chapter also lead to approximations that are comparable to IJGP run at a much higher computational complexity.

These findings open the door for much future work. Rather than simply finding a tree-robust subgraph, as is required in Algorithm 3.1, it would be preferable to have an algorithm that searches for tree-robust bases in a graph, or at the very least identifies when a graph does not admit a tree-robust basis. In addition, the recommendations made in this paper are purely structural in nature. A natural extension would be to incorporate interaction strengths into the search for suitable loop regions. In other words, search for (fundamental) cycle bases that are somehow weighted by the level of determinism along each cycle.

Finally, and perhaps most important, the current recommendations are only for models with pairwise interactions. A natural extension is to consider factor graphs or, more generally, region graphs. In this more general setting, it seems reasonable to require the collection of outer regions (or factors), $\{C_i\}$, to have the property that there exists an ordering π for which $C_{\pi(i)} \setminus \left\{C_{\pi(1)} \cup \cdots \cup C_{\pi(i-1)}\right\} \neq \emptyset$. In other words, the subsets can be ordered so that region $C_{\pi(i)}$ has some element that does not appear in any subset preceding it in the ordering. This definition is identical to that of a fundamental cycle basis (see definition 3.3) and will guarantee, through the reduction rules of [95], that the region graph is non-Singular. A next step would then be to define tree-robustness as a collection of regions that can be decomposed along some ordering if we do not allow certain elements that correspond to the cliques of an embedded junction tree to become unique. Again this is very similar to definitions 3.4 and 3.5 of tree-robust cycle bases. Whether these generalizations can be captured with mathematical structures as elegant as the theory of cycle spaces (or more generally matroids) remains to be seen.

Chapter 4

Investigating the Bottom-Up Approach to Learning and Prediction with Approximate Inference

In this chapter, we turn our attention to the task of learning graphical models with discrete random variables. A graphical model represents a joint distribution as a product of factors over subsets of variables. In the context of learning, these factors are parameterized functions and given some data one seeks to find a setting of the parameters that optimize some criterion.

We consider the joint estimation and prediction problem, where our desire is to first estimate the parameters of some discrete graphical model and then use the learned model to make predictions. For example, say we want to learn some model, $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$, that given an observed, noise-corrupted image, $\boldsymbol{x}$, can be used to recover the original, uncorrupted version of that image, $\boldsymbol{y}$. We can estimate the model parameters, $\boldsymbol{\theta}$, given some training data consisting of $(\boldsymbol{x}, \boldsymbol{y})$ pairs and then use the learned model to de-noise newly observed, noise-corrupted images. The approach we adopt in this chapter is to estimate the model parameters using the principle of maximum likelihood and make predictions using Bayesian decision theory.

Several difficulties arise when estimating the parameters of a discrete graphical model from data and using the learned model to make predictions. First, the model being learned may fail to capture important variable dependencies in the data. Such modeling difficulties ultimately diminish our ability to make accurate predictions on new data. Second, we face statistical difficulties because there may not be enough training data to accurately estimate the model parameters. This is often the case in supervised learning settings, such as image de-noising, where the training data is hand-labeled and expensive to collect. Finally, we face computational difficulties. Optimizing the likelihood criterion requires computing the log-partition function and its derivatives, both of which are computationally intractable. And, even after the model parameters have been estimated, making predictions under the learned model often requires intractable inference, such as computing the MAP configuration.

In this chapter, we examine the effect of using different approximate inference methods on the accuracy of both parameter estimation and prediction. In particular, we consider approximate inference methods that utilize a control parameter - the *iBound* - to trade computation for accuracy. In such methods, a smaller iBound requires less memory and time, but typically provides a worse approximation.

In theory, more accurate approximate inference leads to more accurate estimation of model parameters and ultimately better predictions. However, modeling and statistical issues confound our ability to learn a model that yields accurate predictions. As a result, it is not clear when, or even if, increasing the iBound computational limit will actually lead to better parameter estimates and predictions. This chapter is fundamentally about understanding the trade-off between the computational, modeling and statistical issues faced in the joint estimation and prediction setting.

The main contributions of this chapter are:

1. We formalize the computation-accuracy trade-off for likelihood-based learning of MRFs and CRFs.

2. We show that inference quality in our setting directly trades off bias and variance effects, and hence can lead to different quality estimates and predictions in different problem settings.

4.1 Learning & Prediction

Suppose that we are given a collection of independent and identically distributed (i.i.d) samples $\{\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(N)}\}$ from some distribution $p(\boldsymbol{y}; \boldsymbol{\theta}^{\star})$, where the true parameter setting $\boldsymbol{\theta}^{\star}$ is unknown. Given these samples, we seek to find a setting of $\boldsymbol{\theta}$ that makes the model $p(\boldsymbol{y}; \theta)$ as close as possible to the unknown distribution $p(\boldsymbol{y}; \boldsymbol{\theta}^{\star})$.

In this chapter we focus on finding a setting of the parameters that maximize the likelihood: meaning that the parameters are set so the training data has largest total probability under the model. More formally, for a model $p(\boldsymbol{y}; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^D$, the maximum likelihood estimate (MLE) is the point estimate

$$\boldsymbol{\theta}^{\text{ML}} = \operatorname*{arg\,max}_{\boldsymbol{\theta} \in \mathbb{R}^D} \ell_N(\boldsymbol{\theta}) = \operatorname*{arg\,max}_{\boldsymbol{\theta} \in \mathbb{R}^D} \frac{1}{N} \sum_{n=1}^{N} \log p(\boldsymbol{y}^{(n)}; \boldsymbol{\theta}). \tag{4.1}$$

Note that even though we write the likelihood as a function of the parameters, $\ell_N(\boldsymbol{\theta})$, it also depends on the choice of data set. Different collections of samples from the unknown distribution $p(\boldsymbol{y}; \boldsymbol{\theta}^{\star})$ will yield different likelihood functions and therefore different MLEs.

Often times we are interested in learning a model and then using it to make predictions on unseen data. In particular, we are given a dataset $\{(\boldsymbol{y}^{(1)}, \boldsymbol{x}^{(1)}), ..., (\boldsymbol{y}^{(N)}, \boldsymbol{x}^{(N)})\}$ of $(\boldsymbol{x}, \boldsymbol{y})$ pairs drawn i.i.d. from some true (unknown) distribution $p(\boldsymbol{y}, \boldsymbol{x}; \boldsymbol{\theta}^{\star})$. We use the training data to first estimate the parameters $\boldsymbol{\theta} \in \mathbb{R}^D$ of a model $p(\cdot;\boldsymbol{\theta})$ and then use the learned model to predict $\hat{\boldsymbol{y}}^{\text{tst}}$ given an observation $\boldsymbol{x}^{\text{tst}}$. An application that we consider later in the chapter is image de-noising, where the input, $\boldsymbol{x}$, is a noise-corrupted version of an image and the output, $\boldsymbol{y}$, is the original, noise-free version of the image. Our objective is thus to learn a model that can be used to de-noise a new, noisy image $\boldsymbol{x}^{\text{tst}}$.

Making a prediction requires access to the conditional distribution $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$. While it is possible to follow a generative approach that first estimates a joint distribution $p(\boldsymbol{y},\boldsymbol{x};\boldsymbol{\theta})$ and then converts it to a conditional distribution, such an approach requires modeling $p(\boldsymbol{y})$ which may be difficult in practice. For example, in the image de-noising application, $p(\boldsymbol{y})$ is a distribution over noise-free images. Discriminative approaches avoid this issue by directly estimating the conditional distribution $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$. To find such a model, we once again consider finding a setting of $\boldsymbol{\theta}$ that maximizes the likelihood. The *Conditional* MLE is the point estimate

$$\boldsymbol{\theta}^{ML} = \operatorname*{arg\,max}_{\boldsymbol{\theta}\in\mathbb{R}^D} \ell_N(\boldsymbol{\theta}) = \operatorname*{arg\,max}_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{n=1}^{N} \log p(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)};\boldsymbol{\theta}). \tag{4.2}$$

After finding the conditional MLE, we can appeal to Bayesian decision theory to make our predictions [74]. Bayesian decision theory requires the specification of a *loss* function, $\Delta(\hat{\boldsymbol{y}},\boldsymbol{y})$, that encodes how unhappy we are if we predict $\hat{\boldsymbol{y}}$ when the true output is $\boldsymbol{y}$. For example, if $\boldsymbol{y}$ is the original version of an image and $\hat{\boldsymbol{y}}$ is our predicted noise-free version, then we might be unhappy if we don't restore all of the pixels exactly. This would be encoded by the 0/1-loss function: $\Delta_{0/1}(\hat{\boldsymbol{y}},\boldsymbol{y}) = I\left[\hat{\boldsymbol{y}} \neq \boldsymbol{y}\right]$, where $I\left[\cdot\right]$ is the indicator function. Another possibility is that our level of unhappiness is proportional to the number of pixels that we incorrectly restore. This would be encoded by the Hamming loss function: $\Delta_{Hamming}(\hat{\boldsymbol{y}},\boldsymbol{y}) = \sum_i I\left[\hat{y}_i \neq y_i\right]$, where i sums over all the pixels in an image.

Given our choice of loss function, the optimal prediction will be the one that minimizes the expected loss [74]:

$$\boldsymbol{y}^\star = \arg\min_{\hat{\boldsymbol{y}}} E_{p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}^\star)} \left[\Delta(\hat{\boldsymbol{y}}, \boldsymbol{y})\right]. \tag{4.3}$$

Different inference tasks arise from several common choices of loss function. For example, if the 0/1-loss is used in (4.3), then the Bayes optimal predictor is the MAP assignment: $\boldsymbol{y}^\star = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}^\star)$. If the Hamming-loss is used in (4.3), then the Bayes optimal predictor is the max-marginal assignment: $y_i^\star = \arg\max_{y_i} p(y_i|\boldsymbol{x};\boldsymbol{\theta}^\star)$ for each pixel i, where $p(y_i|\boldsymbol{x};\boldsymbol{\theta}^\star)$ is the true marginal distribution[1].

The joint learning and prediction problem is illustrated in Figure 4.1. In the estimation phase, we find the maximum likelihood parameter setting, $\boldsymbol{\theta}^{ML}$, which provides an estimate, $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}^{ML})$, to the true conditional distribution, $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}^\star)$. Then in the prediction phase, we appeal to decision theory to incorporate our loss function, $\Delta(\hat{\boldsymbol{y}}, \boldsymbol{y})$, and devise a prediction rule $f(\boldsymbol{x};\boldsymbol{\theta}^{ML})$ that utilizes our parameter estimate. For example, if $\Delta(\hat{\boldsymbol{y}}, \boldsymbol{y})$ is the 0/1 loss, then $f(\boldsymbol{x},\boldsymbol{\theta}^{ML}) = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}^{ML})$ is the MAP prediction rule. Finally, when given a new input $\boldsymbol{x}^{\text{tst}}$ we apply the prediction rule to produce the prediction, $\hat{\boldsymbol{y}}^{\text{tst}}$. Rather than first estimating the parameters of a model and then using the learned model to make predictions – the classic approach – we note that it is also possible to minimize the loss function directly – the empirical loss minimization approach. In this chapter we focus on the classic approach; more detail on the latter approach can be found in [73].

Figure 4.1 nicely illustrates the two places where inference is needed. Finding the MLE, as we discuss in the next section, requires inference to compute the log-partition function and marginals of the model $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$. And predicting $\hat{\boldsymbol{y}}^{\text{tst}}$ requires inference in the model $p(\boldsymbol{y}|\boldsymbol{x}^{\text{tst}},\boldsymbol{\theta}^{ML})$, for example, to find the MAP assignment. Since exact inference is infeasible, approximate inference must be used in both the estimation and prediction phases.

[1]In practice we do not have access to the true conditional $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}^\star)$ and instead use the model $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$. If $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ does not match $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}^\star)$, the optimality of these prediction rules is no longer guaranteed.

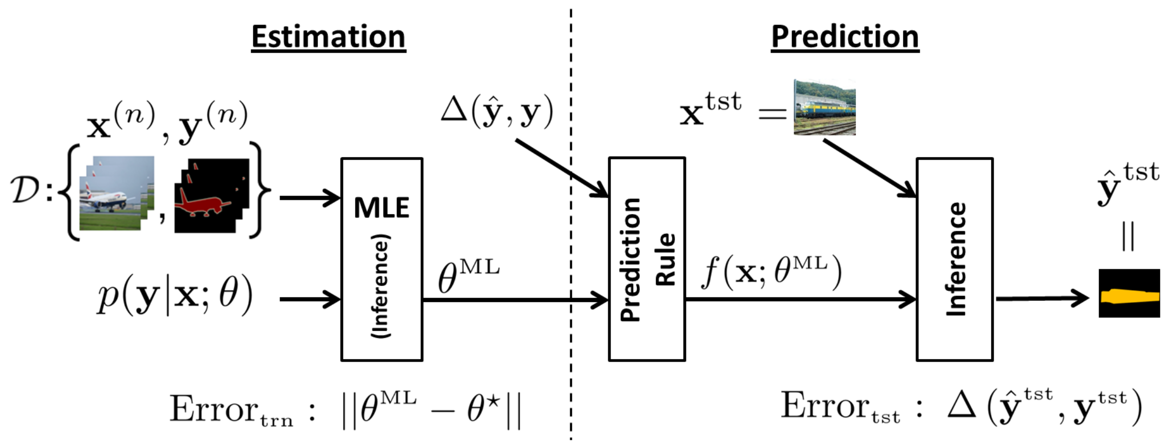

Estimation
Prediction
$\mathbf{x}^{(n)}, \mathbf{y}^{(n)}$
$\mathcal{D}$:
$p(\mathbf{y}|\mathbf{x};\theta)$
MLE
(Inference)
θ^{ML}
$\Delta(\hat{\mathbf{y}}, \mathbf{y})$
Prediction Rule
$\mathbf{x}^{\mathrm{tst}} =$
$f(\mathbf{x};\theta^{\mathrm{ML}})$
Inference
$\hat{\mathbf{y}}^{\mathrm{tst}}$
$||$
$\mathrm{Error}_{\mathrm{trn}} : ||\theta^{\mathrm{ML}} - \theta^{\star}||$
$\mathrm{Error}_{\mathrm{tst}} : \Delta(\hat{\mathbf{y}}^{\mathrm{tst}}, \mathbf{y}^{\mathrm{tst}})$

Figure 4.1: The joint learning and prediction problem.

We focus in this chapter on approximate inference methods that utilize a control parameter to trade computation for accuracy. Such methods provide a family of approximations to the MLE that have different computational requirements and ultimately yield different predictions on the test set. A primary objective of this chapter is to clarify what computational limit should be imposed during estimation – i.e., is it beneficial to find an approximation to the MLE using an expensive, but accurate method over a cheap, but inaccurate method.

In Section 4.2 we discuss several different approximations to the likelihood function. Then in Section 4.3 we analyze the error introduced by our choice of likelihood approximation and propose a framework for comparing this source of training error to both statistical and modeling sources of error. Finally, in Sections 4.4 and 4.5 we put this theory into practice and empirically study the impact of different likelihood approximations on both estimation and test error.

Before concluding this opening section, we note that the joint estimation and prediction problem has been studied by Wainwright [87]. He showed that for convex variational relaxations, it is best to *match* the approximate inference method used in learning to the approximate method used in prediction. For example, if in the prediction phase we plan to approximate the max-marginal assignment using the Tree Re-Weighted (TRW) method, then in the estimation phase we should use TRW to approximate the log-partition function and marginals as well. The intuition behind this finding is that even though approximate methods can lead to errors in both estimation and prediction, the errors will partially cancel each other.

4.2 Likelihood-Based Learning of Graphical Models

We begin this section by introducing the MLE problem for MRFs and CRFs. These optimization problems have no closed form solution, meaning that we are forced to use numerical methods to find the MLE. The likelihood is a smooth, concave function that is well-suited for gradient-based optimization. However, its gradients cannot be computed efficiently. Thus, in Sections 4.2.2 and 4.2.3 we review a few important approximations to the likelihood function.

4.2.1 The Likelihood and Conditional Likelihood

Recall from Section 1.2.1 that an MRF $p(\boldsymbol{y}) = \frac{1}{Z}\prod_\alpha \psi_\alpha(\boldsymbol{y}_\alpha)$ can be written in exponential family form as

$$p(\boldsymbol{y}|\boldsymbol{\theta}) = \exp\left(\boldsymbol{\theta} \cdot \boldsymbol{s}(\boldsymbol{y}) - \log Z(\boldsymbol{\theta})\right) \tag{4.4}$$

where $\boldsymbol{s}(\boldsymbol{y})$ is vector of sufficient statistics, $\boldsymbol{s}(\boldsymbol{y}) = \{I\left[\boldsymbol{Y}_\alpha = \boldsymbol{y}_\alpha\right] \mid \forall \alpha, \boldsymbol{y}_\alpha\}$, with an indicator function for every configuration of each factor and where $\boldsymbol{\theta}$ is a vector of parameters with components $\theta_\alpha(\boldsymbol{y}_\alpha) = \log \psi_\alpha(\boldsymbol{y}_\alpha)$ corresponding to each configuration in $\boldsymbol{s}(\boldsymbol{y})$.

Plugging (4.4) into $\ell_N(\boldsymbol{\theta})$ gives the following expression for the likelihood of an MRF:

$$\ell_N^{\text{MRF}}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} \log p(\boldsymbol{y}^{(n)}; \boldsymbol{\theta}) = \bar{\boldsymbol{\mu}}_N \cdot \boldsymbol{\theta} - \log Z(\boldsymbol{\theta}), \tag{4.5}$$

where $\bar{\boldsymbol{\mu}}_N = \frac{1}{N}\sum_n \boldsymbol{s}(\boldsymbol{y}^{(n)})$ is a vector of empirical marginals with components computed from the training data as

$$\bar{\mu}_N(\boldsymbol{Y}_\alpha = \boldsymbol{y}_\alpha) = \frac{1}{N}\sum\nolimits_n I\left[\boldsymbol{Y}_\alpha^{(n)} = \boldsymbol{y}_\alpha\right].$$

The likelihood function in (4.5) cannot be optimized in closed form because of the log-partition function's non-linear and non-trivial dependence on $\boldsymbol{\theta}$. However, $\ell_N^{\mathrm{MRF}}(\boldsymbol{\theta})$ is a concave function of $\boldsymbol{\theta}$: the first term is linear in $\boldsymbol{\theta}$ and, as we have already seen, the log-partition function, $\log Z(\boldsymbol{\theta})$, is a convex function of $\boldsymbol{\theta}$. Thus, the MLE can be found by standard numerical optimization methods if we can evaluate $\ell_N^{\mathrm{MRF}}(\boldsymbol{\theta})$ and its gradients. Evaluating $\ell_N^{\mathrm{MRF}}(\boldsymbol{\theta})$ requires computing the log partition function and its derivatives are:

$$\frac{\partial \ell_N^{\mathrm{MRF}}(\boldsymbol{\theta})}{\partial \theta_\alpha(\boldsymbol{y}_\alpha)} = \bar{\mu}_N(\boldsymbol{y}_\alpha) - \mu(\boldsymbol{y}_\alpha; \boldsymbol{\theta}), \tag{4.6}$$

where $\mu(\boldsymbol{y}_\alpha; \boldsymbol{\theta})$ is equal to the marginal probability of $\boldsymbol{y}_\alpha$ under the model $p(\boldsymbol{y}; \boldsymbol{\theta})$ (see e.g. Section 1.14 for a discussion of mean parameterization of expoential families).

Conditional models have an analogous form. A CRF $p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})}\prod_\alpha \psi_\alpha(\boldsymbol{y}_\alpha, \boldsymbol{x})$ can be written in exponential family form as

$$p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta}) = \exp\left(\boldsymbol{\theta} \cdot \boldsymbol{s}(\boldsymbol{y}, \boldsymbol{x}) - \log Z(\boldsymbol{\theta}, \boldsymbol{x})\right), \tag{4.7}$$

where the distribution over $\boldsymbol{y}$ is now determined by both the input $\boldsymbol{x}$ and the parameters $\boldsymbol{\theta}$. The parameter vector $\boldsymbol{\theta}$ is no longer indexed by components corresponding to configurations $I\left[\boldsymbol{Y}_\alpha = \boldsymbol{y}_\alpha\right]$ as in the MRF. Instead, the vector $\boldsymbol{\theta}$ may index sufficient statistics that are rich functions of both $\boldsymbol{y}$ and $\boldsymbol{x}$. For example, in the image denoising problem the d^{th} component of $\boldsymbol{\theta}$ might correspond to a function $s_d(\boldsymbol{y}, \boldsymbol{x}) = \sum_i I\left[y_i = x_i\right]$ that counts the number of pixels in which the input x_i matches the output y_i.

Plugging (4.7) into $\ell_N(\boldsymbol{\theta})$ gives the following expression for the likelihood of a CRF:

$$\ell_N^{\text{CRF}}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} \log p(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)};\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{\theta}\cdot\boldsymbol{s}(\boldsymbol{y}^{(n)},\boldsymbol{x}^{(n)}) - \log Z(\boldsymbol{\theta},\boldsymbol{x}^{(n)}), \tag{4.8}$$

where the log-partition function, $\log Z(\boldsymbol{\theta},\boldsymbol{x}^{(n)})$, is dependent on data point n. The conditional likelihood $\ell_N^{\text{CRF}}(\boldsymbol{\theta})$ is also a concave function of $\boldsymbol{\theta}$ and can be optimized by standard numerical methods if we can evaluate its gradients, which have components

$$\frac{\partial \ell_N^{\text{CRF}}(\boldsymbol{\theta})}{\partial \theta_d} = \frac{1}{N}\sum_{n=1}^{N} s_d(\boldsymbol{y}^{(n)},\boldsymbol{x}^{(n)}) - \frac{1}{N}\sum_{n=1}^{N}\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}^{(n)};\boldsymbol{\theta}) s_d(\boldsymbol{y},\boldsymbol{x}^{(n)}). \tag{4.9}$$

Unfortunately, gradient-based optimization methods cannot immediately be used to optimize the likelihood $\ell_N^{\text{MRF}}(\boldsymbol{\theta})$ or conditional likelihood $\ell_N^{\text{CRF}}(\boldsymbol{\theta})$ because the log-partition function and marginals are intractable. In the next section, we consider using approximate inference to estimate these quantities. Then in Section 4.2.3 we discuss the pseudo-likelihood and composite likelihood that replace the intractable likelihood function with tractable alternatives.

4.2.2 Approximate Inference and the Surrogate Likelihood

As the gradient of the likelihood can be written in terms of marginals, a reasonable idea would be to approximate these marginals (and the actual likelihood gradient) using an algorithm such as Weighted Mini-Bucket (WMB). The use of approximate methods can be put on stronger theoretical ground, however, by viewing them from a variational perspective. The approximate marginals found by a variational approximation, such as WMB, are in fact the exact gradient of an approximate log-partition function. As a result, we can think of replacing the true log-partition function with a *surrogate*, approximate log-partition function and ultimately optimizing a *surrogate* to the likelihood function.

This perspective on approximate inference-based estimation and the term "surrogate likelihood" are due to Wainwright [87]. We briefly review the variational representation of the likelihood function and then discuss the surrogate likelihood functions that arise from using the WMB and GBP approximations.

Recall from Section 1.4.2 that the log-partition function can be written as a constrained optimization problem [90]:

$$\log Z(\boldsymbol{\theta}) = \max_{\boldsymbol{\mu} \in \mathcal{M}} \left[\boldsymbol{\theta} \cdot \boldsymbol{\mu} + H(\boldsymbol{\mu})\right], \tag{4.10}$$

where $\mathcal{M} = \{\mu' \mid \exists \boldsymbol{\theta},\ \boldsymbol{\mu}' = \boldsymbol{\mu}(\boldsymbol{\theta})\}$ is the marginal polytope and $H(\boldsymbol{\mu})$ is the entropy of the distribution $p(\boldsymbol{y}; \boldsymbol{\theta}(\boldsymbol{\mu}))$ that produces the marginals $\boldsymbol{\mu}$.

Now consider an approximation to the optimization problem in (4.10)

$$\log \tilde{Z}(\boldsymbol{\theta}) = \max_{\boldsymbol{\mu} \in \tilde{\mathcal{M}}} \left[\boldsymbol{\theta} \cdot \boldsymbol{\mu} + \tilde{H}(\boldsymbol{\mu})\right], \tag{4.11}$$

where $\tilde{\mathcal{M}}$ is some relaxation of the marginal polytope and $\tilde{H}$ some approximation to the exact entropy H. As discussed in Section 1.4.2, Generalized Belief Propagation (GBP) and Weighted Mini-Bucket (WMB) arise from specific choices of $\tilde{\mathcal{M}}$ and $\tilde{H}$. The first derivative of the *surrogate* log-partition function, $\log \tilde{Z}(\boldsymbol{\theta})$, yields the approximate marginals

$$\tilde{\boldsymbol{\mu}}(\boldsymbol{\theta}) = \frac{\boldsymbol{d} \log \tilde{Z}(\boldsymbol{\theta})}{\boldsymbol{d\theta}} = \arg\max_{\boldsymbol{\mu} \in \tilde{\mathcal{M}}} \left[\boldsymbol{\theta} \cdot \boldsymbol{\mu} + \tilde{H}(\boldsymbol{\mu})\right] \tag{4.12}$$

where we have used $\tilde{\boldsymbol{\mu}}$ to make a distinction between the vector of true marginals. Plugging (4.11) into 4.6 yields the *surrogate likelihood*,

$$\tilde{\ell}_N^{\text{MRF}}(\boldsymbol{\theta}) = \bar{\boldsymbol{\mu}}_N \cdot \boldsymbol{\theta} - \max_{\boldsymbol{\mu} \in \tilde{\mathcal{M}}} \left[\boldsymbol{\theta} \cdot \boldsymbol{\mu} + \tilde{H}(\boldsymbol{\mu})\right]. \tag{4.13}$$

Thus, we can see that using a variational method to find approximate marginals, $\tilde{\boldsymbol{\mu}}(\boldsymbol{\theta})$, can be seen as computing the exact gradient of the surrogate likelihood, $\tilde{\ell}_N^{\text{MRF}}(\boldsymbol{\theta})$. This is perhaps a stronger justification for using approximate inference to estimate the MLE than the heuristic argument that it approximates the true likelihood gradients, $\frac{d\ell_N^{\text{MRF}}(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$.

We now consider two approximations to $\log Z(\boldsymbol{\theta})$ that utilize a control parameter – the *iBound* – to trade computation for accuracy. The iBound was defined in Section 1.4.1 when the Mini-Bucket Elimination (MBE) and Weighted Mini-Bucket (WMB) algorithms were first introduced. The iBound limits the number of variables appearing in any intermediate message computations. In the remainder of this chapter we will use $\log \tilde{Z}(\boldsymbol{\theta}, i)$ to denote an approximation to $\log Z(\boldsymbol{\theta})$ with an iBound of i variables.

The Generalized Belief Propagation (GBP) Surrogate

Generalized Belief Propagation (GBP) [101, 102, 20] is a family of approximate inference methods introduced in Section 1.4.2. Recall that GBP approximates the log-partition function as

$$\log Z(\boldsymbol{\theta}) \approx \log \tilde{Z}^{\text{GBP}}(\boldsymbol{\theta}, i) = \max_{\boldsymbol{\mu} \in \mathcal{M}_{\text{GBP}}} \left[\boldsymbol{\mu} \cdot \boldsymbol{\theta} + \tilde{H}^{\text{GBP}}(\boldsymbol{\mu}, i) \right], \tag{4.14}$$

where $\tilde{H}(\boldsymbol{\mu}, i)$ is the Kikuchi entropy approximation defined in (1.44) and $\mathcal{M}_{\text{GBP}}$ is the GBP approximation to the marginal polytope defined in (1.45). We assume that GBP is defined on a collection of regions $\mathcal{R}(i)$ on subsets of at most i variables ($|\boldsymbol{y}_\gamma| \leq i$ for all $\gamma \in \mathcal{R}(i)$).

Plugging (4.14) into $\ell_n(\boldsymbol{\theta})$ gives the GBP surrogate to the likelihood:

$$\tilde{\ell}_N^{\text{GBP}}(\boldsymbol{\theta}) = \bar{\boldsymbol{\mu}}_N \cdot \boldsymbol{\theta} - \log \tilde{Z}^{\text{GBP}}(\boldsymbol{\theta}, i) = \bar{\boldsymbol{\mu}}_N \cdot \boldsymbol{\theta} - \max_{\boldsymbol{\mu} \in \mathcal{M}_{\text{GBP}}} \left[\boldsymbol{\mu} \cdot \boldsymbol{\theta} + \tilde{H}(\boldsymbol{\mu}, i) \right]. \tag{4.15}$$

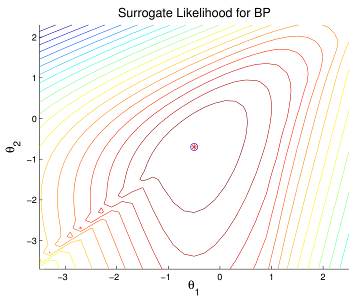
We wish to find a setting of $\boldsymbol{\theta}$ that maximizes (4.15). As noted in [41, 69], the GBP surrogate likelihood is a concave function of $\boldsymbol{\theta}$. This follows because $\log \tilde{Z}^{\text{GBP}}(\boldsymbol{\theta}, i)$ is a point-wise maximum over functions that are linear in $\boldsymbol{\theta}$ [12]. However, the log-partition function approximation, $\log \tilde{Z}^{\text{GBP}}(\boldsymbol{\theta}, i)$, need not be a concave function of $\boldsymbol{\mu}$. As a result, there may be many different vectors $\boldsymbol{\mu}$ that maximize (4.14) and small changes in the setting of $\boldsymbol{\theta}$ may lead to finding very different settings of $\boldsymbol{\mu}$. It is therefore unprincipled to use numerical routines that utilize line searches to optimize the GBP surrogate because basic continuity and differentiability assumptions are violated.

To remedy this issue, throughout this chapter we consider a "convexified" form of GBP discussed in [43, 64] that ignores any terms in the Kikuchi entropy approximation, $\tilde{H}(\boldsymbol{\mu}, i)$, with negative over-counting numbers. This restriction makes (4.14) a concave function of $\boldsymbol{\mu}$ and ensures that: 1) the approximate marginals $\boldsymbol{\mu}$ are a continuous function of the parameters $\boldsymbol{\theta}$; and 2) there is a unique optimum for every setting of $\boldsymbol{\theta}$. We note that the canonical parent-child message passing introduced in Section 1.4.2 is not guaranteed to converge on this "convexified" approximation. However, message passing can be made convergent by modifying the updates as described in [64].

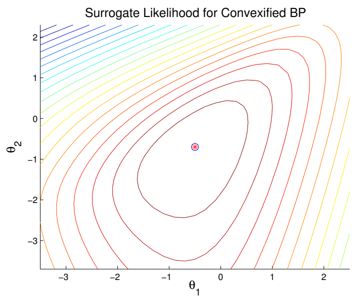
This effect of convexifying the GBP surrogate is illustrated in Figure 4.2, which depicts the surrogate likelihood functions arising from using GBP as well as the true likelihood function computed using exact inference. The likelihoods are computed for a statistically identifiable pairwise MRF on a 10×10 grid

$$p(\boldsymbol{y}; \theta_1, \theta_2) \propto \exp\left(\sum_{ij} \theta_1 I\left[|y_i - y_j| = 1\right] + \theta_2 I\left[|y_i - y_j| = 2\right]\right), \tag{4.16}$$

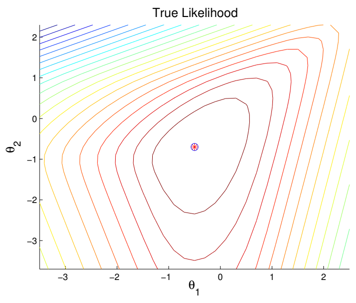
where $\theta_1 = -0.5$ and $\theta_2 = -0.7$ are the two model parameters and $y_i \in \{0, 1, 2\}$ are ternary variables. Notice that the BP surrogate likelihood in Figure 4.2a has several local-optima (e.g. at $\theta_1 = \theta_2 = -2.2$), while the convexified BP surrogate in Figure 4.2b and the true likelihood in Figure 4.2c do not.



(a) BP-based Surrogate



(b) Convexified BP Surrogate



(c) True Likelihood

Figure 4.2: Comparison of GBP-based surrogate likelihoods and the true likelihood.

Finding the approximate MLE, $\tilde{\boldsymbol{\theta}}$, using the GBP surrogate results in the following saddle-point optimization problem

$$\tilde{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\mathbb{R}^D}{\arg\max} \left[\bar{\boldsymbol{\mu}}_N \cdot \boldsymbol{\theta} - \max_{\boldsymbol{\mu}\in\mathcal{M}_{\text{GBP}}} \left[\boldsymbol{\mu}\cdot\boldsymbol{\theta} + \tilde{H}(\boldsymbol{\mu}, i)\right]\right] = \underset{\boldsymbol{\theta}\in\mathbb{R}^D}{\arg\max} \min_{\boldsymbol{\mu}\in\mathcal{M}_{\text{GBP}}} \bar{\boldsymbol{\mu}}_N\cdot\boldsymbol{\theta} - \boldsymbol{\mu}\cdot\boldsymbol{\theta} - \tilde{H}(\boldsymbol{\mu}, i). \tag{4.17}$$

Optimizing (4.17) typically involves a "double-loop" procedure: in the outer loop we update the parameters $\boldsymbol{\theta}$ by steepest ascent or second order method and in the inner loop we minimize over $\boldsymbol{\mu}$ given the current setting of $\boldsymbol{\theta}$. This exposes another practical issue. The approximate marginals $\boldsymbol{\mu}$ computed in the inner loop are typically found by iteratively passing messages. To expedite the entire learning procedure, we often run message passing to some loose convergence and thus fail to solve the optimization problem in (4.14) exactly[2]. Moreover, the approximate marginals computed by approximate inference may be non-deterministic if message initialization and passing schedules are randomized. These issues could result in estimating likelihood gradients that are not an ascent direction and could further confuse line searches. In our experience, however, these issues do not affect the operation of numerical routines.

[2]For a further discussion of "truncated fitting" procedures see [23, 24].

The Weighted Mini-Bucket (WMB) Surrogate

Weighted Mini-Bucket Elimination (WMB) [61, 21] is another family of approximate inference methods introduced in Section 1.4.2. Given some elimination order π and a partitioning of the functions assigned to each bucket, WMB provides an upper bound on the log-partition function

$$\log Z(\boldsymbol{\theta}) \leq \max_{\bar{\boldsymbol{\mu}} \in \mathcal{M}(\bar{G})} \left[\bar{\boldsymbol{\mu}} \cdot \bar{\boldsymbol{\theta}} + \tilde{H}_{\bar{\boldsymbol{w}}}(\bar{\boldsymbol{\mu}}) \right], \tag{4.18}$$

where $\bar{G}$ is the split graph with an expanded set of variables $\bar{\boldsymbol{y}}$, $\bar{\boldsymbol{\theta}}$ is an extended set of factors defined over $\bar{\boldsymbol{y}}$, $\bar{\boldsymbol{\mu}}$ is an expanded set of approximate marginals and $\bar{\boldsymbol{w}}$ is a collection of positive weights. The entropy approximation $\tilde{H}_{\bar{\boldsymbol{w}}}(\bar{\boldsymbol{\mu}})$ was defined in (1.51) and is a weighted sum of conditional entropies.

Plugging (4.18) into the expression for $\ell_N^{\text{MRF}}(\boldsymbol{\theta})$ in (4.6) gives us an expression for the WMB surrogate log-likelihood. However, as mentioned in Section 1.4.2, it is far more convenient to optimize the primal form of the WMB bound than its dual form. As we now explain, it happens that optimizing the form of the WMB surrogate likelihood involving the primal bound is easier than the dual form too.

Recall from (1.50) that the primal WMB bound takes the following form

$$\log Z(\boldsymbol{\theta}) \leq \log \tilde{Z}^{\text{WMB}}(\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{w}}) = \log \sum_{\bar{\boldsymbol{y}}_{\bar{m}}}^{\bar{\boldsymbol{w}}_{\bar{m}}} \cdots \sum_{\bar{\boldsymbol{y}}_1}^{\bar{\boldsymbol{w}}_1} \prod_{\alpha \in F} \exp(\bar{\theta}_\alpha(\bar{\boldsymbol{y}}_\alpha)) \tag{4.19}$$

where $\sum_{\bar{\boldsymbol{y}}_i}^{\bar{\boldsymbol{w}}_i}$ is the weighted summation operator (see (1.29) for a definition). The tightest upper bound is

$$\min_{\bar{\boldsymbol{\theta}} \in \mathcal{D}(\bar{\boldsymbol{\theta}}), \bar{\boldsymbol{w}} \in \mathcal{D}(\bar{\boldsymbol{w}})} \log \tilde{Z}^{\text{WMB}}(\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{w}}), \tag{4.20}$$

where $\mathcal{D}(\bar{\boldsymbol{\theta}}) = \{\bar{\boldsymbol{\theta}} \mid \bar{\boldsymbol{\theta}}(\boldsymbol{y}) = \boldsymbol{\theta}(\boldsymbol{y}),\ \text{when } y_i^r = y_i\ \forall i,\ R_i\}$ is the set of valid reparameterizations of $\boldsymbol{\theta}$ and $\mathcal{D}(\bar{\boldsymbol{w}}) = \{\bar{\boldsymbol{w}} \mid \sum_{r=1}^{R_i} w_i^r = 1,\ \forall i\}$ is the set of valid weights.

Plugging (4.20) into $\ell_N(\boldsymbol{\theta})$ gives the primal form of the WMB surrogate likelihood

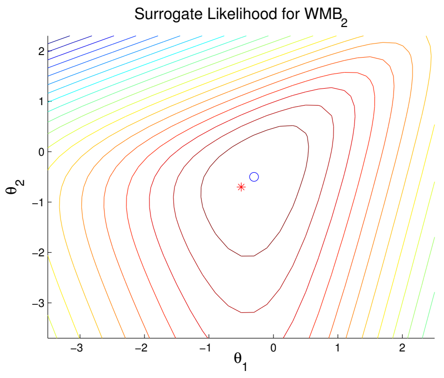
$$\tilde{\ell}_N^{\text{WMB}}(\boldsymbol{\theta}) = \bar{\boldsymbol{\mu}}_N \cdot \boldsymbol{\theta} - \min_{\bar{\boldsymbol{\theta}} \in \mathcal{D}(\bar{\boldsymbol{\theta}}), \bar{\boldsymbol{w}} \in \mathcal{D}(\bar{\boldsymbol{w}})} \log \tilde{Z}^{\text{WMB}}(\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{w}}). \tag{4.21}$$

Since the WMB log-partition function upper bounds the true partition function, it follows that the WMB surrogate likelihood lower bounds the true likelihood. Finding the ML estimate using the WMB surrogate involves the joint maximization

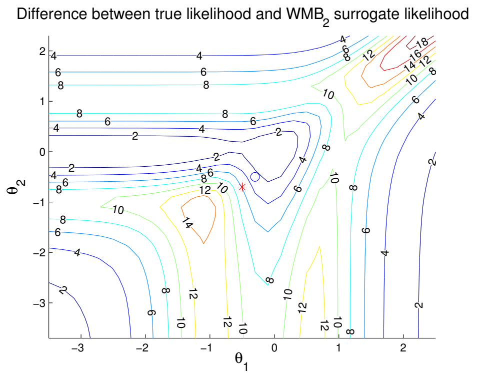
$$\operatorname*{arg\,max}_{\boldsymbol{\theta} \in \mathbb{R}^D} \quad \max_{\bar{\boldsymbol{\theta}} \in \mathcal{D}(\boldsymbol{\theta}), \bar{\boldsymbol{w}} \in \mathcal{D}(\boldsymbol{w})} \bar{\boldsymbol{\mu}}_N \cdot \boldsymbol{\theta} - \log \tilde{Z}^{\text{WMB}}(\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{w}}). \tag{4.22}$$

which is in a much nicer form than the saddle-point problem faced by GBP in (4.17). The weighted log-partition function, $\log \tilde{Z}^{\text{WMB}}$, is a differentiable and jointly convex function of $\bar{\boldsymbol{\theta}}$ and $\bar{\boldsymbol{w}}$. We can therefore update the weights, $\bar{\boldsymbol{w}}$, and/or update the extended parameters, $\bar{\boldsymbol{\theta}}$, in every iteration. Further, there is also no need to completely tighten the primal bound by exactly solving the problem in (4.20).

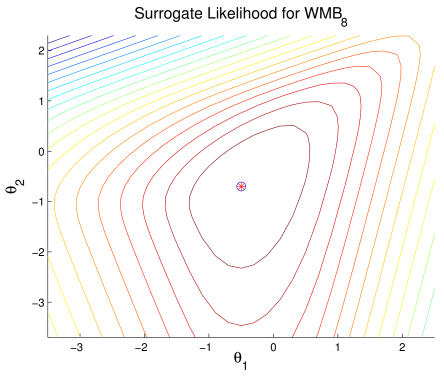
Figure 4.3 illustrates the WMB surrogate likelihood for the two parameter MRF described in (4.16) of the previous section. Plots 4.3a and 4.3c show the surrogate likelihood surfaces for iBounds of 2 and 8, respectively. Figures 4.3b and 4.3d illustrate the difference between the true likelihood function and the WMB-based surrogates. Notice that all the differences are positive because the WMB surrogate likelihood lower bounds the true likelihood. Also notice that the difference between the true likelihood and the iBound 2-based surrogate (Figure 4.3b) are larger than the differences involving the iBound 8-based surrogate (Figure 4.3b). In this problem, the better surrogate results in an optimum (show by the red asterisk) that is closer to the true optima (shown by the blue circle).



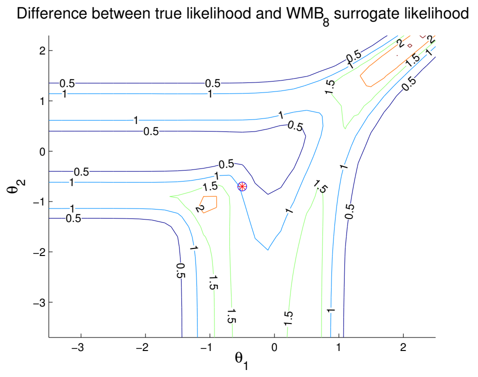
(a) WMB-based Surrogate with $iBound = 2$



(b) Likelihood difference: $\ell(\boldsymbol{\theta}) - \ell^{\mathrm{WMB}_2}(\boldsymbol{\theta})$



(c) WMB-based Surrogate with $iBound = 8$



(d) Likelihood difference: $\ell(\boldsymbol{\theta}) - \ell^{\mathrm{WMB}_8}(\boldsymbol{\theta})$

Figure 4.3: Comparison of WMB-based likelihood surrogates and the true likelihood.

4.2.3 Pseudo and Composite Likelihood Approximations

The fundamental problem encountered when optimizing the likelihood is intractability of the likelihood gradient. The approach adopted in the previous section was to approximate the likelihood gradient via approximate inference. Fortunately, the use of approximate inference could be justified by interpreting it as exactly optimizing an approximation, or surrogate, to the true likelihood function. In this section, we consider methods that directly replace the likelihood function with alternatives that can be tractably optimized.

The Pseudo-Likelihood Approximation

We begin by discussing the pseudo-likelihood, which was first introduced by Besag [7]. The pseudo-likelihood approximation can be described in many ways. We motivate it by expanding the joint distribution $p(\boldsymbol{y})$ using the chain rule and then approximating the terms in the chain rule expansion by adding extra conditioning variables

$$p(\boldsymbol{y};\boldsymbol{\theta}) = \prod_j p(y_j|y_{j-1},...,y_1;\boldsymbol{\theta}) \approx \prod_j p(y_j|y_1,...,y_{j-1},y_{j+1},...,y_M;\boldsymbol{\theta}) = \prod_j p(y_j|\boldsymbol{y}_{\neg j};\boldsymbol{\theta}). \tag{4.23}$$

In the case of an MRF, the terms $p(y_j|\boldsymbol{y}_{\neg j};\boldsymbol{\theta})$ take a particularly nice form as y_j is independent of the other variables in $\boldsymbol{y}$ given its neighbors $\delta(j)$

$$p(y_j|\boldsymbol{y}_{\neg j};\boldsymbol{\theta}) = p(y_j|\boldsymbol{y}_{\delta(j)};\boldsymbol{\theta}) = \frac{p(y_j,\boldsymbol{y}_{\delta(j)};\boldsymbol{\theta})}{\sum_{y'_j} p(y'_j,\boldsymbol{y}_{\delta(j)};\boldsymbol{\theta})} = \exp\left(\sum_{\alpha:y_j\in\boldsymbol{y}_\alpha} \theta_\alpha(\boldsymbol{y}_\alpha) - \log Z(\boldsymbol{y}_{\delta(j)};\boldsymbol{\theta})\right). \tag{4.24}$$

Note that $Z(\boldsymbol{y}_{\delta(j)};\boldsymbol{\theta}) = \sum_{y_j} \exp\left(\sum_{\alpha:y_j\in\boldsymbol{y}_\alpha} \theta_\alpha(\boldsymbol{y}_\alpha)\right)$ is a *local* partition function that is efficiently computable – it is just a sum over the states of variable y_j.

The pseudo-likelihood function is revealed by plugging the approximation in (4.23) into the expression for the likelihood

$$\begin{aligned}\ell_N^{\text{PL}}(\boldsymbol{\theta}) =& \frac{1}{N}\sum_{n=1}^{N}\sum_j \log p(y_j^{(n)}|\boldsymbol{y}_{N(j)}^{(n)};\boldsymbol{\theta}) \\ =& \sum_j \sum_{\alpha:y_j\in\boldsymbol{y}_\alpha} \sum_{\boldsymbol{y}_\alpha} \bar{\mu}_N(\boldsymbol{y}_\alpha)\theta_\alpha(\boldsymbol{y}_\alpha) - \sum_j \sum_{\boldsymbol{y}_{\delta(j)}} \bar{\mu}_N(\boldsymbol{y}_{\delta(j)}) \log Z(\boldsymbol{y}_{\delta(j)};\boldsymbol{\theta}),\end{aligned} \tag{4.25}$$

where once again $\bar{\mu}_N(\cdot)$ are empirical marginals computed on the training set. The pseudo-likelihood is also a concave function of $\boldsymbol{\theta}$ and all of the terms in (4.25) are tractable. The max pseudo-likelihood estimate $\boldsymbol{\theta}^{\text{PL}} = \arg\max_{\boldsymbol{\theta}} \ell_N^{\text{PL}}(\boldsymbol{\theta})$ can be found by gradient-based optimization. We refer the reader to [45] for details on optimizing $\ell^{\text{PL}}(\boldsymbol{\theta})$.

The pseudo-likelihood is a well-studied object and it is known to be statistically *consistent* [30, 45]: meaning that if the parameters of the true (unknown) distribution $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$ are in some parametric family $\boldsymbol{\theta}^\star \in \Theta$ and our model happens to be in the same family $\boldsymbol{\theta} \in \Theta$, then the max pseudo-likelihood estimate, $\boldsymbol{\theta}^{\text{PL}}$, converges to the true parameter setting $\boldsymbol{\theta}^{\text{PL}} \to \boldsymbol{\theta}^\star$ as the size of the training data is increased $N \to \infty$ [3]. The intuition for this result is that the max pseudo-likelihood estimate, $\boldsymbol{\theta}^{\text{PL}}$, attempts to match all of the conditional distributions $p(y_j | \boldsymbol{y}_{\delta(j)}; \boldsymbol{\theta})$ to the empirical conditionals. If $\boldsymbol{\theta}^{\text{PL}}$ matches all of these empirical conditionals exactly, then running a Gibbs sampler on the model $p(\boldsymbol{y}; \boldsymbol{\theta}^{\text{PL}})$ will have the same stationary distribution as the true distribution $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$, meaning that $\boldsymbol{\theta}^{\text{PL}} = \boldsymbol{\theta}^\star$.

The Composite Likelihood Approximation

The composite likelihood can be seen as a direct generalization of the pseudo-likelihood [60, 22]. Rather than approximating the likelihood as a sum of terms involving single variable conditionals, $\log p(y_j | \boldsymbol{y}_{\delta(j)}; \boldsymbol{\theta})$, the composite likelihood considers terms over larger sets of variables. In particular, let $\{(\boldsymbol{y}_{A_c}, \boldsymbol{y}_{B_c})\}_{c=1}^{C}$ be a collection of disjoint subsets of $\boldsymbol{y}$ (i.e. $\boldsymbol{y}_{A_c} \cap \boldsymbol{y}_{B_c} = \emptyset$, $\boldsymbol{y}_{A_c}, \boldsymbol{y}_{B_c} \subseteq \boldsymbol{y}$) such that $\boldsymbol{y}_{A_c} \neq \emptyset$, then the composite likelihood is defined as:

$$\ell_N^{\text{CL}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} \log p(\boldsymbol{y}_{A_c} | \boldsymbol{y}_{B_c}; \boldsymbol{\theta}). \tag{4.26}$$

Note that when $\{\boldsymbol{y}_{A_c}\} = \{y_j\}_j$ and $\{\boldsymbol{y}_{B_c}\} = \{\boldsymbol{y} \setminus y_j\}_j$ we recover the pseudo-likelihood.

The max composite likelihood estimate $\boldsymbol{\theta}^{\text{CL}} = \arg\max_{\boldsymbol{\theta}} \ell_N^{\text{CL}}(\boldsymbol{\theta})$ can also be found by gradient-based optimization. The complexity of optimizing $\ell_N^{\text{CL}}(\boldsymbol{\theta})$ now depends on the size of the clus-

[3] Technically, for this to occur the model must be statistically *identifiable*.

ters $\boldsymbol{y}_{A_c}$. However, as the size of $\boldsymbol{y}_{A_c}$ is increased, the model will capture more inter-variable dependencies and hopefully provide a better estimate of $\boldsymbol{\theta}^{\mathrm{CL}}$. This computation-accuracy trade-off was explored in [22], where it was also shown that the composite likelihood is statistically *consistent* if a proper set of clusters are chosen[4]. In addition, maximum composite liklelihood estimation has been shown to be statistically more efficient than maximum pseudo-likelihood estimation [59].

The composite likelihood is quite flexible as one is free to select arbitrary subsets $\boldsymbol{y}_{A_c}$ and $\boldsymbol{y}_{B_c}$. Nowozin recently proposed two novel composite likelihoods for undirected graphical models [69]. His criss-cross likelihood is applicable to pairwise models on a grid and the sets $\{\boldsymbol{y}_{A_c}\}$ are chosen to be horizontal and vertical "strips" in the grid structure. He also proposed a composite likelihood for general graphs where the sets $\{\boldsymbol{y}_{A_c}\}$ come from very-acyclic subgraph decompositions[11] – thus ensuring that computing the conditional $p(\boldsymbol{y}_{A_c}|\boldsymbol{y}_{B_c};\boldsymbol{\theta})$ only requires inference on a chain.

4.3 Error in Approximate Likelihood Optimizations

The previous section discussed different methods for approximating the intractable likelihood computation. In general, these approximations will introduce error that prevent us from finding the MLE, $\boldsymbol{\theta}^{\mathrm{ML}}$. In most cases, however, other sources of error cause the model $p(\boldsymbol{y};\boldsymbol{\theta}^{\mathrm{ML}})$ to be an approximation to the true (unknown) distribution $p(\boldsymbol{y};\boldsymbol{\theta}^{\star})$ of interest. In this section, we present two frameworks for describing the distinct sources of error affecting parameter estimation. We utilize these frameworks in the following section to systematically study the effect that using different likelihood approximations has on our ability to accurately estimate $\boldsymbol{\theta}^{\star}$.

[4]The conditional distributions must be *in* the model and every variable must appear in some cluster $\boldsymbol{y}_{A_j}$.

4.3.1 Excess Error Decomposition

Recall that our data $\{\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(N)}\}$ is sampled i.i.d from some true (unknown) distribution $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$, where $\boldsymbol{\theta}^\star \in \Theta$ is a member of some parametric family of distributions. Let us further assume that the true probability model, $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$, is *statistically identifiable*[5] so that we can uniquely recover the parameter setting $\boldsymbol{\theta}^\star$ given large enough N.

The maximum likelihood estimate of $\boldsymbol{\theta}^\star$, given our set of N i.i.d. samples from $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$ is:

$$\boldsymbol{\theta}_N^{\text{ML}} = \arg\max_{\boldsymbol{\theta}\in\Theta} \frac{1}{N}\sum_{n=1}^{N} \log p(\boldsymbol{y}^{(n)}; \boldsymbol{\theta}), \tag{4.27}$$

where we now use a subscript N to make the point estimate's dependence on the choice of data set explicit. By the strong law of large numbers, as $N \to \infty$ the Monte-Carlo estimate converges to an expectation under $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$:

$$\frac{1}{N}\sum_{n=1}^{N} \log p(\boldsymbol{y}^{(n)}; \boldsymbol{\theta}) \overset{a.s.}{\to} \mathbb{E}_{\boldsymbol{\theta}^\star}\left[\log p(\boldsymbol{y}; \boldsymbol{\theta})\right] = \sum_{\boldsymbol{y}} p(\boldsymbol{y}; \boldsymbol{\theta}^\star) \log p(\boldsymbol{y}; \boldsymbol{\theta}). \tag{4.28}$$

Since we assumed that $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$ is statistically identifiable, $\boldsymbol{\theta}_N^{\text{ML}} \to \boldsymbol{\theta}^\star$ as $N \to \infty$. In other words, the maximum likelihood estimator is asymptotically consistent [13].

Unfortunately, the true parameter setting $\boldsymbol{\theta}^\star$ is unattainable for several practical reasons. First, we are often interested in learning models for which the true parametric family Θ is unknown. As a result, we often restrict attention to easy-to-specify model families $\underline{\Theta} \subset \Theta$. For example, in the image-denoising task, we often learn pairwise MRFs that model the interaction between adjacent pixels in an image, but ignore longer range interactions that may also be important. Let

$$\underline{\boldsymbol{\theta}}^\star = \arg\max_{\boldsymbol{\theta}\in\underline{\Theta}} \mathbb{E}_{\boldsymbol{\theta}^\star}\left[\log p(\boldsymbol{y}; \boldsymbol{\theta})\right] \tag{4.29}$$

[5]see Definition 1.1

be the best parameter setting in this restricted family of models.

Second, note that $\mathbb{E}_{\boldsymbol{\theta}^\star}[\cdot]$ is an expectation over the true and unknown distribution $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$. Since we usually don't have access to the marginals of $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$ we instead use our samples to compute an empirical approximation to $\mathbb{E}_{\boldsymbol{\theta}^\star}[\cdot]$. Let

$$\underline{\boldsymbol{\theta}}_N = \arg\max_{\boldsymbol{\theta} \in \underline{\Theta}} \frac{1}{N} \sum_n \log p(\boldsymbol{y}^{(n)}; \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta} \in \underline{\Theta}} \ell_N(\boldsymbol{\theta}) \tag{4.30}$$

be the empirical estimate in the restricted family of models.

Finally, as discussed in Section 4.2, it is often infeasible to optimize the true likelihood function $\ell_N(\boldsymbol{\theta})$. Instead, we optimize an approximation to it $\tilde{\ell}_N(\boldsymbol{\theta})$. Let

$$\tilde{\underline{\boldsymbol{\theta}}}_N = \arg\max_{\boldsymbol{\theta} \in \underline{\Theta}} \tilde{\ell}_N(\boldsymbol{\theta}) \tag{4.31}$$

be the empirical estimate found under our choice of approximate likelihood function. The optimization problem in (4.31) implicitly defines $\tilde{\underline{\boldsymbol{\theta}}}_N = \tilde{\underline{\boldsymbol{\theta}}}(\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(N)})$ as a function of the N observations. In other words, it defines an approximate inference-based estimator, where different approximate inference methods yield different estimators.

The expected excess error in our likelihood-based estimate, $\mathcal{E}$, is:

$$\mathcal{E} = \mathbb{E}\left[\mathbb{E}_{\boldsymbol{\theta}^\star}\left[\log p(\boldsymbol{y}; \boldsymbol{\theta}^\star)\right] - \mathbb{E}_{\boldsymbol{\theta}^\star}\left[\log p(\boldsymbol{y}; \tilde{\underline{\boldsymbol{\theta}}}_N)\right]\right], \tag{4.32}$$

where since $\tilde{\underline{\boldsymbol{\theta}}}_N$ is a function of the N observations, the outer-most expectation is taken with respect to the random choice of data set. We note that the quantity inside the expectation is precisely the KL-Divergence between $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$ and $p(\boldsymbol{y}; \tilde{\underline{\boldsymbol{\theta}}}_N)$.

Following [10], the excess error in our likelihood-based estimate can be decomposed as:

$$\mathcal{E} = \mathcal{E}_{\text{Model}} + \mathcal{E}_{\text{Estimation}} + \mathcal{E}_{\text{Optimization}} \tag{4.33}$$

where

- $\mathcal{E}_{\text{Model}} = \mathbb{E}\left[\mathbb{E}_{\boldsymbol{\theta}^\star}\left[\log p(\boldsymbol{y}; \boldsymbol{\theta}^\star)\right] - \mathbb{E}_{\boldsymbol{\theta}^\star}\left[\log p(\boldsymbol{y}; \underline{\boldsymbol{\theta}}^\star)\right]\right]$, is the model error. It measures how well the *best* model in $\underline{\Theta}$ can represent the true (unknown) model $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$.
- $\mathcal{E}_{\text{Estimation}} = \mathbb{E}\left[\mathbb{E}_{\boldsymbol{\theta}^\star}\left[\log p(\boldsymbol{y}; \underline{\boldsymbol{\theta}}^\star)\right] - \mathbb{E}_{\boldsymbol{\theta}^\star}\left[\log p(\boldsymbol{y}; \underline{\boldsymbol{\theta}}_N)\right]\right]$, is the estimation error. It measures the error due to optimizing an empirical likelihood using only N samples.
- $\mathcal{E}_{\text{Optimization}} = \mathbb{E}\left[\mathbb{E}_{\boldsymbol{\theta}^\star}\left[\log p(\boldsymbol{y}; \underline{\boldsymbol{\theta}}_N)\right] - \mathbb{E}_{\boldsymbol{\theta}^\star}\left[\log p(\boldsymbol{y}; \tilde{\underline{\boldsymbol{\theta}}}_N)\right]\right]$, is the optimization error. It measures the error introduced by our approximate likelihood function. It can be seen as a bias inherent to the approximate inference-based estimator.

Figure 4.4 depicts the excess error, $\mathcal{E}$, for the case when there is no model error. The empirical likelihood function, $\ell_N(\boldsymbol{\theta})$, is shown in red and takes its maximum at the point estimate $\boldsymbol{\theta}_N$. The empirical surrogate likelihood function, $\tilde{\ell}_N(\boldsymbol{\theta})$, is shown by the dashed red line and takes its maximum at the point estimate $\tilde{\underline{\boldsymbol{\theta}}}_N$. Finally, the solid black line depicts the expected likelihood, $\ell(\theta) = \mathbb{E}_{\boldsymbol{\theta}^\star}\left[\log p(\boldsymbol{y}; \boldsymbol{\theta})\right]$. It takes its maximum at the true parameter setting $\boldsymbol{\theta}^\star$. The empirical likelihood (solid red line) is a stochastic approximation to the expected likelihood (solid black line). The empirical surrogate likelihood (dashed red line) will lower bound the empirical likelihood function (solid red line) when using the WMB-surrogate.

We note that there is a fundamental trade-off between model error and estimation error: a larger family of models may reduce or even eliminate model error, but will require many additional samples to maintain a constant level of estimation error. As a result, if $\boldsymbol{\theta} \in \mathbb{R}^D$ and N is small relative to the degrees of freedom in the model, D, then one typically optimizes

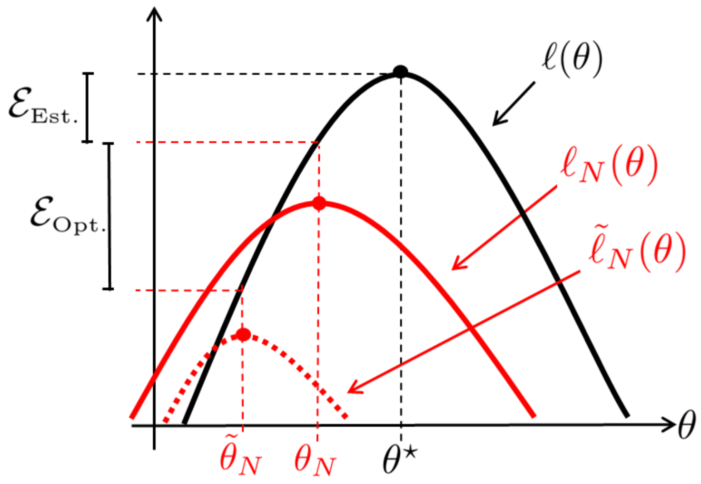


Figure 4.4: Illustration of excess error in the approximate likelihood-based estimate.

a regularized form of the (approximate) likelihood: $\arg\max_{\boldsymbol{\theta}\in\mathbb{R}^D} \tilde{\ell}_N(\boldsymbol{\theta}) + \lambda||\boldsymbol{\theta}||^2$, where $\lambda > 0$ controls the model-estimation error trade-off. We will keep the regularization parameter λ fixed throughout this chapter as doing so will allow us to study the behavior of different approximate inference-based estimators under varying levels of model and estimation error. In practice, one would certainly want to adjust λ given their particular choice of model, algorithm and data set.

4.3.2 Bias-Variance Decomposition

The excess error maps closeness in parameter setting to closeness in expected likelihood so that two parameter settings $\boldsymbol{\theta}^1$ and $\boldsymbol{\theta}^2$ are presumed to be close only if their expected likelihoods $\ell(\boldsymbol{\theta}^1)$ and are $\ell(\boldsymbol{\theta}^2)$ close. However, we can also measure error directly in terms of the difference between the true parameter setting, $\boldsymbol{\theta}^\star$, and the estimate produced by our approximate inference-based estimator, $\underline{\tilde{\boldsymbol{\theta}}}$. In Figure 4.4 this corresponds to measuring error on the horizontal, rather than vertical axis. The mean-squared error (MSE) describes precisely this error

$$\text{MSE}: \ \ \mathbb{E}\left[||\underline{\tilde{\boldsymbol{\theta}}} - \boldsymbol{\theta}^\star||^2\right] = Tr[Var(\underline{\tilde{\boldsymbol{\theta}}})] + ||Bias(\underline{\tilde{\boldsymbol{\theta}}}))||^2, \tag{4.34}$$

where once again the outer expectation is with respect to the random choice of data set. We have decomposed the MSE into a variance component and a bias component. The variance term measures how sensitive the estimator is to changes in the data set of N samples and is computed as: $Tr[Var(\underline{\tilde{\boldsymbol{\theta}}})] = \sum_{d=1}^{D} Var(\underline{\tilde{\boldsymbol{\theta}}}_d)$ for $\boldsymbol{\theta} \in \mathbb{R}^D$. The bias term measures whether the estimator $\underline{\tilde{\boldsymbol{\theta}}}(\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(N)})$ is correct on average and is computed as: $Bias(\underline{\tilde{\boldsymbol{\theta}}})) = \mathbb{E}(\underline{\tilde{\boldsymbol{\theta}}}) - \boldsymbol{\theta}^\star$.

While one normally considers the asymptotic properties of an estimator (e.g. consistency and efficiency) it is important to note that the bias and variance of an estimator are a function of sample size, N. An estimator $\boldsymbol{\theta}^\dagger = \boldsymbol{\theta}^\dagger(\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(N)})$ may be unbiased as $N \to \infty$, but exhibit large bias when N is small. In contrast, an estimator $\boldsymbol{\theta'} = \boldsymbol{\theta'}(\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(N)})$ may be asymptotically biased, but exhibit smaller bias than $\boldsymbol{\theta}^\dagger$ when N is small. The variance of an estimator will, in general, decrease as N grows, but the rate at which it decreases will differ among estimators. As a result, an unbiased estimator $\boldsymbol{\theta}^\dagger$ may be asymptotically efficient, meaning that it has the minimum variance among all possible estimators as $N \to \infty$, but exhibit larger variance than some asymptotically inefficient estimator, $\boldsymbol{\theta'}$, for small N.

Ideally, both the bias and variance of an estimator will be small at any N, but one usually has to increase bias in order to reduce variance (or vice-versa). In the experiments that follow, we will see exactly this trade-off: certain estimators will exhibit low variance and high bias, while others will exhibit high variance, but low bias. By studying the bias and variance characteristics of many different estimators, we are in a good position to select an estimator suited for our particular application.

4.4 Empirical Study of Inference-Based Estimation

We conducted a variety of experiments to study the effect of using different approximate inference methods and, therefore, different surrogate likelihood functions on the accuracy of parameter estimation. In particular, we focus on GBP and WMB as both methods utilize the iBound control parameter to trade computation for accuracy. In such methods, a smaller iBound requires less computation, but typically provides a less accurate approximation to the true likelihood function.

Since increasing iBound also increases computation, a natural question to ask is: how does the increased computation of a higher iBound manipulate the bias-variance trade-off and lead to higher quality parameter estimates? This of course depends on properties of the problem under consideration, but our study exposes regimes where it is beneficial to invest more computation. Interestingly, we also found regimes where it may be detrimental to use a higher iBound method.

4.4.1 Experimental Setup

We focus in this section on estimating the parameters of pairwise MRFs defined over many different underlying graph structures. Every variable in the MRF model is K-ary – i.e. $y_i \in \{1, .., K\}$ – and has an associated unary potential, with parameters sampled as $\theta_i(y_i) \sim \mathcal{N}(0, \sigma_i^2)$. Each edge in the model has a pairwise potential, with parameters sampled as $\theta_{ij}(y_i, y_j) \sim \mathcal{N}(0, \sigma_{ij}^2)$.

To ensure that the model is statistically identifiable, we force the pairwise potentials to be symmetric $\theta_{ij}(y_i, y_j) = \theta_{ij}(y_j, y_i)$ and set $\theta_i(y_i = 1) = 0$ and $\theta_{ij}(y_i = 1, y_j) = 0$ for $y_j \in \{1, .., K\}$[104]. For example, if $K = 3$ then the unary and pairwise potentials are as follows:

$$\boldsymbol{\theta}_i(y_i) = \begin{bmatrix} 0 \\ \theta_i(y_i = 2) \\ \theta_i(y_i = 3) \end{bmatrix} \quad \boldsymbol{\theta}_{ij}(y_i, y_j) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \theta_{ij}(y_i = 2, y_j = 2) & \theta_{ij}(y_i = 2, y_j = 3) \\ 0 & \theta_{ij}(y_i = 3, y_j = 2) & \theta_{ij}(y_i = 3, y_j = 3) \end{bmatrix}$$

with $\theta_{ij}(y_i = 2, y_j = 3) = \theta_{ij}(y_i = 3, y_j = 2)$.

After specifying the parameters of the unary and pairwise potentials, we generate a set of N samples from the model using a Gibbs sampler. In order to generate samples that are nearly i.i.d., we run the sampler for 5000 burn-in iterations and retain every 100^{th} sample. We then compute the sufficient statistics over the N samples and use Marc Schmidt's *minFunc* solver[6] to find the parameter setting that optimizes (4.31), where different approximate inference methods are used to specify different surrogate likelihoods, $\tilde{\ell}_N(\theta)$. The minFunc solver optimizes (4.31) using a quasi-Newton strategy, with limited-memory BFGS updates and a bracketing line search. Unless otherwise specified, the minFunc solver was run to a convergence threshold of $1e^{-6}$.

As mentioned in the previous section, we keep the regularization parameter fixed at $\lambda = 1e{-}6$ throughout these experiments. While one would typically set λ to an appropriate value for their learning task, in this chapter we are interested in studying the effect that different surrogate likelihoods have on estimation accuracy as both model and estimation error are systematically varied. In particular, we want to understand the bias and variance properties of the different inference-based estimators, which would undoubtedly be obscured if the regularization parameter were varied too. By keeping the regularization parameter fixed, it is important to note that concrete conclusions about the absolute performance of different estimators cannot be drawn.

We focus on surrogates based on the WMB and GBP inference methods. In particular, we

[6]http://www.di.ens.fr/ mschmidt/Software/minFunc.html

use the WMB method with iBound $= \{2, 4, 8\}$ and GBP with iBound $= \{2, 4\}$. The WMB-based estimators utilize the same, minimum induced width elimination order for all iBound settings. GBP with iBound 2 is just a convexified form of Loopy BP described in Section 4.2.2. The collection of regions $\mathcal{R}$ used by GBP with iBound 4 depends on the problem under consideration. In experiments involving a grid of variables, the set of regions $\mathcal{R}$ are comprised of the faces and all interior edges and vertices of the grid. In all other experiments, GBP takes the collection of regions to be $\mathcal{R} = \mathcal{O} \cup \mathcal{I}$, where the outer regions, $\mathcal{O}$, are chosen to be the cluster of variables in each mini-bucket used by WMB with an iBound of 4, and the inner regions, $\mathcal{I}$, are identified using the cluster-variation method discussed in Section 1.4.2.

4.4.2 Estimation and Optimization Error Comparison

We begin with a set of experiments for which $\underline{\Theta} = \Theta$ so there is no model error ($\mathcal{E}_{\text{model}} = 0$). This is an admittedly artificial setting as model error is unavoidable in practice, but it is useful because it allows us to examine the interplay between estimation and optimization error in some commonly used models and, ultimately, identify regimes in which a better (higher iBound) surrogate is beneficial.

Synthetic Grids: Varying Interaction Strength

We begin with a set of experiments on a 20×20 grid of binary variables in the standard 4-neighbor connectivity. The (identifiable) parameters $\boldsymbol{\theta}^{\star}$ of each pairwise model are sampled as described in Section 4.4.1. 8 different models were generated with unary potentials sampled using $\sigma_i = 0.1$ and pairwise potentials sampled using $\sigma_{ij} = \{0.5, 1.0, 2.0\}$, for a total of 24 different models. As the interaction strength, σ_{ij}, is increased, the models become increasingly frustrated and the accuracy of approximate inference will often degrade. This is thus an important setting to study as it tells us how the error of the different surrogates scales as inference in model being learned gets increasingly difficult.

Data sets of size $N = \{100, 250, 500, 1000, 2500, 5000, 10000\}$ were generated for each of the 24 models. We report the mean and standard deviation of the MSE, $||\tilde{\boldsymbol{\theta}}_N - \boldsymbol{\theta}^\star||^2$, across the 8 models for each setting of σ_{ij}. We also report the mean and standard deviation of the run-times of the WMB- and GBP-based estimators, which is the time in seconds required by the minFunc solver to find the maximum of each surrogate likelihood function. The run-time depends not only on the time complexity of each inference algorithm, but also the curvature of each algorithm's surrogate likelihood function and properties of the *minFunc* solver. As a result, it is possible for WMB with iBound 8 to have a smaller run-time than WMB with iBound 2 simply because the minFunc solver needs fewer evaluations to find the optima $\tilde{\boldsymbol{\theta}}_N^{\text{iBound}=8}$ than the optima $\tilde{\boldsymbol{\theta}}_N^{\text{iBound}=2}$.

Figure 4.5 shows the MSE and run-times for the three WMB-based and two GBP-based estimators. The left column contains the MSEs and the right column the run-times. The top, middle and bottom rows are for models generated with $\sigma_{ij} = 0.5$, $\sigma_{ij} = 1.0$ and $\sigma_{ij} = 2.0$, respectively. As expected, the size of the MSE for all estimators increases as we move from the top row to the bottom row. Interestingly, the run-times also increase as we move from top to bottom. This suggests that the curvature of the surrogates decreases as the models become increasingly frustrated, causing the minFunc solver to use more function evaluations to find a maxima.

As expected, the MSE of all methods decreases as the training set size N is increased. However, two other interesting trends appear. First, we see that higher iBound methods have greater error when N is small. For example, the MSE of WMB with iBound 2 (WMB_2 in red) lies below the MSE of WMB_8 for $N < 500$. As N is increased, the MSE of WMB_8 drops below the MSE of WMB_2, which is the expected behavior. Second, we see that the GBP-based estimators have smaller error for large N than the WMB-based estimators. This suggests that the WMB upper bound on the log-partition function is loose and less accurate than the GBP-based log-partition function approximation.

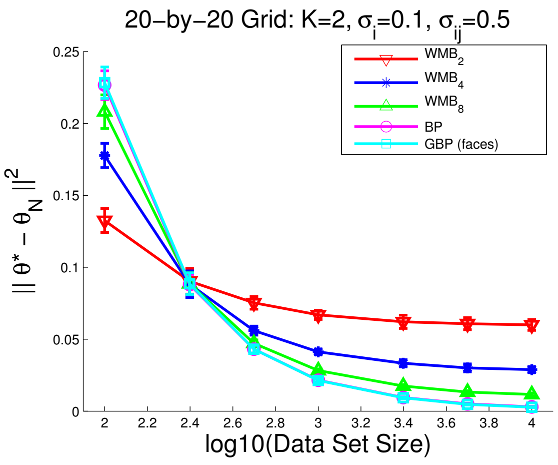
Synthetic Grids: Varying Grid Size

We also conducted experiments to see how the estimation error scales as the size of the underlying grid is increased. In particular, we considered $d \times d$ grids with $d \in \{10, 20, 30, 40\}$. For each setting of d, we generated 8 different models with parameters $\boldsymbol{\theta}^{\star}$ sampled using $\sigma_i = 0.1$ and $\sigma_{ij} = 0.5$.

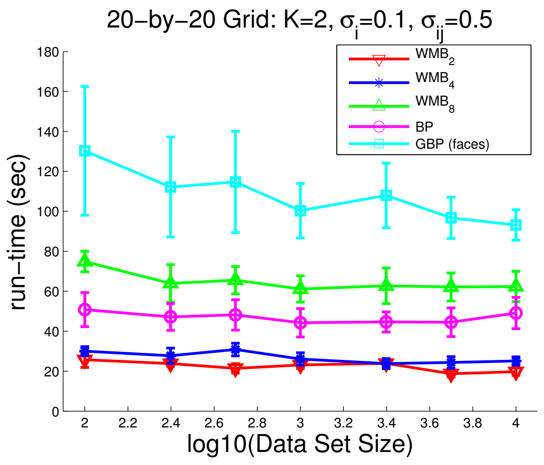
Figure 4.6 shows the MSE and run-times for the WMB and GBP-based estimators. Once again, the left column contains the MSEs and the right column the run-times. The top, middle and bottom row show the error in the estimates found using data sets of size $N = 100$, $N = 1000$ and $N = 10000$, respectively.

As we would hope, the MSE of all the estimators decreases as N is increased. In Figure 4.6a when $N = 100$ we once again see that WMB_2 estimator has smaller MSE than both the WMB_4 and WMB_8 estimators. While, in Figure 4.6e when $N = 10000$, the situation has reversed and WMB_8 has smaller MSE than both the WMB_4 and WMB_2 estimators.

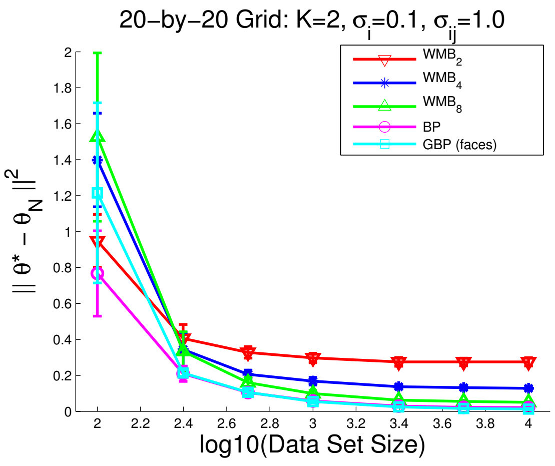
As expected, the run-times of the estimators grow as grid size is increased. However, the error of the different estimators remains fairly flat as d is increased from 10 to 40. As in the previous experiments, the MSE of the GBP-based estimators is smaller than the MSE of the WMB-based estimators for large N. However, the difference in run-times of the WMB and GBP-based estimators is quite large. As d is increased, the run-time of the WMB_2 estimator is much smaller than the BP-based estimator even though they are in the same iBound 2 complexity class. Similarly, the run-time of the WMB_4 estimator is much smaller than the GBP-based estimator even though both are iBound 4 methods.



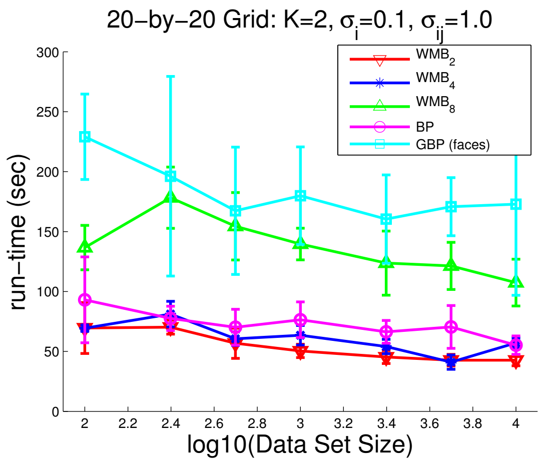
(a) MSEs for $\sigma_{ij} = 0.5$



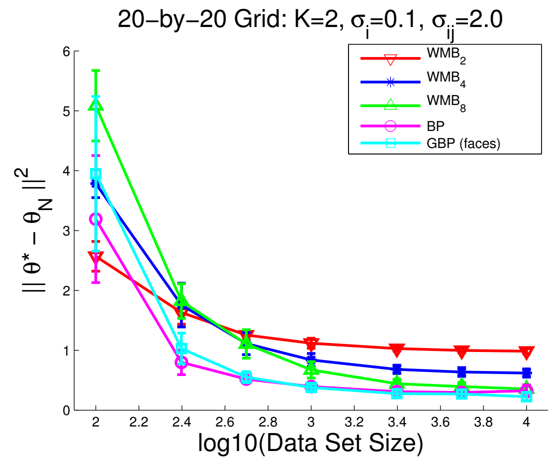
(b) Run-times for $\sigma_{ij} = 0.5$



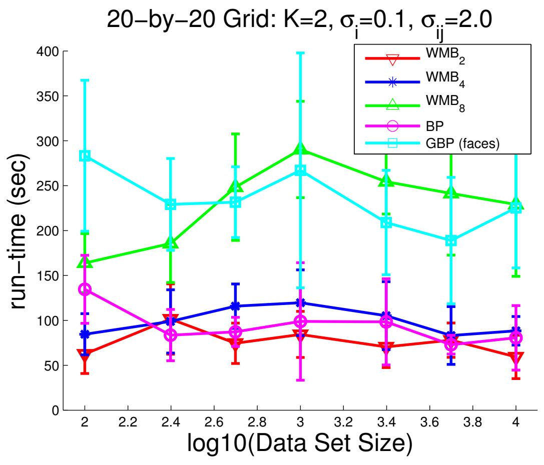
(c) MSEs for $\sigma_{ij} = 1.0$



(d) Run-times for $\sigma_{ij} = 1.0$



(e) MSEs for $\sigma_{ij} = 2.0$



(f) Run-times for $\sigma_{ij} = 2.0$

Figure 4.5: MSE and run-times of inference-based estimators on 20×20 grids.

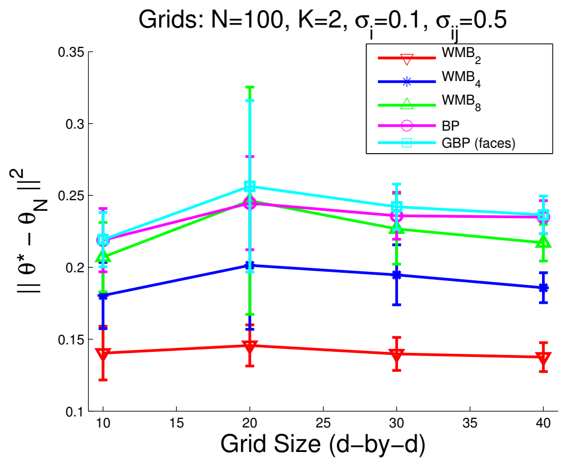
In order to determine if this disparity in run-time is due to differences in the curvature of the WMB and GBP-based surrogates or implementation differences, we recored the number of surrogate likelihood evaluations needed by the minFunc solver to find a maxima. Figure 4.7 plots the number of function evaluations for each grid size when $N = 10000$. Since, the GBP-based estimator requires fewer function evaluations (on average) than the WMB_4 based estimator, the run-time discrepancies appear to be due to implementation differences.

Synthetic Grids: Changing Variable Domain Size

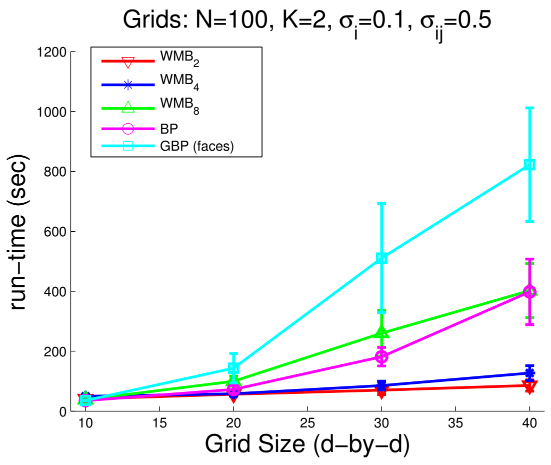
We also conducted experiments to see how the estimation error scales as the domain size of the variables in our model is changed. In particular, we considered a 10×10 grid of K-ary variables with $K \in \{2, 3, 4, 5\}$. For each setting of K, we generated 8 different models with parameters $\boldsymbol{\theta}^\star$ sampled using $\sigma_i = 0.1$ and $\sigma_{ij} = 0.5$.

Figure 4.8 shows the MSE and run-times for the WMB and GBP-based estimators. Once again, the left column contains the MSEs and the right column the run-times. The top, middle and bottom row show the error in the estimates found when training with data sets of size $N = 100$, $N = 1000$ and $N = 10000$, respectively.

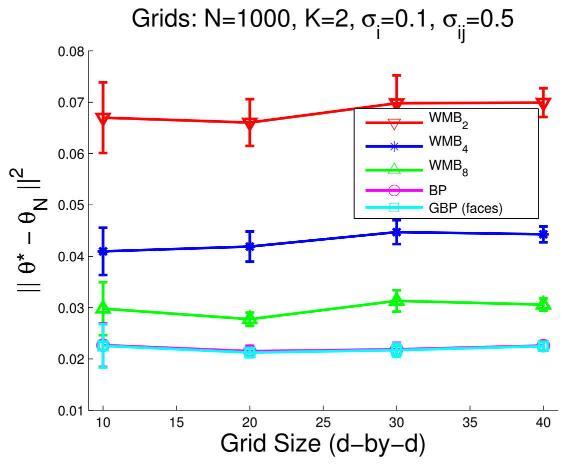
As K is increased, we see that the MSE of all the estimators grows. Interestingly, with a data set of size $N = 100$ in Figure 4.8a, the MSE of the WMB_2 estimator is below that of the WMB_4 estimator for all K. When $N = 1000$ in Figure 4.8c, we see that the MSE of the WMB_2 estimator is below that of the WMB_4 estimator for $K >= 4$. However, when $K < 4$, the WMB_4 estimator has smaller MSE. Since the number of parameters needing to be estimated grows as K is increased it is not that surprising that more data are needed to fit a model with larger K. Once again, when $N = 10000$ the GBP-based estimators have smaller MSE than the WMB-based estimators.



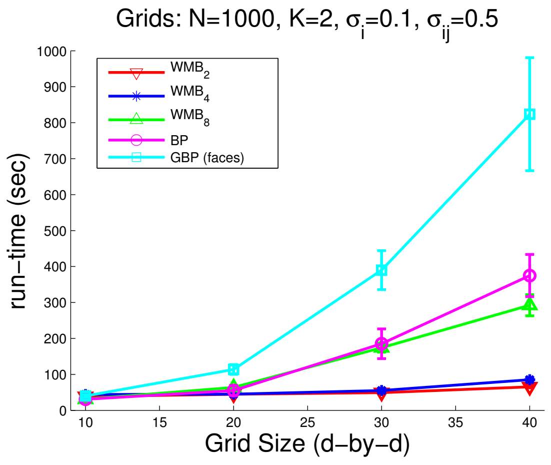
(a) MSEs for $N = 100$



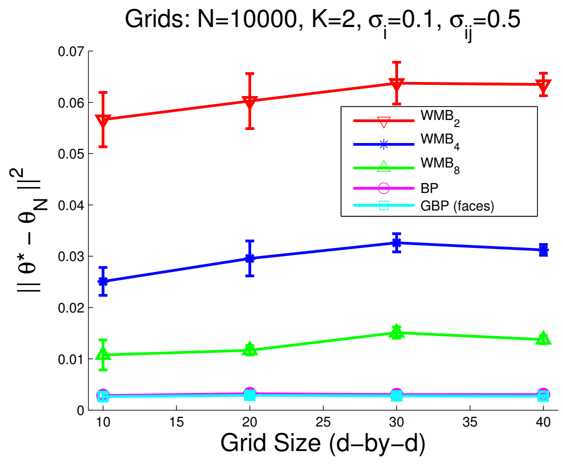
(b) Run-times for $N = 100$



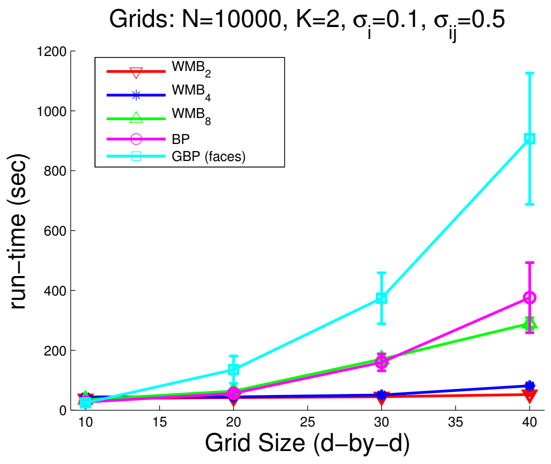
(c) MSEs for $N = 1000$



(d) Run-times for $N = 1000$



(e) MSEs for $N = 10000$



(f) Run-times for $N = 10000$

Figure 4.6: MSE and run-times of inference-based estimators on differently sized grids.

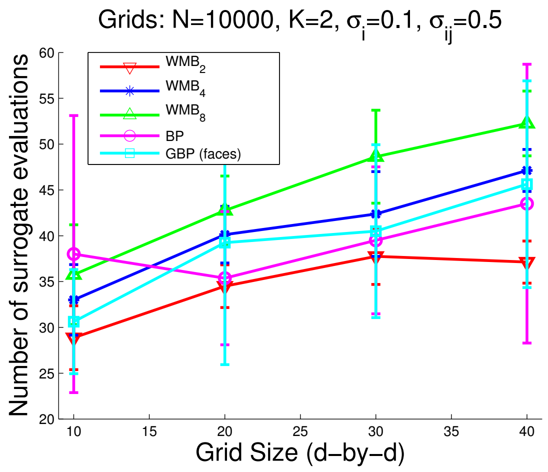


Figure 4.7: Number of Surrogate Likelihood evaluations with $N = 10000$.

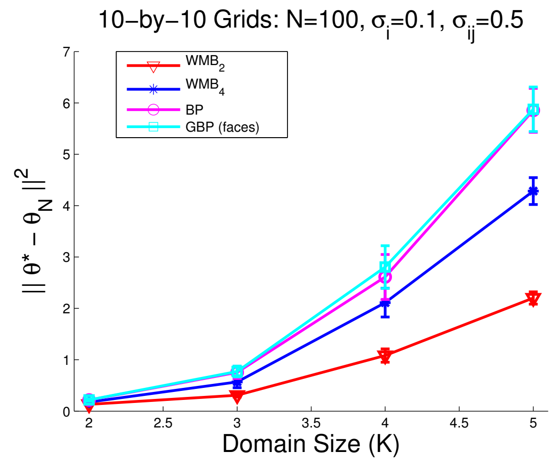
Synthetic Planar Graphs

The previous set of experiments have all considered models with a regular grid structure. This somewhat limits the utility of our empirical findings because the quality of approximate inference depends not only on the models parameters, but also on the underlying graph structure. We thus decided to conduct a set of experiments on some less regularly structured planar graphs. MRFs on planar graphs are a useful class of models to study as they are commonly used in computer vision tasks such as image segmentation.

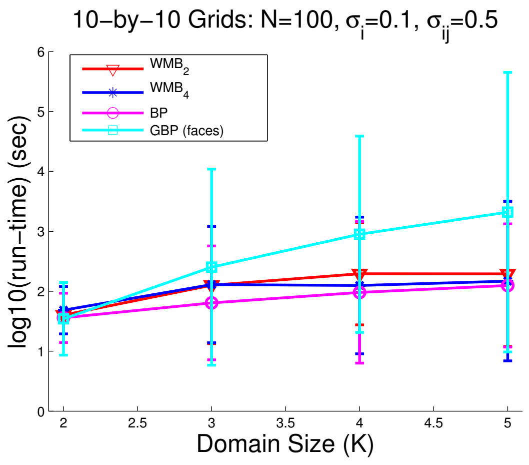
We formed the planar graph structure underlying each MRF by randomly placing $d \in \{50, 100, 150, 200, 250\}$ points in the unit square and computing a Delaunay triangulation on the set of d points. For each setting of d, we generated 25 different models with parameters $\boldsymbol{\theta}^\star$ sampled using $\sigma_i = 0.1$ and $\sigma_{ij} = 0.5$.

Figure 4.9 shows the MSE and run-times for the WMB and BP-based estimators. Once again, the left column contains the MSEs, the right column contains the run-times and the top, middle and bottom rows show the errors with training sets of size $N = 100$, $N = 1000$ and $N = 10000$, respectively.

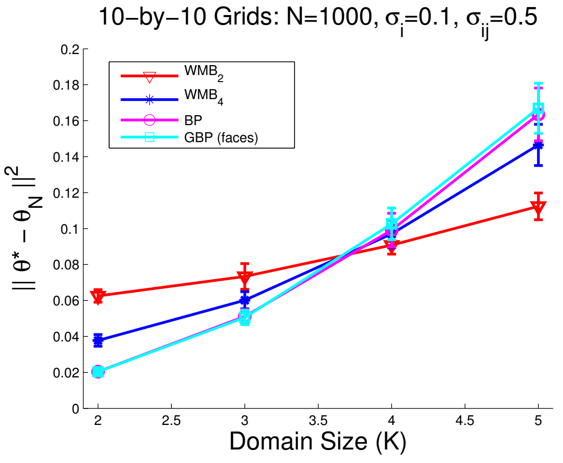
The trends for the planar graphs are quite similar to the grid instances. As N is increased



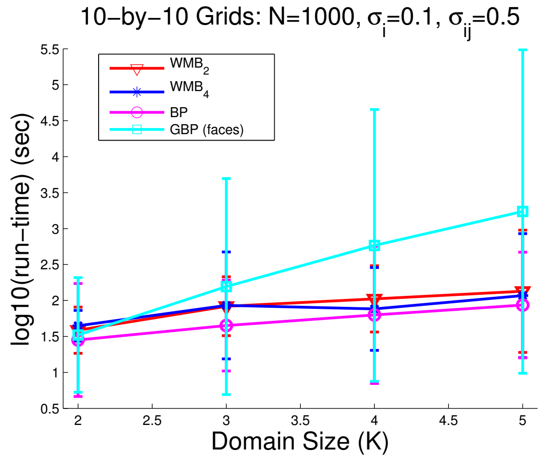
(a) MSEs for $N = 100$



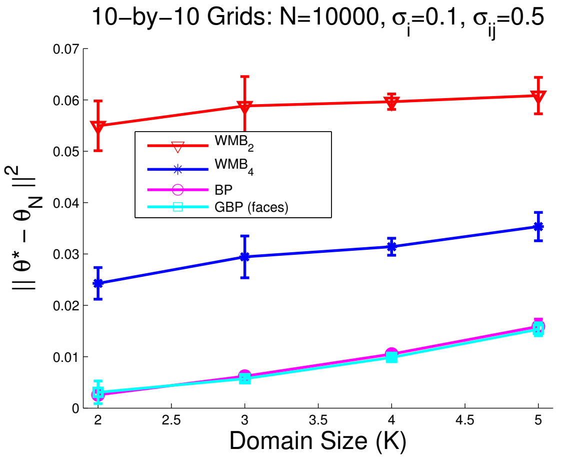
(b) Run-times for $N = 100$



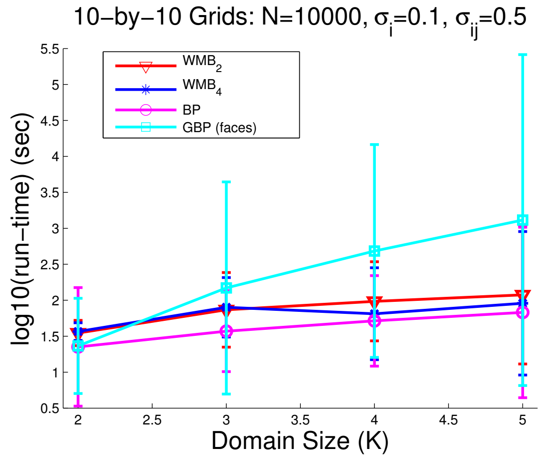
(c) MSEs for $N = 1000$



(d) Run-times for $N = 1000$



(e) MSEs for $N = 10000$



(f) Run-times for $N = 10000$

Figure 4.8: MSE and run-times of inference-based estimators for different K.

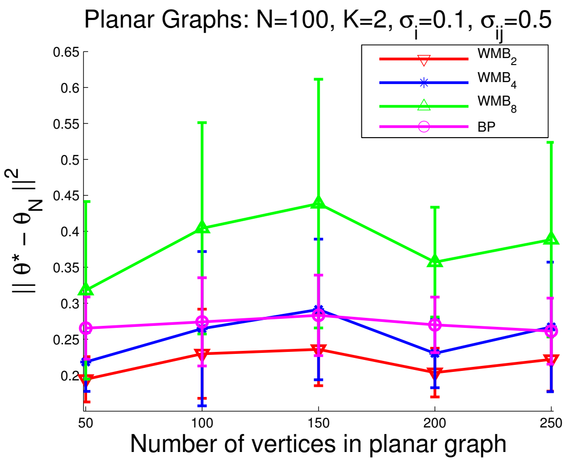
the MSE of all the estimators improves. Once again, the WMB_2 estimator has smallest MSE for $N = 100$ samples (Figure 4.9a), but largest error when $N = 10000$ (Figure 4.9e). The BP-based estimator once again has smaller error than the WMB-based estimators for large N.

One interesting new trend in these set of instances is the clear reduction in run-time of the WMB-based estimators as N is increased. Note that in Figure 4.9b, it takes the minFunc solver more than 200 seconds to find the WMB_8 point estimate, $\tilde{\boldsymbol{\theta}}_N^{\text{iBound}=8}$, while in Figure 4.9f it takes fewer than 100 seconds to find the maxima. This is a clear instance where the larger data set size increases the curvature or our surrogate likelihood function and makes it easier for the minFunc solver to find the optimum. Figure 4.10 is a zoomed-in view of the run-times of the different estimators for the models with $d = \{50, 100\}$ variables. Note that the run-time of the WMB_8 estimator (in green) is in fact lower than the run-time of the WMB_2 estimator (in red) in this chart. This is remarkable given that the run-time complexity of the WMB_8 is 4 times that of WMB_4 and demonstrates the role that curvature plays when finding a point estimate.

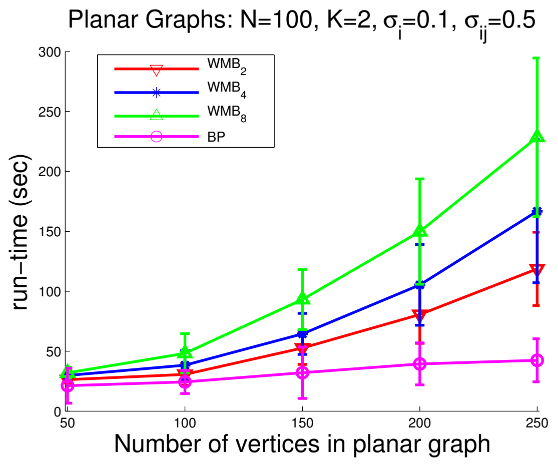
Bias-Variance Comparison

The results in the previous sections suggested that lower iBound methods are (on average) more accurate than higher iBound methods in the small data setting. However, as N is increased our intuition that higher iBound methods have asymptotically smaller bias was also confirmed. In other words, it appears that lower iBound methods trade larger bias for reduced variance, while higher iBound methods trade smaller bias for increased variance.

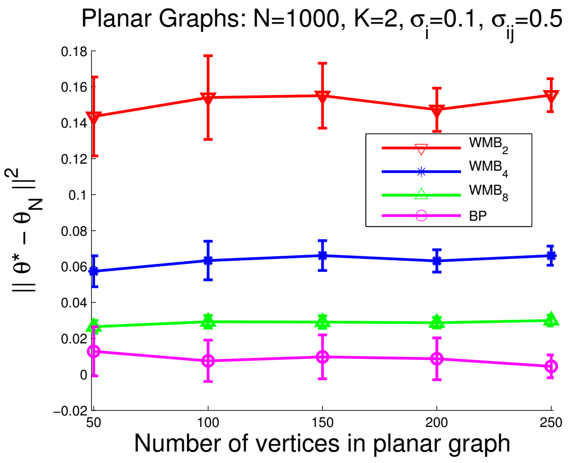
We conducted a simple experiment in order to study the bias and variance characteristics of different approximate inference-based estimators. To estimate the bias and variance of each estimator we first fix the parameters $\boldsymbol{\theta}^{\star}$ of a 20×20 binary grid of variables. We then sample



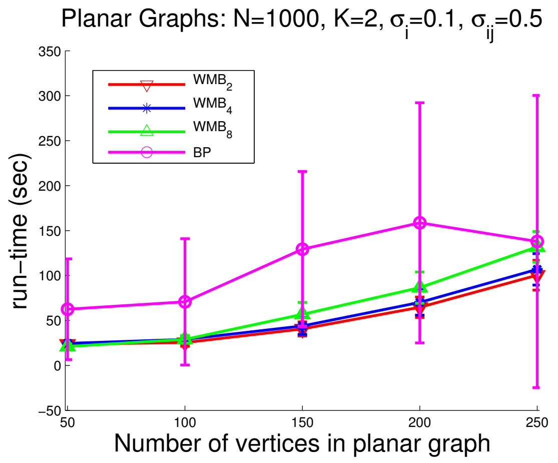
(a) MSEs for $N = 100$



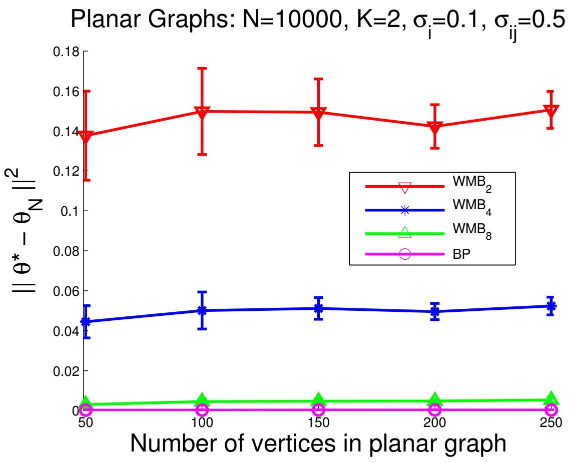
(b) Run-times for $N = 100$



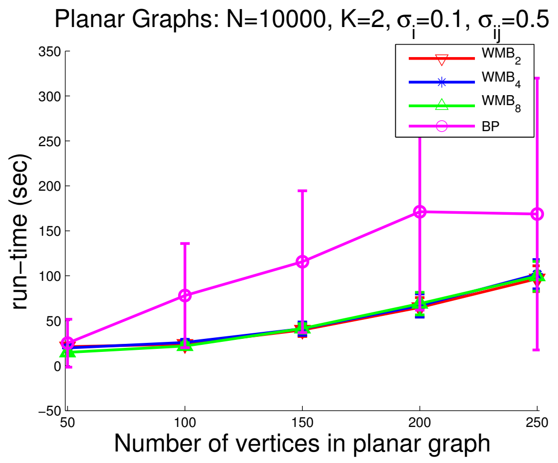
(c) MSEs for $N = 1000$



(d) Run-times for $N = 1000$



(e) MSEs for $N = 10000$



(f) Run-times for $N = 10000$

Figure 4.9: MSE and run-times of inference-based estimators on planar graphs.

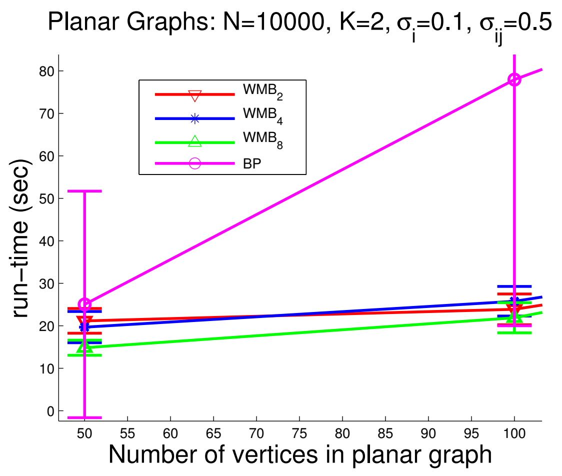


Figure 4.10: Zoom of run-times of estimators on planar graphs with $N = 10000$.

$L = 25$ different data sets from $p(\boldsymbol{y}; \boldsymbol{\theta}^\star)$ for each $N \in \{100, 250, 500, 1000, 2500, 5000, 10000\}$, giving a total of 175 different data sets. We then estimate the parameters $\tilde{\boldsymbol{\theta}}_N$ using each approximate inference-based estimator on each of the 175 data sets. For each data set size, N, we estimate the bias of each estimator as

$$Bias(\tilde{\boldsymbol{\theta}}_N) = \sum_{d=1}^{D} Bias(\tilde{\theta}_{N,d}) \quad \text{with} \quad Bias(\tilde{\theta}_{N,d}) \approx mean(\tilde{\theta}_{N,d}) - \theta^\star_d \tag{4.35}$$

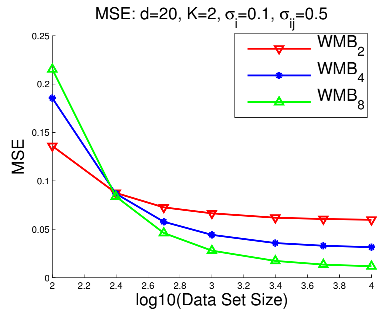
where $\tilde{\theta}_{N,d}$ is the d^{th} element of the parameter vector $\tilde{\boldsymbol{\theta}}_N \in \mathbb{R}^D$ and

$$Mean(\tilde{\theta}_{N,d}) = \frac{1}{L}\sum_{l=1}^{L} \tilde{\theta}^{(l)}_{N,d} \tag{4.36}$$

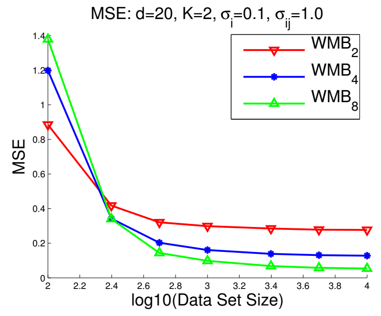
with $\tilde{\theta}^{(l)}_{N,d}$ the setting of $\tilde{\theta}_{N,d}$ estimated using data set l. The variance of each estimator is estimated as

$$Var(\tilde{\boldsymbol{\theta}}_N) = \sum_{d=1}^{D} Var(\tilde{\theta}_{N,d}) \quad \text{with} \quad Var(\tilde{\theta}_{N,d}) \approx \frac{1}{L}\sum_{l=1}^{L}\left(\tilde{\theta}^{(l)}_{N,d} - Mean(\tilde{\theta}_{N,d})\right)^2. \tag{4.37}$$

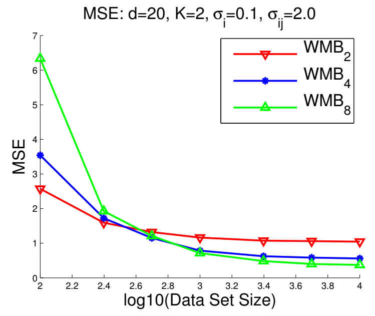
Figure 4.11 exposes the bias-variance trade-off for WMB-based estimators with iBounds of 2, 4 and 8. Each column corresponds to estimating the parameters of a different MRF: the



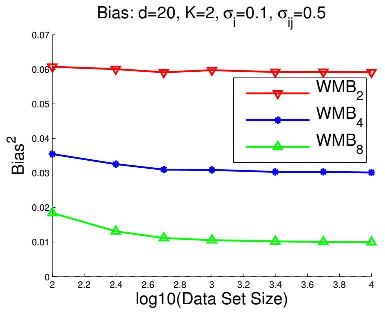
(a) MSE in grid with $\sigma_{ij} = 0.5$



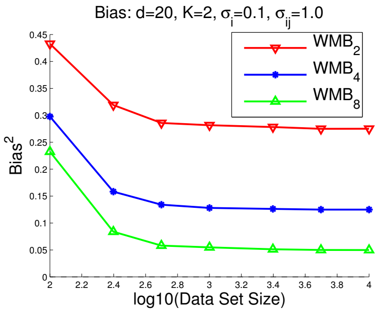
(b) MSE in grid with $\sigma_{ij} = 1.0$



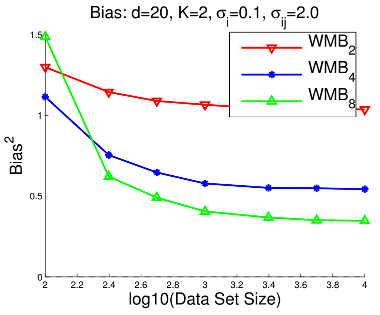
(c) MSE in grid with $\sigma_{ij} = 2.0$



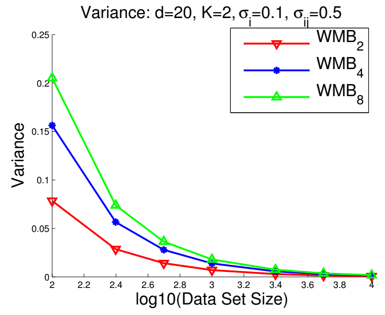
(d) Bias in grid with $\sigma_{ij} = 0.5$



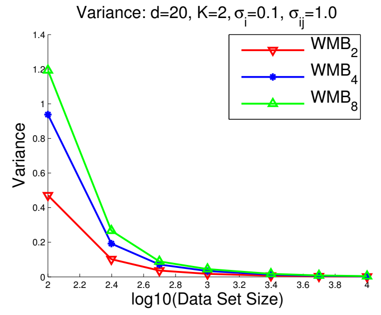
(e) Bias in grid with $\sigma_{ij} = 1.0$



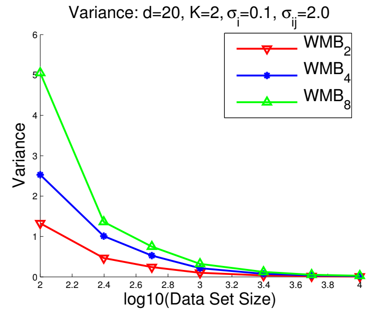
(f) Bias in grid with $\sigma_{ij} = 2.0$



(g) Variance in grid with $\sigma_{ij} = 0.5$



(h) Variance in grid with $\sigma_{ij} = 1.0$



(i) Variance in grid with $\sigma_{ij} = 2.0$

Figure 4.11: MSE, Bias and Variance of WMB-based estimators.

left column is a model in which the unary and pairwise potentials were drawn with $\sigma_i = 0.1$ and $\sigma_{ij} = 0.5$, respectively; in the middle column, the potentials were drawn with $\sigma_i = 0.1$ and $\sigma_{ij} = 1.0$; and, in the right column with $\sigma_i = 0.1$ and $\sigma_{ij} = 2.0$. The top, middle and bottom rows show the MSE, bias^2 and variance, respectively.

The top row shows the MSE as a function of data set size, N, for the different WMB-based estimators. Notice that for small N, the WMB-based estimator with an iBound of 2 has smaller MSE than the WMB-based estimators with higher iBounds. As N is increased, however, the MSE of the iBound 2 estimator becomes larger than the other WMB-based estimators.

The second and third rows of Figure 4.11 provide an explanation for this behavior. The middle row shows the bias and the bottom row the variance of the estimators as a function of data set size. For all three settings of σ_{ij} and for almost all N, the bias of the iBound 8 estimator is less than the bias of the iBound 2 estimator. However, this reduced bias comes at the expense of increased variance. In particular, for small N the variance of the iBound 8 estimator is typically much larger than the variance of the iBound 2 estimator. As a result, when the difference between the bias of the WMB_2 and WMB_8 estimators is less than the difference in their variance, the MSE of the WMB_2 is smaller than the MSE of the WMB_8 estimator.

This finding has important practical consequences. In practice, we only observe one data set $\{\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(N)}\}$. If the data set size, N, is small relative to the dimensionality, D, of the parameter vector being estimated, then the reduced variance of a lower iBound method may be able to compensate for higher variance of the data, without much additional regularization. Figure 4.12 plots the 2-norm of the parameter vectors found by the different WMB-based estimators as a function of data set size. Notice that the WMB_2 curve (in red) is always below the curves of the WMB_4 and WMB_8 estimators (in blue and green, respectively). This indicates that the WMB_2 estimator effectively regularizes, or controls the complexity of our learned model, more strongly than the higher iBound methods.

Figure 4.13 further illustrates the difference in the regularization inherent to low and high iBound methods. The contours trace out the difference between the true likelihood function, $\ell(\boldsymbol{\theta})$, and the surrogate likelihood function, $\tilde{\ell}(\boldsymbol{\theta})$, of the WMB estimator with iBounds of 2 and 8. The contours are shown for a two parameter model, $\boldsymbol{\theta} = (\theta_i, \theta_{ij})$, on a 10×10 grid

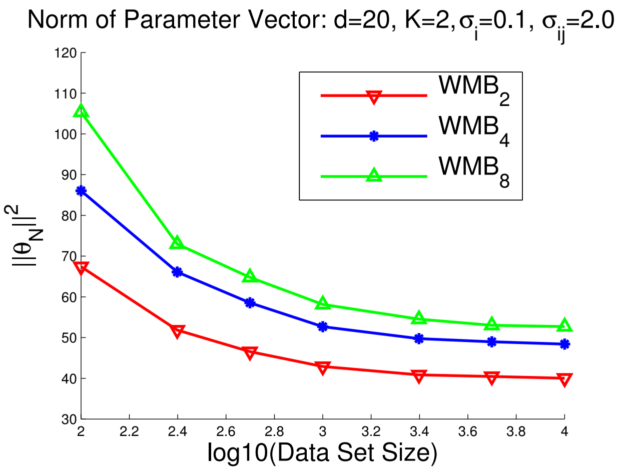
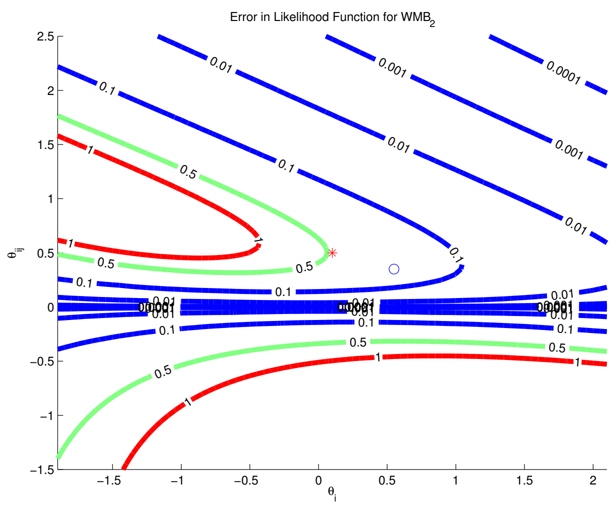


Figure 4.12: 2-Norm of parameter vectors for WMB-based estimators.

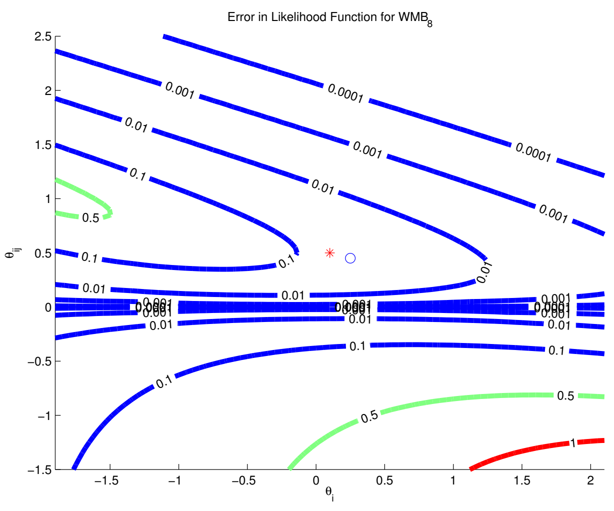
of binary variables ($y_i \in \{0, 1\}$) with the following parameterization:

$$p(\boldsymbol{y}; \theta_i, \theta_{ij}) = \exp\left(\sum_{i \in V} \theta_i y_i + \sum_{(i,j) \in E} \theta_{ij} y_i y_j - \log Z(\theta_i, \theta_{ij})\right). \tag{4.38}$$

Notice that both the iBound 2 method in Figure 4.13a and the iBound 8 method in Figure 4.13b have areas along the horizontal axis where the difference between the likelihood functions, $\ell(\boldsymbol{\theta}) - \tilde{\ell}(\boldsymbol{\theta})$, is small. This occurs because when $\theta_{ij} = 0$ there is no inter-variable interaction and both methods become exact. The error of both WMB-based estimators increases as we move away from the horizontal axis, but the error increases much more rapidly for the iBound 2 method than the iBound 8 method. For example, the location of $\boldsymbol{\theta}^\star$ indicated by the red asterisk, lies on the 0.5 error contour for the iBound 2 method, but is between the 0.1 and 0.01 contours for the iBound 8 method. The blue circles indicate the location of the point estimates $\boldsymbol{\theta}^{\text{WMB}} = \arg\max_{\boldsymbol{\theta}} \tilde{\ell}(\boldsymbol{\theta})$. Both WMB-based estimators are biased towards parameter configurations where they are more accurate. However, the lower iBound 2 estimator is more heavily biased than the iBound 8 estimator because the region in which it is accurate is shifted farther to the right. The stronger bias means there is more resistance to move away from these favored configurations, which in turn provides a reduction in variance.



(a) $\ell(\boldsymbol{\theta}) - \tilde{\ell}(\boldsymbol{\theta})$ for WMB$_2$ estimator.



(b) $\ell(\boldsymbol{\theta}) - \tilde{\ell}(\boldsymbol{\theta})$ for WMB$_8$ estimator.

Figure 4.13: Error in WMB-based surrogates. Red asterisk is $\boldsymbol{\theta}^\star = (\theta_i^\star, \theta_{ij}^\star)$; blue circle is $\boldsymbol{\theta}^{\text{WMB}}$.

4.4.3 Model, Estimation and Optimization Error Comparison

We also consider an experiment in which $\underline{\Theta} \subset \Theta$. In particular, we generate our data from a pairwise grid with 8-neighbor connectivity, but learn a model with only 4-neighbor connectivity (see Figure 4.14). So for a $d \times d$ grid, we have $\Theta = \mathbb{R}^{2(d(d-1)+(d-1)^2)}$ and $\underline{\Theta} = \mathbb{R}^{2d(d-1)}$. The pairwise potentials on the *new* diagonal edges are drawn from $\mathcal{N}(0, \sigma_{ik}^2)$ and we increase σ_{ik} from 0 to 1 to increase the level of model error. The unary potentials and *standard* edge potentials are drawn from $\sigma_i = 0.1$ and $\sigma_{ij} = 0.3$, respectively.

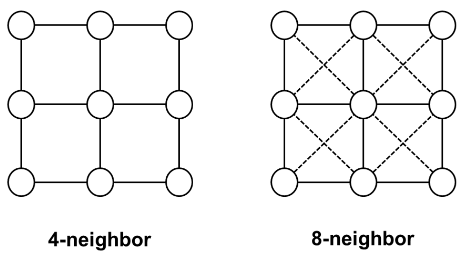


Figure 4.14: Illustration of 4- and 8-neighbor connecivity.

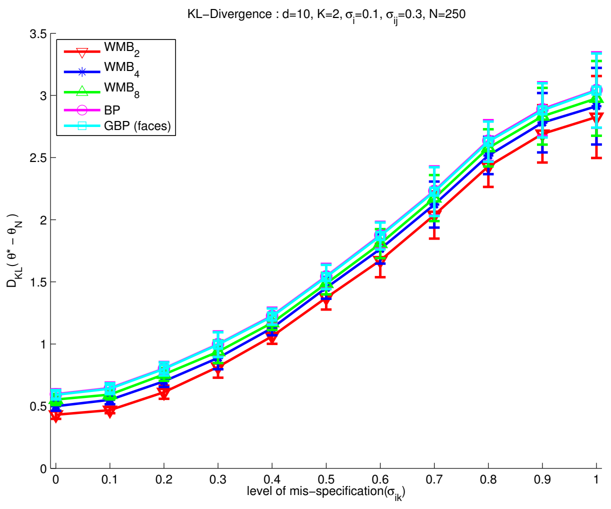
As usual, we generate 8 different models for each setting of the diagonal interactions $\sigma_{ik} \in \{0, 0.1, 0.2, ..., 1.0\}$ and then sample data sets of various size for each model. Since $\underline{\Theta} \subset \Theta$

we compute the estimation error in these experiments via the KL-Divergence,

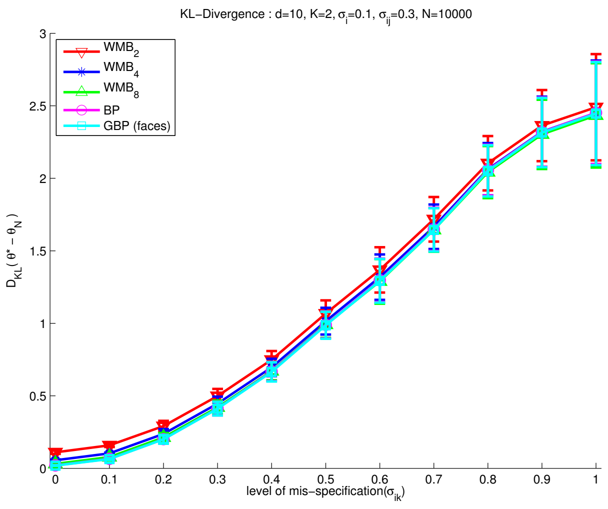
$$D_{KL}(p(\boldsymbol{y};\boldsymbol{\theta}^{\star})||p(\boldsymbol{y};\underline{\tilde{\boldsymbol{\theta}}}_N)) = \mathbb{E}_{\boldsymbol{\theta}^{\star}}\left[\log p(\boldsymbol{y};\boldsymbol{\theta}^{\star})\right] - \mathbb{E}_{\boldsymbol{\theta}^{\star}}\left[\log p(\boldsymbol{y};\underline{\tilde{\boldsymbol{\theta}}}_N)\right],$$

which you may recall is the expression inside the outer expectation over data sets in the excess error decomposition of (4.32).

Figures 4.15a and 4.15b plot the estimation accuracy of the different inference-based estimators as a function of misspecification level, σ_{ik}, for data sets of size $N = 250$ and $N = 10000$, respectively. Notice that WMB_2 estimator has smaller estimation error than the other inference-based estimators when $N = 250$. However, for $N = 10000$ the situation flips and it has the largest error.



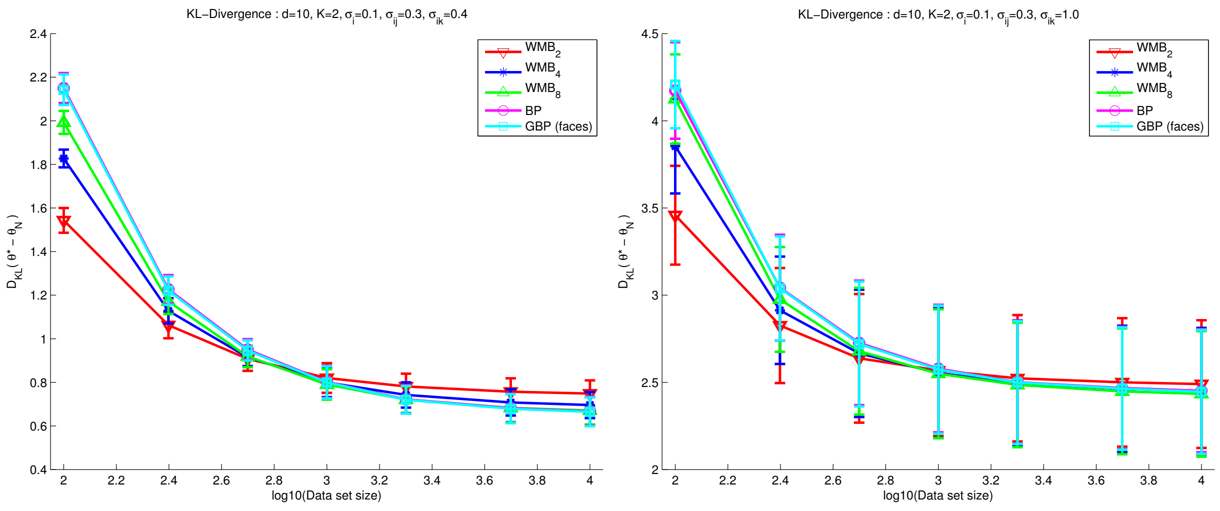
(a) Error vs. noiselevel (σ_{ik}) with $N = 250$



(b) Error vs. noiselevel (σ_{ik}) with $N = 10000$

Figure 4.15: Estimation error of inference-based estimators as function of model error.

Figures 4.16a and 4.16b plot the estimation error as a function of data set size for noise levels of $\sigma_{ik} = 0.4$ and $\sigma_{ik} = 1.0$, respectively. Note that the estimation error of the estimators decreases as N is increased for both noiselevels. This occurs because with an increased number of samples we are able to reduce estimation error despite the presence of both model and optimization error. Notice also that the error bars in Figure 4.16b are larger than the error bars in Figure 4.16a. This is expected because the increased level of model error increases the overall error in our parameter estimates.



(a) Error vs. data size (N) for $\sigma_{ik} = 0.4$

(b) Error vs. data size (N) for $\sigma_{ik} = 1.0$

Figure 4.16: Estimation error of inference-based estimators as function of data set size.

4.5 Empirical Study of Inference-Based Prediction

The joint estimation and prediction problem, as illustrated in Figure 4.1, involves estimating the parameters of a model and then using the fitted model to make predictions on new data. The experiments in Section 4.4 focused on the estimation problem and, in particular, examined the effect of using different approximate inference methods on the accuracy of parameter estimation (as measured by MSE). In this section, we focus on the prediction task and study the effect of using different inference methods on prediction accuracy. In other words, rather than focusing on how well we can learn the parameters of a model, we focus on how accurately we can predict under the learned model.

We follow the classical approach to joint estimation and prediction described in Section 4.1, where we first estimate the parameters of our model, $\boldsymbol{\theta}$, on a training set and then appeal to Bayesian decision theory to construct a prediction rule $\boldsymbol{y}^{\text{tst}} = f(\boldsymbol{x}^{\text{tst}}, \boldsymbol{\theta})$ to make predictions on test point $\boldsymbol{x}^{\text{tst}}$ for our chosen loss function [74]. We focus in this section on the Hamming loss and Mean Squared Error loss. The Bayes optimal predictor for the Hamming-loss is: $y_i^{\text{tst}} = \arg\max_{y_i \in \mathcal{Y}} p(y_i|\boldsymbol{x};\boldsymbol{\theta})$, for each variable i in our model. The Bayes optimal predictor for the Mean Squared Error loss is: $y_i^{\text{tst}} = E_{\boldsymbol{\theta}}\left[y_i\right] = \sum_{y_i \in \mathcal{Y}} y_i p(y_i|\boldsymbol{x};\boldsymbol{\theta})$.

Inference is thus needed in both the estimation phase and the prediction phase. We once again focus on approximate inference methods that utilize an iBound parameter to trade computation for accuracy – with a smaller iBound requiring less computation, but typically providing a less accurate approximation. Since increasing the iBound increases computation, in this section we consider whether higher iBound methods yield more accurate predictions. As in the previous section, the answer depends on the problem under consideration: in some cases the higher iBound is beneficial and in other cases it is actually detrimental. And as in the previous section, we hold the regularization parameter fixed during training in order to study how the bias and variance properties of the different inference-based estimators affect prediction accuracy.

This section is split into three different experiments. In the first experiment, we estimate the parameters of a joint MRF model, $p(\boldsymbol{y},\boldsymbol{x};\boldsymbol{\theta})$, and then fix the elements of $\boldsymbol{x}$ to an observed value $\boldsymbol{x}^{\text{tst}}$ to form the conditional distribution $p(\boldsymbol{y}|\boldsymbol{x}^{\text{tst}};\boldsymbol{\theta})$ used for prediction. Importantly, we choose a statistically identifiable parameterization of the joint model, $p(\boldsymbol{y},\boldsymbol{x};\boldsymbol{\theta})$, so that we can compute both the estimation error and the prediction error. In the second and third experiments, we use a CRF model to directly learn a conditional distribution $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$. Since the underlying parameters of the conditional model are not known, we evaluate the prediction accuracy of the learned model on a held out test set.

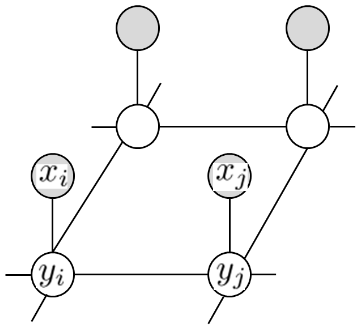


Figure 4.17: Markov Random Field (MRF) model for image de-noising.

MRF Image De-noising

We conducted an experiment involving the real world task of image de-noising, where we are given a noisy image, $\boldsymbol{x}$, and our goal is to reconstruct the original, noise free image $\boldsymbol{y}$. We model the joint probability of noisy and noise-free images, $p(\boldsymbol{y}, \boldsymbol{x})$, as an MRF with the undirected graph structure in Figure 4.17. Each pixel i is modeled with a binary variable $y_i \in \{0, 1\}$ and the noisy pixels are also assumed to be binary $x_i \in \{0, 1\}$. The joint distribution over the collection of all pixels $\boldsymbol{y}$ and $\boldsymbol{x}$ is given by

$$p(\boldsymbol{y}, \boldsymbol{x}; \boldsymbol{\theta}) \propto \exp\left(\sum_i \theta_i y_i + \sum_{ij} \theta_{ij} y_i y_j + \sum_i \theta_{ii} y_i x_i\right) \tag{4.39}$$

where θ_i is a bias parameter for pixel i, θ_{ij} is a parameter governing how likely neighboring pixels i and j are to be 'on' at the same time and θ_{ii} is a parameter governing how likely both the noisy and original pixel at site i are to be 'on'. This MRF is in a minimal parameterization and is therefore statistically identifiable.

We generated a training and test set using the USPS handwritten digits data set[7], which contains 1100, 16×16 pixel grayscale images of each digit '0' to '9'. We construct a data set for each digit by first binarizing the original grayscale images and then constructing a noisy version of each binarized image by randomly flipping pixels with probability $p_{\text{noise}} = 0.3$. We

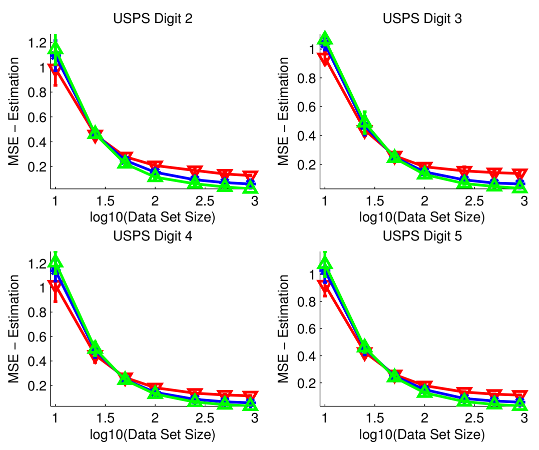
[7]http://www.cs.nyu.edu/ roweis/data.html

then randomly split each digit's data set $\{(\boldsymbol{y}^{(1)}, \boldsymbol{x}^{(1)}), ..., (\boldsymbol{y}^{(1100)}, \boldsymbol{x}^{(1100)})\}$ into a test set of $N_{\text{tst}} = 200$ images and a training set of $N_{\text{trn}} = 900$ images. The entire process is repeated 10 times and the reported errors are averaged across these 10 folds.

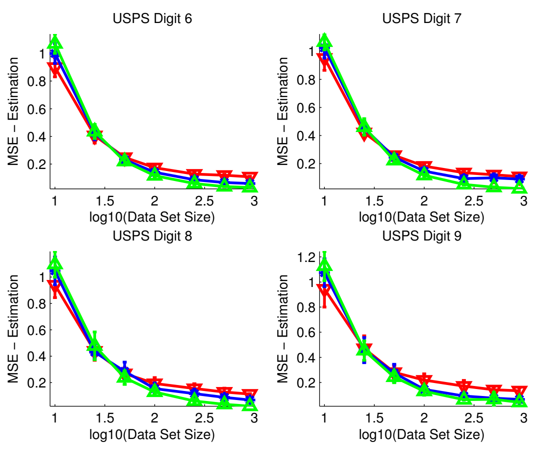
We estimate the parameters $\tilde{\boldsymbol{\theta}}_N$ of the de-noising MRF using different WMB-based estimators on subsets of the training data of sizes $N = \{25, 50, 100, 250, 500, 900\}$. We also compute the MLE, $\boldsymbol{\theta}_{N=900}$, of the de-noising MRF using exact inference. The regularization parameter was fixed to $\lambda = 1e^{-2}$ for the different WMB-based estimators. The regularization parameter when computing the MLE, $\boldsymbol{\theta}_{N=900}$, was determined via a grid search on the values $\{1e^{-1}, 1e^{-2}, 1e^{-3}, 1e^{-4}, 1e^{-5}\}$ and was set to $\lambda = 1e^{-4}$.

Figure 4.18 shows the estimation error of the different estimators on digits $2-9$, where the red, blue and green lines once again indicate the WMB_2, WMB_4 and WMB_8 estimators. Since we do not know the true parameters, $\boldsymbol{\theta}^{\star}$, in this experiment, we measure the error between the MLE and the WMB-based estimate: $||\boldsymbol{\theta}_{N=900} - \tilde{\boldsymbol{\theta}}_N||^2$. Notice that much like the experiments in Section 4.4, we see that for small N, the reduced variance of the WMB_2 estimator lead to smaller estimation error than the higher iBound methods. However, unlike the prior experiments these parameters are being estimated from real image data. This shows that the bias-variance trade-off of the WMB-based estimators is a real phenomena rather than an artifact of the experimental setup in Section 4.4.

We then used the model learned using each of the WMB-based estimators to make predictions on the test set of images. In particular, we fix the elements of $\boldsymbol{x}$ to be the observed pixel values in each test image $\boldsymbol{x}^{\text{tst}}$. This implicitly defines the distribution $p(\boldsymbol{y}|\boldsymbol{x}^{\text{tst}}; \tilde{\boldsymbol{\theta}}_N)$ which we can then use to make predictions. We take the loss in these experiments to be the Mean Squared Error (MSE) and make predictions in our binary de-noising model as: $y_i^{\text{tst}} = E_{\tilde{\boldsymbol{\theta}}_N}[y_i|\boldsymbol{x}^{\text{tst}}] = p(y_i = 1|x_i^{\text{tst}}; \tilde{\boldsymbol{\theta}}_N)$. The inference method used to estimate $\tilde{\boldsymbol{\theta}}_N$ was also used to compute the marginals needed for prediction – i.e., if WMB_2 was used in learning, then WMB_2 was also used in prediction.



(a) Estimation error for digits $2-5$.



(b) Estimation error for digits $6-9$.

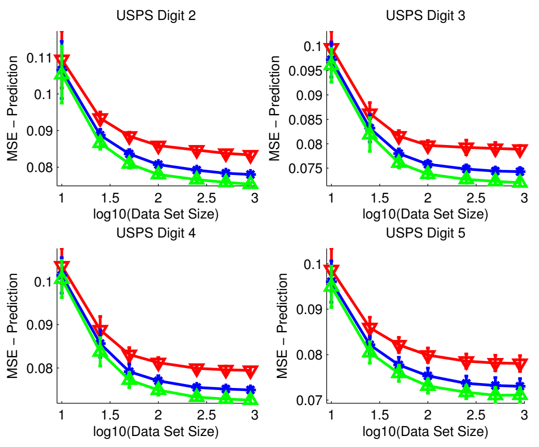
Figure 4.18: Estimation error of WMB-based estimators in MRF de-noising task with $p_{\text{noise}} = 0.3$.

Figure 4.19 shows the prediction error for digits $2-9$ as a function of the size, N, of the data set used to estimate $\tilde{\boldsymbol{\theta}}_N$. The prediction error for all of the methods decreases as N is increased as we would hope. However, we see that the prediction error of higher iBound methods is consistently less than the error of lower iBound methods. This is quite interesting given that in Figure 4.18 the higher iBound methods had larger estimation error for small N. In other words, it seems that the error introduced by having the wrong parameters is far less important for prediction accuracy than the error introduced by computing the wrong (or approximate) marginals on the model.

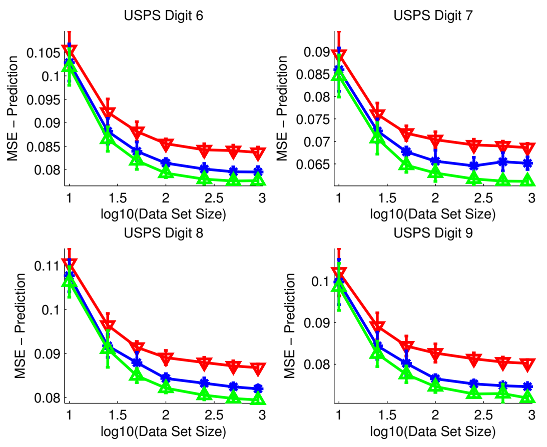
CRF Image De-noising

We conducted another set of image de-noising experiments on the USPS digits dataset. Since we are ultimately interested in recovering an image, $\boldsymbol{y}$, given a noisy observation, $\boldsymbol{x}$, in these experiments we directly model the conditional distribution, $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$, using a CRF of the following form:

$$p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) = \exp\left(\sum_i \sum_u \theta_u f_u(y_i, \boldsymbol{x}) + \sum_{ij} \sum_p f_p(y_i, y_j, \boldsymbol{x}) - \log Z(\boldsymbol{\theta}, \boldsymbol{x})\right), \tag{4.40}$$



(a) Prediction error for digits $2-5$.



(b) Prediction error for digits $6-9$.

Figure 4.19: Prediction error of WMB-based estimators in MRF de-noising task with $p_{\text{noise}} = 0.3$.

where $\boldsymbol{f}_u = \{f_u(\cdot)\}$ is a collection of unary features and $\boldsymbol{f}_p = \{f_p(\cdot)\}$ is a collection of pairwise features and $\log Z(\boldsymbol{\theta}, \boldsymbol{x})$ is a data-dependent normalization term. $\boldsymbol{\theta}_u = \{\theta_u\}$ and $\boldsymbol{\theta}_p = \{\theta_p\}$ are the parameters for each of these features and the entire CRF model is parameterized by $\boldsymbol{\theta} = (\boldsymbol{\theta}_u, \boldsymbol{\theta}_p)$.

The set of univariate and pairwise features used in de-noising are:

$$\boldsymbol{f}_u(y_i) = \begin{pmatrix} f_1(y_i, \boldsymbol{x}) = 1 \\ f_2(y_i, \boldsymbol{x}) = x_i \\ f_3(y_i, \boldsymbol{x}) = \text{loc}(i,1) \\ f_4(y_i, \boldsymbol{x}) = \text{loc}(i,2) \\ \vdots \\ f_{18}(y_i, \boldsymbol{x}) = \text{loc}(i,16) \end{pmatrix}, \quad \boldsymbol{f}_p(y_i, y_j) = \begin{pmatrix} f_1(y_i, y_j, \boldsymbol{x}) = 1 \,\&\, \text{hz}(i,j) \\ f_2(y_i, y_j, \boldsymbol{x}) = |x_i - x_j| \,\&\, \text{hz}(i,j) \\ f_3(y_i, y_j, \boldsymbol{x}) = 1 \,\&\, \text{vt}(i,j) \\ f_4(y_i, y_j, \boldsymbol{x}) = |x_i - x_j| \,\&\, \text{vt}(i,j) \end{pmatrix},$$

where $\text{loc}(i,j) = \begin{cases} 1 & \text{if pixel } i \text{ in tile } j, \\ 0 & \text{otherwise} \end{cases}$ is a function that indicates if pixel i is in a specific region j of the image. In particular, we partition the 16×16 pixel image into a set of 16, non-overlapping 4×4 tiles where, for example, the first tile corresponds to the 4×4 pixel region in the top left corner of the image. The functions $\text{hz}(i,j)$ and $\text{vt}(i,j)$ indicate if the edge between adjacent pixels i and j is oriented horizontally or vertically, respectively.

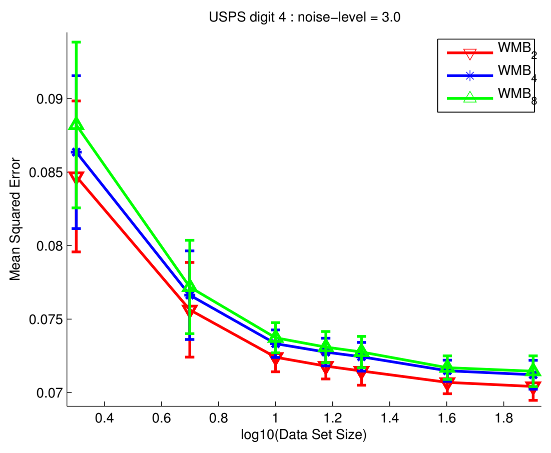
We use one copy of $\boldsymbol{f}_u(y_i, \boldsymbol{x})$ for each setting of y_i and one copy of $\boldsymbol{f}_p(y_i, y_j, \boldsymbol{x})$ for each joint configuration of (y_i, y_j), so that $\boldsymbol{f}_u = (\boldsymbol{f}_u(0), \boldsymbol{f}_u(1))$ and $\boldsymbol{f}_p = (\boldsymbol{f}_p(0,0), ..., \boldsymbol{f}_p(1,1))$. This gives a total of 36 unary features and 16 pairwise features so that $\boldsymbol{\theta}_u \in \mathbb{R}^{36}$ and $\boldsymbol{\theta}_p \in \mathbb{R}^{16}$.

As in the MRF de-noising experiments, we took the set of 1100, 16×16 pixel grayscale images of each digit, binarized the original grayscale images and then constructed a noisy version of each binarized image. In these experiments the noisy images are generated as $x_i = y_i(1-t_i^{nl})+(1-y_i)t_i^{nl}$, where $y_i \in \{0,1\}$ is the true binary label of pixel i, $t_i \sim Unif[0,1]$ is random, and $nl \in (0, \infty]$ is a parameter controlling the noise-level [24]. Here, smaller values of nl correspond to more noise.

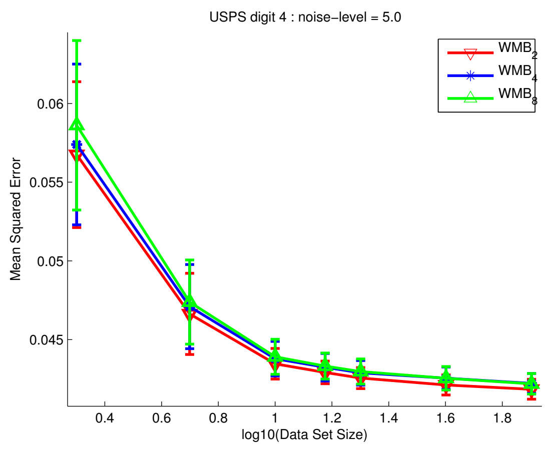
We randomly split the 1100 data points into 900 training points and 200 test points and estimate the parameters, $\tilde{\boldsymbol{\theta}}_N$, of the de-noising CRF model using different WMB-based estimators on data sets of size $N = \{2, 5, 10, 15, 20, 40, 80\}$, where a data set consists of noisy image and de-noised image pairs, $\{(\boldsymbol{y}^{(n)}, \boldsymbol{x}^{(n)})\}_{n=1}^{N}$. After training, we make predictions on the 200 test images using the MSE prediction rule. The process of splitting into a training and test set is repeated 10 times and the errors reported are computed across these 10 folds.

Figure 4.20 reports the MSE prediction error on digits 4 and 5 at two different noise levels. The top row is the prediction error on digit 4 and the bottom row is the error on digit 5. The left column contains results for a noise level of $nl = 3$ and the right column for the easier noise level of $nl = 5$. Rather surprisingly, we see that the WMB_8 method has larger prediction error than either the WMB_4 or WMB_2 method across all training set sizes. In addition, the gap between the WMB_8 method and the lower iBound methods is larger in the harder de-noising problems in the left column.

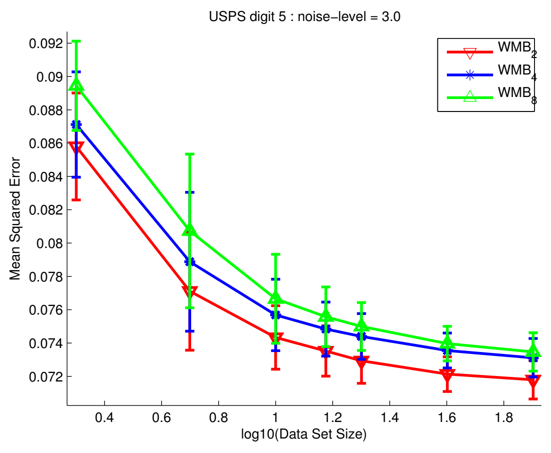
Figure 4.21 depicts the de-noised images produced by the WMB-based estimators for a couple of digits at the two different noise-levels, $nl = 3$ and $nl = 5$.



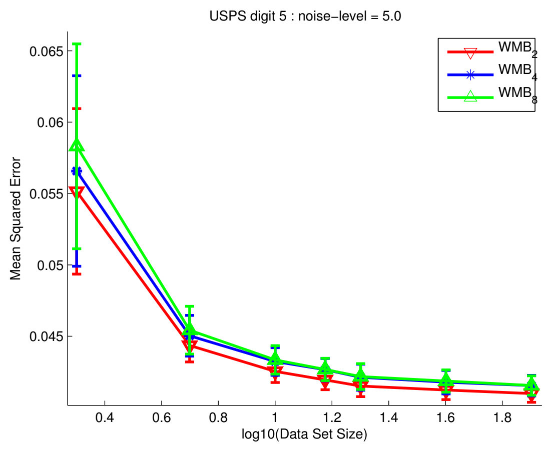
(a) Prediction error for digit 4 with $nl = 3$.



(b) Prediction error for digit 4 with $nl = 5$.



(c) Prediction error for digit 5 with $nl = 3$.



(d) Prediction error for digit 5 with $nl = 5$.

Figure 4.20: Prediction error of WMB-based estimators in CRF de-noising task.

CRF foreground-background labeling

We also considered a simple problem from computer vision, where one takes an RGB image with M pixels $\boldsymbol{x} \in \mathbb{R}^{3M}$ as input and wishes to predict a foreground-background labeling $\boldsymbol{y} \in \{0,1\}^M$ as output. In particular, we consider the collection of 328 images from the Weizmann horse database[8]. Each image in this data set contains a horse in some natural setting and our objective is to predict if each pixel i in an image is part of a horse ($y_i = 1$) or is background ($y_i = 0$).

[8]http://www.msri.org/people/members/eranb/

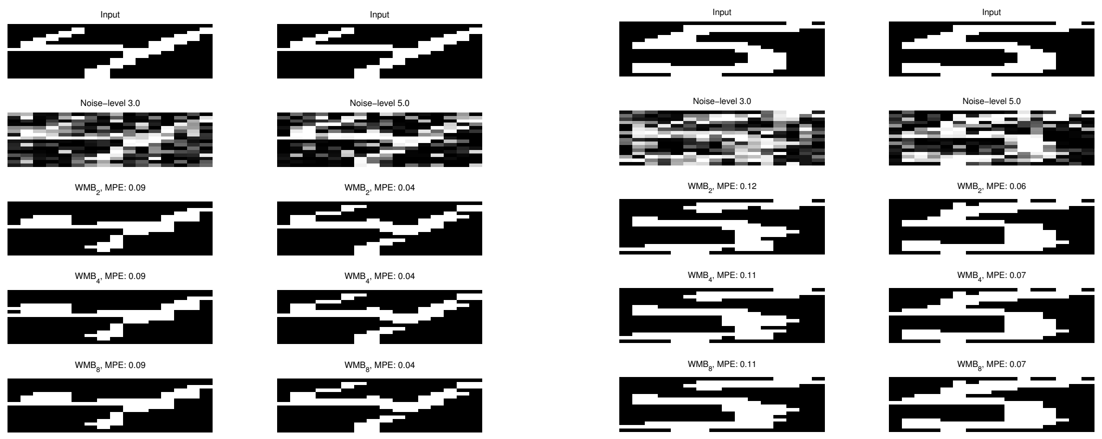


Figure 4.21: Sample images de-noised by the WMB.

Since we are only interested in predicting a foreground-background labeling of the RGB image $\boldsymbol{x}$, we directly model the conditional distribution $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ using the same pairwise CRF model used in the previous de-noising experiments (see equation (4.40)). The set of univariate and pairwise features used for foreground-background labeling are:

$$\boldsymbol{f}_u(y_i) = \begin{pmatrix} f_1(y_i,\boldsymbol{x}) = 1 \\ f_2(y_i,\boldsymbol{x}) = \mathrm{R}_i \\ f_3(y_i,\boldsymbol{x}) = \mathrm{G}_i \\ f_4(y_i,\boldsymbol{x}) = \mathrm{B}_i \\ f_5(y_i,\boldsymbol{x}) = \mathrm{hPos}_i \\ f_6(y_i,\boldsymbol{x}) = \mathrm{vPos}_i \end{pmatrix}, \quad \boldsymbol{f}_p(y_i,y_j) = \begin{pmatrix} f_1(y_i,y_j,\boldsymbol{x}) = 1 \,\&\, \mathrm{hz}(i,j) \\ f_2(y_i,y_j,\boldsymbol{x}) = \Delta_{\mathrm{RGB}}(i,j) < \frac{1}{20} \,\&\, \mathrm{hz}(i,j) \\ \vdots \\ f_{16}(y_i,y_j,\boldsymbol{x}) = \Delta_{\mathrm{RGB}}(i,j) < \frac{15}{20} \,\&\, \mathrm{hz}(i,j) \\ f_{17}(y_i,y_j,\boldsymbol{x}) = \Delta_{\mathrm{Sobel}}(i,j) < \frac{1}{20} \,\&\, \mathrm{hz}(i,j) \\ \vdots \\ f_{31}(y_i,y_j,\boldsymbol{x}) = \Delta_{\mathrm{Sobel}}(i,j) < \frac{15}{20} \,\&\, \mathrm{hz}(i,j) \\ f_{32}(y_i,y_j,\boldsymbol{x}) = 1 \,\&\, \mathrm{vt}(i,j) \\ f_{33}(y_i,y_j,\boldsymbol{x}) = \Delta_{\mathrm{RGB}}(i,j) < \frac{1}{20} \,\&\, \mathrm{vt}(i,j) \\ \vdots \\ f_{47}(y_i,y_j,\boldsymbol{x}) = \Delta_{\mathrm{RGB}}(i,j) < \frac{15}{20} \,\&\, \mathrm{vt}(i,j) \\ f_{48}(y_i,y_j,\boldsymbol{x}) = \Delta_{\mathrm{Sobel}}(i,j) < \frac{1}{20} \,\&\, \mathrm{vt}(i,j) \\ \vdots \\ f_{62}(y_i,y_j,\boldsymbol{x}) = \Delta_{\mathrm{Sobel}}(i,j) < \frac{15}{20} \,\&\, \mathrm{vt}(i,j) \end{pmatrix},$$

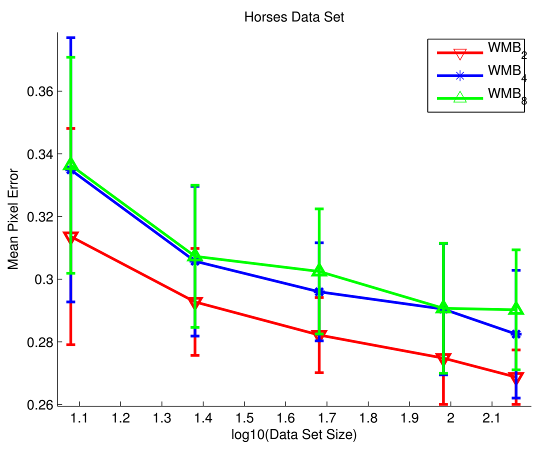
where R_i, G_i and B_i are the Red, Green, Blue intensities of pixel i scaled to the $[0, 1]$ interval, hPos_i and vPos_i are the horizontal and vertical position of pixel i also scaled to the interval $[0, 1]$, $\Delta_{\mathrm{RGB}}(i, j)$ is the L-2 difference of the (scaled) RGB intensities between pixel i and pixel j and $\Delta_{\mathrm{Sobel}}(i, j)$ is the max between the response of a Sobel edge filter at pixel i and the response of the filter at pixel j.

We expand this initial set of 6 unary features, $\boldsymbol{f}_u(y_i)$, into a set of 21 features, $\tilde{\boldsymbol{f}}_u(y_i)$ by adding all pairwise interaction terms (e.g. $\mathrm{R}_i \cdot \mathrm{hPos}_i$). As in the CRF de-noising experiment, we use one copy of these 21 features when $y_i = 0$ and a separate copy when $y_i = 1$, giving a total a total of 42 unary features, $\boldsymbol{f}_u = (\tilde{\boldsymbol{f}}_u(0), \tilde{\boldsymbol{f}}_u(1))$, in our CRF foreground-background model. We also expand the set of 62 edge features to a total of 248 edge features by using one set of features for each of the 4 possible joint configurations of (y_i, y_j). The foreground-background CRF model thus has $\boldsymbol{\theta} \in \mathbb{R}^{290}$.

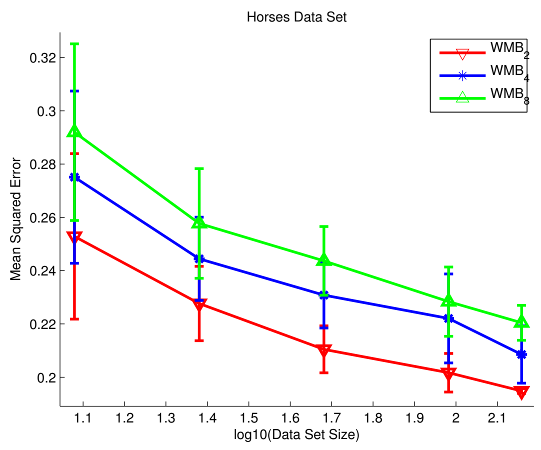
We split the horse data set of 328 images into 200 training images and 128 test images. Since the images are irregularly sized and typically quite large, we randomly sample 40×40 patches from the images. In particular, we create a training set of 500 patches by first choosing one of the 200 training images at random and then randomly choosing a fully-labeled 40×40 patch within that image. The test set is constructed in a similar fashion, but we randomly sample a total of 200 patches from the 128 test images instead.

We then randomly sample 144 patches from the training set of 500 patches and train our CRF model on subsets of size $N = \{12, 24, 48, 96, 144\}$ using the different WMB-based estimators. We evaluate each learned model on the 200 test patches using both the MSE and Hamming prediction rules. We repeat this process 5 times and report averages across these 5 folds.

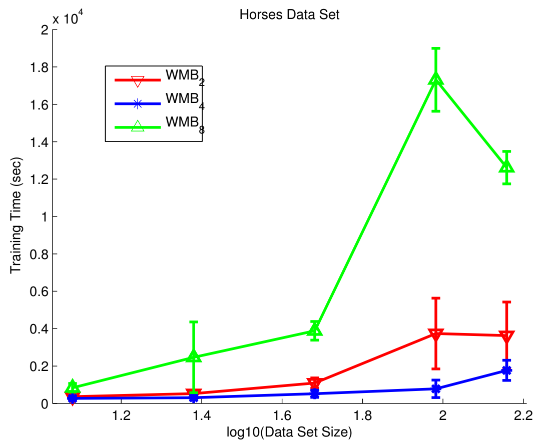
Figure 4.22 contains the results from this experiment. The top row contains the Mean Pixel (Hamming) Error 4.22a and the Mean Squared Error 4.22b as a function of training set size. As in the CRF de-noising experiments, we see that training and predicting with the WMB_8



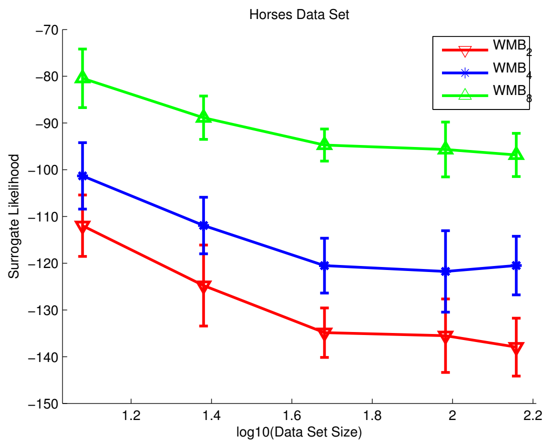
(a) Mean Pixel Error on horses data set.



(b) Mean Squared Error on horses data set.



(c) Training times on horses data set.



(d) Training likelihood on horses data set.

Figure 4.22: Results of foreground-background labeling in Weizmann horses data.

method yields greater error than training and predicting with lower iBound methods. Figure 4.22c shows the training times of the different methods. As expected, training takes the most time when using the WMB_8 method. However, we see that the WMB_4 method requires less training time than the WMB_2 method for most N. Finally, Figure 4.22d shows the training set likelihoods as a function of data set size. In this plot we see that the WMB_8 method has the highest likelihood for all N, which means that it provides a tighter bound on the log-partition function than the lower iBound methods as we would expect.

Figure 4.23 contains the predicted marginals for several test patches. Each column depicts the input patch, the true labeling and the marginals predicted by the WMB_2, WMB_4 and

WMB_8 methods, respectively. These patches qualitatively demonstrate the difference between the predictions made by each of the methods. In particular, test images 4 and 8 demonstrate how the WMB_8 method seems to prefer smoother transitions between foreground and background than the lower iBound methods.

4.6 Discussion

This chapter focused on the joint estimation and prediction problem, where the goal is to estimate the parameters of a model from data and then use the learned model to make predictions. Estimating model parameters via maximum likelihood is not possible in general graphical models due to the intractability of the log-partition function. However, if one replaces the true log-partition function with an approximation computed, for example, by Loopy BP, the result is an approximation to the true likelihood function known as the *surrogate* likelihood. As a result, different approximate inference methods will yield different surrogate likelihood functions which will differ from the true likelihood in different ways.

By maximizing the surrogate likelihood function we can produce an approximation to the true max likelihood parameter estimate. The difference between the true MLE and our approximation can be viewed as a source of error that will vary depending on our choice of approximate inference method. In this chapter, we considered inference methods that utilize a control parameter - the iBound - to trade computation for accuracy. In principle, as the iBound is increased, the accuracy of our approximation to the log-partition function and the likelihood function will improve and the error between the true MLE and our approximation should decrease. However, as demonstrated theoretically in Section 4.3 and empirically in Section 4.4, the gap between the surrogate and true likelihood is not the only source of error we face. In practice, we encounter model error, due to our inability to accurately describe the true data generating mechanism, and statistical error because our estimates are made using a finite sample.

Through a large empirical study, we examined the question of when higher iBound methods should be used in favor of lower iBound methods - i.e. when better inference actually means better learning and prediction. Through our study we observed that:

1. Smaller iBound estimators exhibit less variance than higher iBound methods. This occurs because low iBound methods are biased towards models (i.e. parameterizations) on which they provide accurate inference, which are inherently lower tree-width, simpler models. As a result, when regularization is held fixed they are less likely to overfit on small sized data sets and are more robust to model misspecification.

2. Higher iBound estimators exhibit smaller bias than lower iBound methods. As a result, they are preferred in situations where the gap between the surrogate and true likelihood functions is the primary source of error.

3. When training and predicting using CRF models, where there is likely lots of model error, lower iBound methods are far more efficient to train and may yield predictions that are as accurate, if not more accurate, than higher iBound methods when regularization is held fixed. In other words, when model error is the dominant source of error and there is little to be gained by manipulating the bias-variance trade-off, it may be advantageous to utilize lower iBound surrogates as they yield more computationally efficient training methods.

The experiments conducted in this chapter revealed many directions for future study. First, there appears to be a rich relationship between the complexity of the inference method used during training and regularization. We saw in Section 4.4.2, for example, that training with a low iBound estimator led to an increase in the effective level of regularization. In other words, even if we have a complex model, a given inference-based estimator seems to find parameters that lie in some particular complexity class. For example, training using the WMB surrogate with an iBound of 2 will be biased towards models (parameterizations) that are roughly tree-like, as these are the class of models on which iBound 2 inference is accurate. Of course, if our true distribution is not tree-like the WMB 2 surrogate will be heavily biased, but it will also be less susceptible to fluctuations in the empirical moments. A direction worth exploring is to better understand this interplay between the choice of surrogate and its inherent regularization.

Another idea, along these same lines, is that the inference method and surrogate used should adapt to the model being learned and the data used to learn it. For example, the mini-bucket partitioning scheme which underlies the WMB method can be interpreted as introducing independences that make inference feasible[9]. Intuitively, it seems like the partitioning scheme used could consult both the model and training data so as to capture important variable dependencies, while ignoring unimportant ones. In this way, we could potentially trade computational efficiency for statistical efficiency in a more fine-grained manner.

One final idea is to develop a statistical test to determine if increasing the iBound will in fact be beneficial. In other words, we envisage an iterative learning scheme where we initially train using some cheap, low iBound method and then must determine whether to continue training using a higher iBound method, or terminate and use the current parameter estimates for prediction. Since the total estimation error decomposes into bias and variance terms, we should be able to estimate whether it will be beneficial to increase computation by estimating the reduction in bias and increase in variance of using a higher iBound method.

[9]See Section 1.4.1 for a discussion of the WMB splitting semantics.

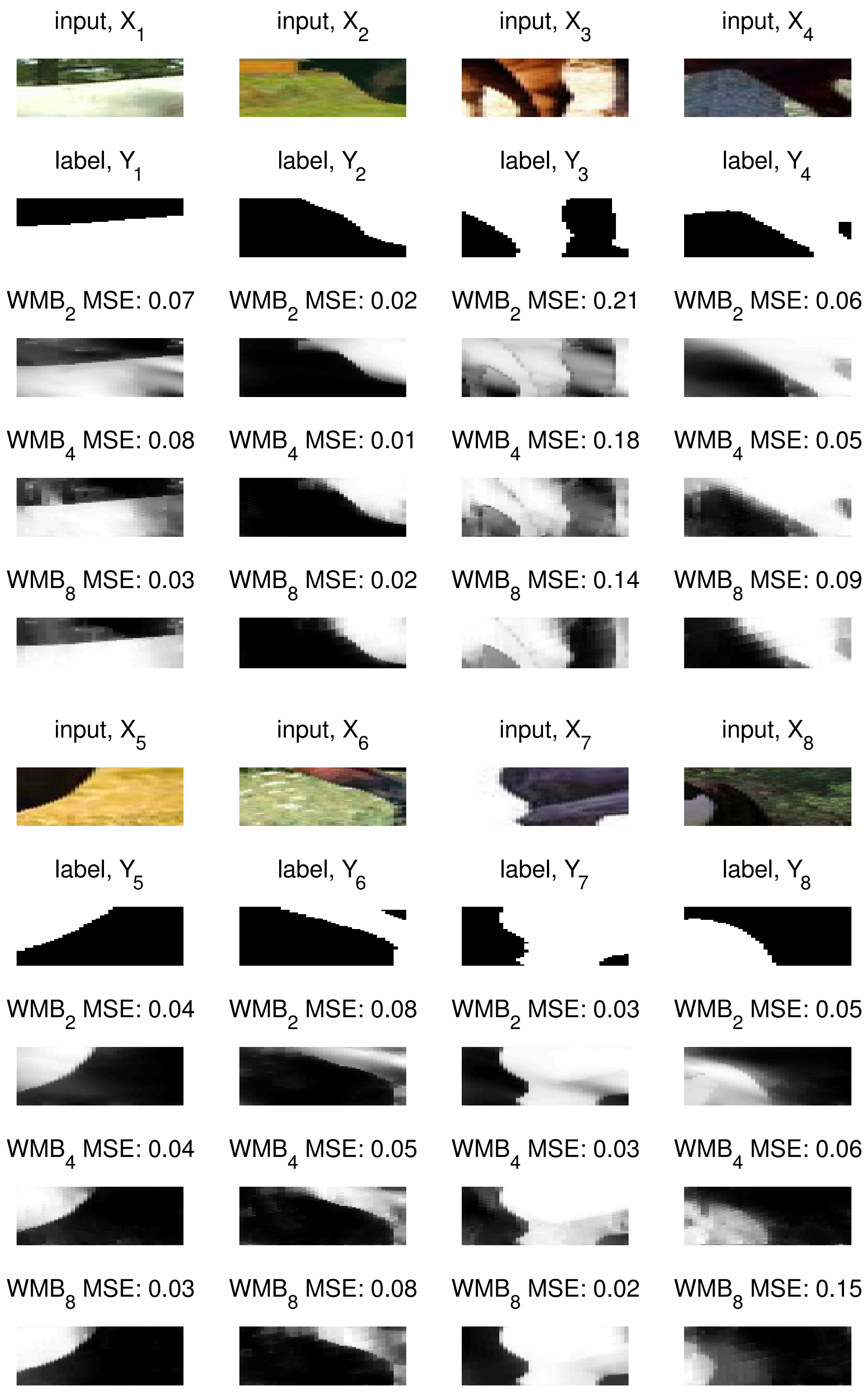


Figure 4.23: Sample foreground-background labelings in Weizmann horses data.

Chapter 5

Conclusion

Graphical models have become the favored approach for representing and reasoning about probability distributions over many random variables. Unfortunately, inference is fundamentally intractable for most models which accurately describe the complex dependencies of data. Developing better approximate inference and learning algorithms is thus central to improving the fidelity of our learned models and the quality of predictions made under those learned models. In this thesis, we advocated a bottom-up approach to approximate inference and learning, where we start with the *natural*, initial approximation to our problem and then improve upon the initial approximation through increased computation.

In Chapter 2 we focused on solving the weighted matching problem using max-product BP. The weighted matching problem is easily formulated as an Integer Linear Program (ILP) and its natural LP relaxation replaces integrality constraints on the mass assigned to each edge, with linear inequality constraints. When the natural LP relaxation of the weighted matching problem is tight, it can be solved by the max-product algorithm[5, 76]. However, when the natural LP relaxation is loose, max-product is no longer provably exact. As a result, we devised a cutting-plane approach that iteratively adds constraints to tighten the relaxation by removing fractional optima and developed a max-product algorithm that provably solves the weighted matching problem when the LP relaxation is made tight. This result expands the set of graphical models that are provably solvable by the max-product algorithm and have improved our understanding of the properties needed for max-product to be convergent and correct.

In Chapter 3 we focused on calculating marginal probabilities in a graphical model using GBP. GBP is a class of approximate inference algorithm that builds upon Loopy BP by passing messages between larger clusters of variables. GBP offers the promise to yield marginal estimates that are far more accurate than Loopy BP, but is also very sensitive to the choice of regions (or clusters) used. We connected the problem of choosing the outer regions which define a Loop-Structured Region Graph (Loop-SRG) to that of finding a fundamental cycle basis in the graph underlying a pairwise graphical model. We showed that choosing such a collection of outer regions give GBP approximations that behave sensibly when a model has extremely weak (non-Singularity) and extremely strong (over-counting number unity) factors. We then proposed a new criterion – tree-robustness – that corresponds to finding a special class of fundamental cycle bases which give GBP approximations that are, in some sense, no worse than Loopy BP when the factors defining a model induce a tree. We then demonstrated empirically that GBP can give accurate estimates when a Loop-SRG is formed from a fundamental basis and that the estimates can be improved by choosing a fundamental basis that is at least partially tree-robust.

In Chapter 4 we focused on learning the parameters of a model from data. Maximum likelihood estimation in graphical models is difficult due to the intractability of computing the log-partition function and marginals. In *surrogate* likelihood training, one approximates these quantities using an approximate inference algorithm. We focused on approximate inference methods that utilize a control parameter to trade computation for accuracy and examined when investing more computation leads to more accurate parameter estimates and models that yield more accurate predictions. The accuracy of approximate inference is only one factor affecting our ability to learn a *good* model, so we introduced a framework for analyzing and comparing the different sources of error in likelihood-based learning. We then used this framework to conduct a large empirical evaluation of different approximate inference-based learning approaches and demonstrated that better inference does not necessarily result in learning a better model, for small sized data sets and mis-specified models. This work also exposed an interesting bias-variance trade-off between low computation inference methods and high computation inference methods.

We close by remarking that the increasing availability of massive data sets brings incredible opportunities to the graphical models community. Capitalizing on all of this data requires increasingly complex models that can better explain the phenomena occurring in the data. However, the more expressive a model is and the better it can represent real-life distributions, the harder inference becomes in that model. In addition, as we saw in Chapter 4, there is a fundamental trade-off between statistical efficiency and computational efficiency when learning richer models: more data is needed to reliably estimate and make predictions under a more complex model, but more data, of course, requires better algorithms if training is to occur within a particular computational budget. As a result, there is and will continue to be an increasing need for better approximate inference methods!

Bibliography

[1] IBM ILOG CPLEX Optimizer, 2014.

[2] S. Aji and R. McEliece. The generalized distributive law and free energy minimization. In *Proceedings of the Allerton Conference on Communication, Control, and Computing*, 2001.

[3] F. Barahona. Finding ground states in random-field ising ferromagnets. *Journal of Physics A*, 18:673–675, 1985.

[4] M. Bayati, C. Borgs, J. T. Chayes, and R. Zecchina. Belief propagation for weighted b-matchings on arbitrary graphs and its relation to linear programs with integer solutions. *SIAM J. Discrete Math.*, 25(2):989–1011, 2011.

[5] M. Bayati, D. Shah, and M. Sharma. Max-product for maximum weight matching: Convergence, correctness, and lp duality. *IEEE Trans. Inf. Theor.*, 54(3):1241–1251, 2008.

[6] J. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B*, 36(2):192–236, 1974.

[7] J. Besag. Statistical analysis of non-lattice data. *The Statistician*, 24(3):179–195, 1975.

[8] R. G. Bland, D. Goldfarb, and M. J. Todd. The Ellipsoid Method: A Survey. *Operations Research*, 29(6):1039–1091, 1981.

[9] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *International Symposium on Mathematical Foundations of Computer Science*, pages 19–36, 1997.

[10] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *NIPS*, pages 161–168, 2008.

[11] A. Bouchard-Côté and M. I. Jordan. Optimization of structured mean field objectives. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 67–74. AUAI Press, 2009.

[12] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[13] G. Casella and R. Berger. *Statistical Inference.* Duxbury, 2 edition, 2002.

[14] K. Chandrasekaran, L. A. Vgh, and S. Vempala. The cutting plane method is polynomial for perfect matchings. In *FOCS*, pages 571–580. IEEE Computer Society, 2012.

[15] A. Choi and A. Darwiche. A variational approach for approximating bayesian networks by edge deletion. In *UAI*, pages 80–89, 2006.

[16] O. Dagdeviren and K. Erciyes. Graph matching-based distributed clustering and backbone formation algorithms for sensor networks. *Computer Journal*, 53(10):1553–1575, 2010.

[17] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2:393–410, 1954.

[18] A. Darwiche. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[19] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.

[20] R. Dechter, K. Kask, and R. Mateescu. Iterative join-graph propagation. In *UAI*, pages 128–136. Morgan Kaufmann, 2002.

[21] R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM*, 50:107–153, 2003.

[22] J. Dillon and G. Lebanon. Statistical and computational tradeoffs in stochastic composite likelihood. In *AISTATS*, 2009.

[23] J. Domke. Parameter learning with truncated message-passing. In *CVPR*, 2011.

[24] J. Domke. Learning graphical model parameters with approximate marginal inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(10):2454–2467, 2013.

[25] J. Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449–467, 1965.

[26] G. Elidan, I. McGraw, and D. Koller. Residual belief propagation: Informed scheduling for asynchronous message passing. In *UAI*, 2006.

[27] J. Feldman, M. J. Wainwright, and D. R. Karger. Using linear programming to decode binary linear codes. *IEEE Transactions on Information Theory*, 51:954–972, 2005.

[28] D. Gamarnik, D. Shah, and Y. Wei. Belief propagation for min-cost network flow: Convergence & correctness. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 279–292, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[29] Y. Gao. The degree distribution of random k-trees. *Theoretical Computer Science*, 410:688–695, 2009.

[30] B. Gidas. Consistency of maximum likelihood and pseudolikelihood estimators for gibbs distributions. Springer, 1988.

[31] A. Globerson and T. Jaakkola. Approximate inference using conditional entropy decompositions. In *AISTATS*, 2007.

[32] A. Globerson and T. Jaakkola. Convergent propagation algorithms via oriented trees. In *UAI*, 2007.

[33] A. Globerson and T. Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP relaxations. In *Neural Information Processing Systems (NIPS)*, 2007.

[34] R. E. Gomory. Outline of an algorithm for integer solutions too linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[35] R. E. Gomory. An algorithm for integer solutions to linear programs. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, 1963.

[36] D. Greig, B. Porteous, and A. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society B*, 51(2):271–279, 1989.

[37] M. Grötschel and O. Holland. Solving matching problems with linear programming. *Mathematical Programming*, 33(3):243–259, 1985.

[38] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations research*, 32(6):1195–1220, 1984.

[39] Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2014.

[40] T. Hazan and A. Shashua. Convergent message-passing algorithms for inference over general graphs with convex free energies. In *UAI*, 2008.

[41] U. Heinemann and A. Globerson. What cannot be learned with bethe approximations. In *UAI*, 2011.

[42] T. Heskes. Stable xed points of loopy belief propagation are minima of the bethe free energy. In *NIPS*, 2003.

[43] T. Heskes. Convexity arguments for efficient minimization of the Bethe and Kikuchi free energies. *Journal of Machine Learning Research*, 26(153-190), 2006.

[44] B. C. Huang and T. Jebara. Loopy belief propagation for bipartite maximum weight b-matching. In M. Meila and X. Shen, editors, *AISTATS*, volume 2, pages 195–202, 2007.

[45] A. Hyvarinen. Consistency of pseudolikelihood estimation of fully visible boltzmann machines. *Neural Computation*, 18(10):2283–2292, 2006.

[46] A. T. Ihler, J. W. Fisher, and A. S. Willsky. Message errors in belief propagation. In *NIPS*, pages 609–616, 2005.

[47] J. Johnson. *Convex relaxation methods for graphical models: Lagrangian and maximum entropy approaches.* PhD thesis, MIT, Department of Electrical Engineering and Computer Science, 2008.

[48] M. I. Jordan, editor. *Learning in Graphical Models.* MIT Press, Cambridge, MA, USA, 1999.

[49] K. Kask, A. Gelfand, L. Otten, and R. Dechter. Pushing the power of stochastic greedy ordering schemes for inference in graphical models. In *AAAI Conference on Artificial Intelligence*, pages 1043–1048, 2011.

[50] T. Kavitha, C. Liebchen, K. Mehlhorn, D. Michail, R. Rizzi, T. Ueckerdt, and K. Zweig. Cycle bases in graphs characterization, algorithms, complexity, and applications. *Computer Science Review*, 3(4):199–243, 2009.

[51] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademiia Nauk SSSR*, 244:1093–1096, 1979.

[52] R. Kikuchi. A theory of cooperative phenomena. *Physical Review*, 81(6):988–1003, 1951.

[53] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques.* MIT Press, Cambridge, MA, USA, firrst edition edition, 2009.

[54] V. Kolmogorov. Blossom V: A new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009.

[55] V. Kolmogorov and M. J. Wainwright. On the optimality of tree-reweighted max-product message-passing. In *UAI*, pages 316–323, 2005.

[56] F. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. on Info. Theory*, 47(2):498 –519, 2001.

[57] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[58] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50(2):157–224, 1988.

[59] P. Liang and M. Jordan. An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators. In *ICML*, pages 584–591, 2008.

[60] B. G. Lindsay. Composite likelihood methods. *Contemporary Mathematics*, 80, 1988.

[61] Q. Liu and A. Ihler. Bounding the partition function using hölder's inequality. In *ICML*, pages 849–856, 2011.

[62] R. Mateescu, K. Kask, V. Gogate, and R. Dechter. Join-graph propagation algorithms. *J. Artif. Intell. Res. (JAIR)*, 37, 2010.

[63] R. McEliece, D. MacKay, and J. Cheng. Turbo decoding as as an instance of Pearls belief propagation algorithm. *IEEE Journal on Selected Areas in Communication*, 16(2):140–152, 1998.

[64] T. Meltzer, A. Globerson, and Y. Weiss. Convergent message passing algorithms: a unifying view. In *UAI*, pages 393–401, 2009.

[65] E. Mezuman, D. Tarlow, A. Globerson, and Y. Weiss. Tighter linear program relaxations for high order graphical models. In *Proc. of the 29th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2013.

[66] J. M. Mooij and H. J. Kappen. Sufficient conditions for convergence of the sum-product algorithm. *IEEE Trans. Inf. Theor.*, 53(12):4422–4437, 2007.

[67] T. Morita. Formal structure of the cluster variation method. *Progress of Theoretical Physics supplement*, 115:27–39, 1994.

[68] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *UAI*, pages 467–475, 1999.

[69] S. Nowozin. Constructing composite likelihoods in general random fields. In *ICML Workshop on Inferning: Interactions between Inference and Learning*, 2013.

[70] M. W. Padberg and M. R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1):pp. 67–80, 1982.

[71] P. Pakzad and V. Anantharam. Estimation and marginalization using kikuchi approximation methods. *Neural Computation*, 17(8):1836–1873, 2005.

[72] J. Pearl. *Probabilistic reasoning in intelligent systems.* Morgan Kaufmann, 1988.

[73] P. Pletscher, S. Nowozin, P. Kohli, and C. Rother. Putting map back on the map. In R. Mester and M. Felsberg, editors, *Pattern Recognition*, volume 6835 of *Lecture Notes in Computer Science*, pages 111–121. Springer Berlin Heidelberg, 2011.

[74] C. Robert. *The Bayesian Choice: From Decision Theoretic Foundations to Computational Implementation.* Springer, 2001.

[75] S. Sanghavi, D. Malioutov, and A. Willsky. Belief propagation and lp relaxation for weighted matching in general graphs. *IEEE Trans. Inf. Theor.*, 57(4):2203–2212, 2011.

[76] S. Sanghavi, D. Malioutov, and A. S. Willsky. Linear programming analysis of loopy belief propagation for weighted matching. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1273–1280. 2007.

[77] S. Sanghavi, D. Shah, and A. S. Willsky. Message passing for max-weight independent set. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1281–1288. 2007.

[78] M. I. Schlesinger. Syntactic analysis of two-dimensional visual signals in noisy conditions (in russian). *Kibernetika*, 4:113–130, 1976.

[79] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, New York, 1987.

[80] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer-Verlag, Berlin, 2003.

[81] J. Shin, A. E. Gelfand, and M. Chertkov. A graphical transformation for belief propagation: Maximum weight matchings and odd-sized cycles. In C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, editors, *NIPS*, pages 2022–2030, 2013.

[82] D. Sontag. *Approximate Inference in Graphical Models using LP Relaxations.* PhD thesis, MIT, Department of Electrical Engineering and Computer Science, 2010.

[83] D. Sontag and T. Jaakkola. New Outer Bounds on the Marginal Polytope. In *Neural Information Processing Systems (NIPS)*, 2007.

[84] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Trans. on Automatic Control*, 37(12), 1992.

[85] M. Trick. *Networks with Additional Structured Constraints.* PhD thesis, Georgia Institute of Technology, School of Industrial and Systems Engineering, 1987.

[86] J. T. Wainwright, M. and A. Willsky. A new class of upper bounds on the log partition function. *IEEE Trans. Info. Theory*, 51(7):2313–2335, July 2005.

[87] M. J. Wainwright. Estimating the "wrong" graphical model: Benefits in the computation-limited setting. *J. Mach. Learn. Res.*, 7:1829–1859, Dec. 2006.

[88] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. Treebased reparameterization framework for analysis of sumproduct and related algorithms. *IEEE Trans. Info. Theory*, 49(5):1120–1146, 2003.

[89] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. Map estimation via agreement on trees: message-passing and linear programming. *IEEE Trans. on Information Theory*, 51:2005, 2005.

[90] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1-2):1–305, 2008.

[91] Y. Weiss. Belief propagation and revision in networks with loops. Technical report, Cambridge, MA, USA, 1997.

[92] Y. Weiss and W. T. Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744, 2001.

[93] Y. Weiss, C. Yanover, and T. Meltzer. Linear programming and variants of belief propagation. In A. Blake, P. Kohli, and C. Rother, editors, *Markov Random Fields for Vision and Image Processing*, pages 95–108. MIT Press, 2011.

[94] M. Welling. On the choice of regions for generalized belief propagation. In *UAI*, pages 585–592, 2004.

[95] M. Welling, T. Minka, and Y. W. Teh. Structured region graphs: Morphing EP into GBP. In *UAI*, pages 607–614, 2005.

[96] M. Welling and Y. Teh. Belief optimization for binary networks: a stable alternative to loopy belief propagation. In *UAI*, pages 554–561, 2001.

[97] T. Werner. A linear programming approach to max-sum problem: a review. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(7):1165–1179, 2007.

[98] T. Werner. High-arity Interactions, Polyhedral Relaxations, and Cutting Plane Algorithm for Soft Constraint Optimisation (MAP-MRF). In *CVPR 2008: Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 109–116, 2008.

[99] W. L. Winston. *Operations Research: Applications and Algorithms.* Thomson Brooks/Cole, 2004.

[100] M. Yannakakis. Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43(3):441–466, 1991.

[101] J. Yedidia, W. Freeman, and Y. Weiss. Generalized belief propagation. In *NIPS*, volume 13, 2000.

[102] J. Yedidia, W. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51:2282–2312, 2005.

[103] A. L. Yuille. Cccp algorithms to minimize the bethe and kikuchi free energies: Convergent alternatives to belief propagation. *Neural Computation*, 14(7):1691–1722, 2002.

[104] X. Zhou and S. C. Schmidler. Bayesian parameter estimation in ising and potts models: A comparative study with applications to protein modeling. Technical report, Duke University, 2009.

Appendix A

Overview of Variational Inference

Computing the log-partition function is typically viewed as an intractable summation task: $\log Z(\boldsymbol{\theta}) = \log \sum_{\boldsymbol{y}} p(\boldsymbol{y}; \boldsymbol{\theta})$. We will now convert computing the log-partition function to an optimization problem using the framework of variational inference. The development borrows heavily from [90, 24].

Let $p(\boldsymbol{y}; \boldsymbol{\theta})$ and $p(\boldsymbol{y}; \boldsymbol{\theta}')$ be two distributions with canonical parameters $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$, respectively. Consider evaluating the KL-divergence between these two distributions:

$$\begin{aligned} D_{\mathrm{KL}}\left(p(\boldsymbol{y}; \boldsymbol{\theta}) || p(\boldsymbol{y}; \boldsymbol{\theta}')\right) =& E_{\boldsymbol{\theta}}\left[\log \frac{p(\boldsymbol{y}; \boldsymbol{\theta})}{p(\boldsymbol{y}; \boldsymbol{\theta}')}\right] \\ =& E_{\boldsymbol{\theta}}\left[\log p(\boldsymbol{y}; \boldsymbol{\theta})\right] - E_{\boldsymbol{\theta}}\left[\boldsymbol{s}(\boldsymbol{y})\boldsymbol{\theta}' + \log Z(\boldsymbol{\theta}')\right] \\ =& - H(\boldsymbol{\mu}) - \boldsymbol{\mu} \cdot \boldsymbol{\theta}' + \log Z(\boldsymbol{\theta}') \end{aligned} \tag{A.1}$$

where $\boldsymbol{\mu} = \boldsymbol{\mu}(\boldsymbol{\theta}) = E_{\boldsymbol{\theta}}\left[\boldsymbol{s}(\boldsymbol{y})\right]$ is the mean vector corresponding to parameter setting $\boldsymbol{\theta}$ and $H(\boldsymbol{\mu})$ is the entropy of the distribution $p(\boldsymbol{y}; \boldsymbol{\theta})$, written as a function of the mean vector,

$$-H(\boldsymbol{\mu}) = E_{\boldsymbol{\theta}}\left[\log p(\boldsymbol{y}; \boldsymbol{\theta})\right] = E_{\boldsymbol{\theta}}\left[\boldsymbol{s}(\boldsymbol{y}) \cdot \boldsymbol{\theta} - \log Z(\boldsymbol{\theta})\right] = \boldsymbol{\mu} \cdot \boldsymbol{\theta} - \log Z(\boldsymbol{\theta}). \tag{A.2}$$

The KL-divergence between two distributions is always non-negative, $D_{\text{KL}}\left(p(\boldsymbol{y};\boldsymbol{\theta})||p(\boldsymbol{y};\boldsymbol{\theta}')\right) \geq 0$. Assume that the sufficient statistics $\boldsymbol{s}(\boldsymbol{y})$ are linearly independent, so that both distributions are in a minimal exponential family representation. Then, $D_{\text{KL}}\left(p(\boldsymbol{y};\boldsymbol{\theta})||p(\boldsymbol{y};\boldsymbol{\theta}')\right) = 0$, if and only if $\boldsymbol{\theta} = \boldsymbol{\theta}'$. As a result, we can write that

$$-H(\boldsymbol{\mu}) = \sup_{\boldsymbol{\theta}'\in\mathbb{R}^d} \left[-\boldsymbol{\mu}\cdot\boldsymbol{\theta}' + \log Z(\boldsymbol{\theta}')\right] = \inf_{\boldsymbol{\theta}'\in\mathbb{R}^d} \left[\boldsymbol{\mu}\cdot\boldsymbol{\theta}' - \log Z(\boldsymbol{\theta}')\right] \tag{A.3}$$

This expression for the negative entropy is the conjugate dual of the log-partition function, which means that we can re-write this as [90]:

$$\log Z(\boldsymbol{\theta}) = \sup_{\boldsymbol{\mu}'} \left[\boldsymbol{\mu}'\cdot\boldsymbol{\theta} + H(\boldsymbol{\mu}')\right]. \tag{A.4}$$

Now the entropy is undefined if the vector $\boldsymbol{\mu}'$ does not correspond to a valid joint distribution. As a result, we restrict attention to vectors $\boldsymbol{\mu}' \in \mathcal{M}$, where $\mathcal{M} = \{\boldsymbol{\mu}' \in \mathbb{R}^d \mid \exists\boldsymbol{\theta} \text{ s.t. } \boldsymbol{\mu}' = \boldsymbol{\mu}(\boldsymbol{\theta})\}$ is the marginal polytope, which is the set of mean vectors $\boldsymbol{\mu}'$ that can arise from some joint distribution $p(\boldsymbol{y};\boldsymbol{\theta})$. This gives the result in (1.36).

Appendix B

Proof of Convergence of BP on the auxiliary Weighted Matching Model

This appendix provides the proof of Theorem 2.4. We also establish the convergence time of the BP algorithm under the *transformed* graphical model in (2.26) (see Lemma B.1). Our proof requires careful study of the computation tree induced by BP with appropriate truncations at its leaves.

B.1 Main Lemma for Proof of Theorem 2.4

Let us introduce the following auxiliary LP over the new graph and weights.

$$\begin{aligned}
\text{match-aux-LP}: \quad & \max \sum_{e\in E'} w'_e y_e \\
\text{s.t.} \quad & \sum_{e\in\delta(i)} y_e \le 1,\ \ \forall i\in V, \quad y_e\in[0,1],\ \ \forall e\in E', \qquad \text{(B.1)}\\
& \sum_{j\in V(C)} (-1)^{d_C(j,e)} y_{i_C,j} \in [0,2],\ \ \forall e\in E(C), \qquad \sum_{e\in\delta(i_C)} y_e \le |C|-1,\ \ \forall C\in\mathcal{C}. \qquad \text{(B.2)}
\end{aligned}$$

Once again, consider the following one-to-one linear mapping between the original edge variables $\boldsymbol{x}=\{x_e : e\in E\}$ and the new edge variables $\boldsymbol{y}=\{y_e : e\in E'\}$:

$$y_e = \begin{cases} \sum_{e'\in E(C)\cap\delta(i)} x_{e'} & \text{if } e=(i,i_C) \\ x_e & \text{otherwise} \end{cases} \qquad x_e = \begin{cases} \frac{1}{2}\sum_{j\in V(C)}(-1)^{d_C(j,e)} y_{i_C,j} & \text{if } e\in\bigcup_{C\in\mathcal{C}} E(C) \\ y_e & \text{otherwise} \end{cases}.$$

Since $\sum\limits_{e\in E} w_e x_e = \sum\limits_{e\in E'} w'_e y_e$ and the mapping between $\boldsymbol{x}$ and $\boldsymbol{y}$ is linear, it is easy to verify that every feasible solution in match-blossom-LP induces a feasible solution in match-aux-LP of the same total weight (and vice versa). Thus, we refer to aux-blossom-LP as being *equivalent* to aux-match-LP. As a result, if the solution $\boldsymbol{x}^{\text{match-blossom-LP}}$ of match-blossom-LP is unique and integral, the solution $\boldsymbol{y}^{\text{match-aux-LP}}$ of match-aux-LP will be as well, i.e., $\boldsymbol{y}^{\text{match-aux-LP}} = \boldsymbol{y}^\star$. Hence, (2.28) in Theorem 2.4 follows.

Furthermore, since the solution $\boldsymbol{y}^\star$ to match-aux-LP is unique and integral, there exists $c>0$ such that

$$c = \inf_{\boldsymbol{y}\neq\boldsymbol{y}^\star\,:\,\boldsymbol{y}\text{ is feasible for match-aux-LP}} \frac{\boldsymbol{w}'\cdot(\boldsymbol{y}^\star-\boldsymbol{y})}{|\boldsymbol{y}^\star-\boldsymbol{y}|},$$

where $\boldsymbol{w}'=\{w'_e \mid e\in E'\}$. Using this notation, we establish the following lemma characterizing the performance of the max-product BP algorithm on the model (2.26). Theorem 2.4 follows directly from this lemma.

Lemma B.1. *If the solution $\boldsymbol{y}^{\text{match-aux-LP}}$ of match-aux-LP is integral and unique, i.e., $\boldsymbol{y}^{\text{match-aux-LP}} = \boldsymbol{y}^\star$, then*

- *If $y_e^\star = 1$, $b_e^t(1) > b_e^t(0)$ for all $t > \frac{6w'_{\max}}{c} + 6$,*
- *If $y_e^\star = 0$, $b_e^t(1) < b_e^t(0)$ for all $t > \frac{6w'_{\max}}{c} + 6$,*

where $b_e^t(\cdot)$ denotes the BP belief of edge e at time t under the graphical model (2.26) and $w'_{\max} = \max_{e \in E'} |w'_e|$.

B.2 Proof of Main Lemma B.1

This section provides the complete proof of Lemma B.1. We focus here on the case of $y_e^* = 1$, while translation of the result to the opposite case of $y_e^* = 0$ is straightforward. To derive a contradiction, assume that $b_e^t(1) \leq b_e^t(0)$ and construct a computational tree $T_e(t)$ of depth $t+1$, using the following scheme:

Construct Computation Tree:

1. Add a copy of $Y_e \in \{0,1\}$ as the (root) variable (with variable function $e^{w'_e Y_e}$).
2. Repeat the following t times for each leaf variable Y_e on the current tree:

   2-1. For each $i \in V$ such that $e \in \delta(i)$ and ψ_i is not associated to Y_e of the current model, add ψ_i as a factor (function) with copies of $\{Y_{e'} \in \{0,1\} : e' \in \delta(i) \setminus e\}$ as child variables (with corresponding variable functions, i.e., $\{e^{w'_{e'} Y_{e'}}\}$).

   2-2. For each $C \in \mathcal{C}$ such that $e \in \delta(i_C)$ and ψ_C is not associated to Y_e of the current model, add ψ_C as a factor (function) with copies of $\{Y_{e'} \in \{0,1\} : e' \in \delta(i_C) \setminus e\}$ as child variables (with corresponding variable functions, i.e., $\{e^{w'_{e'} Y_{e'}}\}$).

This construction was illustrated in Figure 2.9 in Section 2.4.4. We now utilize the following fact relating BP and computation trees: the BP MAP estimate at time t is the MAP assignment of the computation tree of depth t [91]. Since $b_e^t(1) \le b_e^t(0)$ by assumption, we then know that there exists a MAP configuration $\boldsymbol{y}^{\text{TMAP}}$ on $T_e(t)$ with $y_e^{\text{TMAP}} = 0$ at the root variable. We construct a new assignment, $\boldsymbol{y}^{\text{NEW}}$ on the computational tree $T_e(t)$ as follows:

Flipping Procedure

1. Initially, set $\boldsymbol{y}^{\text{NEW}} \leftarrow \boldsymbol{y}^{\text{TMAP}}$ and e is the root of the tree.
2. $\boldsymbol{y}^{\text{NEW}} \leftarrow \texttt{FLIP}_e(\boldsymbol{y}^{\text{NEW}})$.
3. **For** each child factor ψ of e, which is either a vertex factor, ψ_i, if $e \in \delta(i)$ or a blossom factor, ψ_C, if $e \in \delta(i_C)$:
   (a) **If** ψ is satisfied by $\boldsymbol{y}^{\text{NEW}}$ and $\texttt{FLIP}_e(\boldsymbol{y}^\star)$ (i.e., $\psi(\boldsymbol{y}^{\text{NEW}}) = \psi(\texttt{FLIP}_e(\boldsymbol{y}^\star)) = 1$), then do nothing.
   (b) **Else if** there is a child e' of e through factor ψ such that $\boldsymbol{y}_{e'}^{\text{NEW}} \neq \boldsymbol{y}_{e'}^\star$ and ψ is satisfied by $\texttt{FLIP}_{e'}(\boldsymbol{y}^{\text{NEW}})$ and $\texttt{FLIP}_{e'}(\texttt{FLIP}_e(\boldsymbol{y}^\star))$, then go to the step 2 with $e \leftarrow e'$.
   (c) **Else**, report ERROR.

An illustration of this flipping procedure was provided in Figure 2.9 in Section 2.4.4. In the construction, $\texttt{FLIP}_e(\boldsymbol{y})$ is the $0-1$ vector made by flipping (i.e., changing from 0 to 1 or 1 to 0) the assignment of e in $\boldsymbol{y}$. By the construction of the computation tree, there is exactly one child factor ψ in Step 3. Furthermore, we only choose one child e' in Step 3 (b) (even though there are many possible candidates). For this reason, the Flipping Procedure induces a path structure P in tree $T_e(t)$.[1]

We now state the following lemma for the construction of $\boldsymbol{y}^{\text{NEW}}$.

[1] P may not be an alternating path since both y_e^{NEW} and its child $y_{e'}^{\text{NEW}}$ can be flipped the same way.

Lemma B.2. ERROR *is never reported in the Flipping Procedure described above.*

Proof. This lemma tells us that if we follow the Flipping Procedure, we can flip the assignments along a path so that they agree with the MAP assignment without needing to backtrack. With this in mind, it is easy to verify that *ERROR* is not reported when $\psi = \psi_i$ is a vertex factor. We therefore only provide a proof for the case when $\psi = \psi_C$ is a blossom factor. We also assume that y_e^{NEW} is flipped from $1 \to 0$ (i.e., $y_e^\star = 0$); the proof for the case $0 \to 1$ follows in a similar manner. First, one can observe that $\boldsymbol{y}$ satisfies ψ_C if and only if $\boldsymbol{y}$ is the $0-1$ indicator vector of a union of disjoint even paths in the cycle C. Since $\boldsymbol{y}_e^{\text{NEW}}$ is flipped from $1 \to 0$, the even path including e is broken into an even (possibly, empty) path and an odd (always, non-empty) path. We consider two cases: (a) there exists an edge e' within the odd path (i.e., $y_{e'}^{\text{NEW}} = 1$) such that $y_{e'}^\star = 0$ and flipping $y_{e'}^{\text{NEW}}$ from $1 \to 0$ breaks the odd path into 2 even (disjoint) paths; (b) there exists no such e' within the odd path.

For case (a), it is easy to see that we can maintain the structure of disjoint even paths in $\boldsymbol{y}^{\text{NEW}}$ after flipping $y_{e'}^{\text{NEW}}$ as $1 \to 0$, i.e., ψ is satisfied by $\texttt{FLIP}_{e'}(\boldsymbol{y}^{\text{NEW}})$. For case (b), we choose e' as a neighbor of the farthest end point (from e) in the odd path, i.e., $y_{e'}^{\text{NEW}} = 0$ (before flipping). Then, $y_{e'}^\star = 1$ since $\boldsymbol{y}^\star$ satisfies factor ψ_C and induces a union of disjoint even paths in the cycle C. Therefore, if we flip $y_{e'}^{\text{NEW}}$ from $0 \to 1$, we can still maintain the structure of disjoint even paths in $\boldsymbol{y}^{\text{NEW}}$ and ψ will be satisfied by $\texttt{FLIP}_{e'}(\boldsymbol{y}^{\text{NEW}})$. The proof for the case when ψ is satisfied by $\texttt{FLIP}_{e'}(\texttt{FLIP}_e(\boldsymbol{y}^\star))$ is similar. This completes the proof of Lemma B.2. $\square$

By construction $\boldsymbol{y}^{\text{NEW}}$ is a valid configuration that satisfies all the factor functions in $T_e(t)$. Hence, it suffices to prove that $w'(\boldsymbol{y}^{\text{NEW}}) > w'(\boldsymbol{y}^{\text{TMAP}})$, which would contradict the assumption that $\boldsymbol{y}^{MAP}$ is a MAP configuration on $T_e(t)$. To this end, for edge $(i,j) \in E'$, let $n_{ij}^{0\to 1}$ ($n_{ij}^{1\to 0}$) denote the number of Flip operations $0 \to 1$ ($1 \to 0$) for all copies of edge variable

(i,j) in the computation tree $T_e(t)$. Then, one derives

$$w'(y^{\text{NEW}}) = w'(y^{\text{TMAP}}) + \boldsymbol{w}' \cdot \boldsymbol{n}^{0\to1} - \boldsymbol{w}' \cdot \boldsymbol{n}^{1\to0},$$

where $\boldsymbol{n}^{0\to1} = \{n_{ij}^{0\to1}\}$ and $\boldsymbol{n}^{1\to0} = \{n_{ij}^{1\to0}\}$ are vectors containing counts of the variables flipped from $0 \to 1$ and $1 \to 0$, respectively.

Once gain, we note that the Flipping Procedure induces changes along a path P. We therefore consider two cases: (i) the path P does not arrive at a leave variable of $T_e(t)$, and (ii) otherwise. Note that case (i) is possible only when the condition in Step 3 (a) holds during the construction of $\boldsymbol{y}^{\text{NEW}}$.

Case (i). In this case, we define $y_{ij}^{\dagger} := y_{ij}^{\star} + \varepsilon(n_{ij}^{1\to0} - n_{ij}^{0\to1})$, and establish the following lemma:

Lemma B.3. *The vector $\boldsymbol{y}^{\dagger} = (y_{ij}^{\dagger})$ is feasible for match-aux-LP for small enough $\varepsilon > 0$.*

Proof. We have to show that $\boldsymbol{y}^{\dagger}$ satisfies (B.1) and (B.2). Here, we prove that $\boldsymbol{y}^{\dagger}$ satisfies (B.2) for small enough $\varepsilon > 0$. The proof that (B.1) is satisfied can be argued in a similar manner. For a given $C \in \mathcal{C}$, consider the following polytope $\mathcal{P}_C$:

$$\sum_{j \in V(C)} y_{i_C,j} \le |C| - 1, \quad y_{i_C,j} \in [0,1], \quad \forall j \in C, \quad \sum_{j \in V(C)} (-1)^{d_C(j,e)} y_{i_C,j} \in [0,2], \quad \forall e \in E(C).$$

We have to show that $\boldsymbol{y}_C^{\dagger} = (y_e\ : e \in \delta(i_C))$ is within the polytope. For the i-th copy of ψ_C in $P \cap T_e(t)$, we set $\boldsymbol{y}_C^{\star}(i) = \texttt{FLIP}_{e'}(\texttt{FLIP}_e(\boldsymbol{y}_C^{\star}))$ in Step (b), where $\boldsymbol{y}_C^{*}(i) \in \mathcal{P}_C$ as the new configuration is valid for ψ_C. Since the path P does not hit a leaf variable of $T_e(t)$, we have

$$\frac{1}{N} \sum\nolimits_{i=1}^{N} \boldsymbol{y}_C^{\star}(i) = \boldsymbol{y}_C^{\star} + \frac{1}{N} \left(\boldsymbol{n}_C^{1\to0} - \boldsymbol{n}_C^{0\to1} \right),$$

where N is the number of copies of ψ_C in $P \cap T_e(t)$. Furthermore, $\frac{1}{N}\sum_{i=1}^{N} \boldsymbol{y}^{\star}_C(i) \in \mathcal{P}_C$ because $\boldsymbol{y}^{\star}_C(i) \in \mathcal{P}_C$. Therefore, $\boldsymbol{y}^{\dagger}_C \in \mathcal{P}_C$ if $\varepsilon \leq 1/N$. This completes the proof of Lemma B.3. $\square$

The above lemma with $w'(\boldsymbol{y}^{\star}) > w'(\boldsymbol{y}^{\dagger})$ (due to the uniqueness of $\boldsymbol{y}^{\star}$) implies that $w' \cdot \boldsymbol{n}^{0\to1} > w' \cdot \boldsymbol{n}^{1\to0}$ and that $w'(\boldsymbol{y}^{\text{NEW}}) > w'(\boldsymbol{y}^{\text{TMAP}})$.

Case (ii). We consider the case when only one end of P hits a leave variable Y_e of $T_e(t)$, where the proof of the other case follows in a similar manner. In this case, we define $y^{\ddagger}_{ij} := y^{\star}_{ij} + \varepsilon(m^{1\to0}_{ij} - m^{0\to1}_{ij})$, where the vectors $\boldsymbol{m}^{1\to0} = (m^{1\to0}_{ij})$ and $\boldsymbol{m}^{0\to1} = (m^{0\to1}_{ij})$ are constructed as follows:

---

1. Initially, set $\boldsymbol{m}^{1\to0} \leftarrow \boldsymbol{n}^{1\to0}$ and $\boldsymbol{m}^{0\to1} \leftarrow \boldsymbol{n}^{0\to1}$.

2. If y^{NEW}_e is flipped from $1 \to 0$ and the parent of e is a cycle factor ψ_C for some $C \in \mathcal{C}$, then decrease $m^{1\to0}_e$ by 1 and

   2-1 If the parent $y^{\text{NEW}}_{e'}$ was flipped from $1 \to 0$, decrease $m^{1\to0}_{e'}$ by 1.

   2-2 Else if there exists a 'brother' edge $e'' \in \delta(i_C)$ of e such that $y^{\star}_{e''} = 1$ and ψ_C is satisfied by $\texttt{FLIP}_{e''}(\texttt{FLIP}_{e'}(\boldsymbol{y}^{\star}))$, then increase $m^{0\to1}_{e''}$ by 1.

   2-3 Otherwise, report ERROR.

3. If y^{NEW}_e is flipped from $1 \to 0$ and the parent of e is a vertex factor ψ_i for some $i \in V$, then decrease $m^{1\to0}_e$ by 1.

4. If y^{NEW}_e is flipped from $0 \to 1$ and the parent of e is a vertex factor ψ_i for some $i \in V$, then decrease $m^{0\to1}_e, m^{1\to0}_{e'}$ by 1, where $e' \in \delta(i)$ is the 'parent' edge of e, and

   4-1 If the parent $y^{\text{NEW}}_{e'}$ is associated to a cycle parent factor ψ_C, 4-1-1 If the grand-parent $y_{e''}^{\mathrm{NEW}}$ is flipped from $1 \to 0$, then decrease $m_{e''}^{1\to 0}$ by 1.

4-1-2 Else if there exists a 'brother' edge $e''' \in \delta(i_C)$ of e' such that $y_{e'''}^{*} = 1$ and ψ_C is satisfied by $\texttt{FLIP}_{e'''}(\texttt{FLIP}_{e''}(y^*))$, then increase $m_{e'''}^{0\to 1}$ by 1.

4-1-3 Otherwise, report ERROR.

4-2 Otherwise, do nothing.

We establish the following lemmas.

Lemma B.4. ERROR *is never reported in the above construction.*

Lemma B.5. $\boldsymbol{y}^{\ddagger}$ *is feasible to match-aux-LP for small enough* $\varepsilon > 0$*.*

Proofs of Lemma B.4 and Lemma B.5 are analogous to those of Lemma B.2 and Lemma B.3, respectively. From Lemma B.5, we have

$$c \;\le\; \frac{\boldsymbol{w'} \cdot (\boldsymbol{y}^{\star} - \boldsymbol{y}^{\ddagger})}{|\boldsymbol{y}^{\star} - \boldsymbol{y}^{\ddagger}|} \;\le\; \frac{\varepsilon\left(\boldsymbol{w'}(\boldsymbol{m}^{0\to 1} - \boldsymbol{m}^{1\to 0})\right)}{\varepsilon(t-3)} \;\le\; \frac{\varepsilon\left(\boldsymbol{w'}(\boldsymbol{n}^{0\to 1} - \boldsymbol{n}^{1\to 0}) + 3w'_{\max}\right)}{\varepsilon(t-3)},$$

where $|\boldsymbol{y}^{\star} - \boldsymbol{y}^{\ddagger}| \ge \varepsilon(t-3)$ follows from the fact that the path P hits a leaf variable of $T_e(t)$ and there are at most three increases or decreases in $\boldsymbol{m}^{0\to 1}$ and $\boldsymbol{m}^{1\to 0}$ in the above construction. Hence,

$$\boldsymbol{w'}(\boldsymbol{n}^{0\to 1} - \boldsymbol{n}^{1\to 0}) \ge c(t-3) - 3w'_{\max} > 0 \qquad \text{if} \quad t > \frac{3w'_{\max}}{c} + 3,$$

which implies $w'(y^{\mathrm{NEW}}) > w'(y^{\mathrm{TMAP}})$. If both ends of the path P hit leaf variables of $T_e(t)$, then we need $t > \frac{6w'_{\max}}{c} + 6$. This completes the proof of Lemma B.1.

Appendix C

Proofs Involving Region Selection Criteria and Cycle Bases

The following properties of Loop-SRGs are proven in [95]:

Theorem C.1. *A Loop-SRG has $\sum_{\gamma \in \mathcal{R}} c_\gamma = |L| - |E| + |V|$, where $|L|$ is the number of loop regions, $|E|$ the number of edge regions and $|V|$ the number of node regions.*

Theorem C.2. *A Loop-SRG is singular if $\sum_{\gamma \in \mathcal{R}} c_\gamma > 1$.*

Theorem C.3. *A Loop-SRG is singular iff there is a subset of loop regions and constituent edge regions such that all of the edge regions have 2 or more parents.*

The following proofs make use of these theorems as well as the reduction operators presented in [95].

Theorem 3.1: A Loop-SRG is Non-Singular and satisfies Counting Number Unity if its loop outer regions are a Fundamental Cycle Basis (FCB) of G.

Proof. $(FCB \Rightarrow \sum_{\gamma \in \mathcal{R}} c_\gamma = 1)$ From Theorem C.1, we see that $\rho = E - V + 1$ is exactly the number of loops needed to ensure that $\sum_{\gamma \in \mathcal{R}} c_\gamma = 1$. From this it follows that a loop SRG will satisfy counting number unity if the set of loop outer regions form a cycle basis of G.

($FCB \Rightarrow$ non-singularity) Assume that every state of $p(\boldsymbol{y})$ is equi-probable. Since all of the factors are uniform, we can remove them from all of the outer regions. Let $\mathcal{B}$ be a FCB of G and map each outer region R to one of the cycles in this basis. Since $\mathcal{B}$ is fundamental, there exists some ordering π such that cycle $C_{\pi(i)}$ has some edge that does not appear in any cycle preceding it. Let $R_{\pi(i)}$ be the loop outer region corresponding to cycle $C_{\pi(i)}$ and let $E_{\pi(i)}$ be the edge(s) unique to cycle $C_{\pi(i)}$. Let $R_{E_{\pi(i)}}$ be the edge region corresponding to edge $E_{\pi(i)}$. Since $E_{\pi(i)}$ is unique to $C_{\pi(i)}$, edge region $R_{E_{\pi(i)}}$'s only parent is $R_{\pi(i)}$. Thus, edge region $R_{E_{\pi(i)}}$ can be *Dropped*.

Let $\mathcal{C}(R_{\pi(i)})$ be the set of cliques of outer region $R_{\pi(i)}$. The clique corresponding to edge $E_{\pi(i)}$ can be *Shrunk* since child region $R_{E_{\pi(i)}}$ was dropped. Let $\bar{E}_{\pi(i)} = C_{\pi(i)} \setminus E_{\pi(i)}$ be the set of edges not unique to $C_{\pi(i)}$. The *Shrink* operation leaves the structure $\mathcal{G}(R_{\pi(i)})$ of region $R_{\pi(i)}$ as a chain over the edges $\bar{E}_{\pi(i)}$. This chain can be *Split* into its constituent edges by choosing the variables not in edge $E_{\pi(i)}$ as separators. The *Split* operation produces a set of edge outer regions $\mathcal{R}_{\bar{E}_{\pi(i)}}$ and node regions $\mathcal{R}_{\bar{V}_{\pi(i)}}$. These edge and node regions are duplicates of regions already in the SRG. And since all factors were initially removed, the regions in $\mathcal{R}_{\bar{E}_{\pi(i)}}$ and $\mathcal{R}_{\bar{V}_{\pi(i)}}$ can then be merged with the regions that they duplicate.

The loop outer regions can be *reduced* in this way along the ordering π - i.e. beginning with cycle $C_{\pi(\mu)}$ and ending with cycle $C_{\pi(i)}$. Reducing all loop regions yields an acyclic SRG (comprised of edge and node regions) which is non-singular from Theorem 5 in [95].

□

<u>Theorem 3.2</u>: A Loop-SRG is Tree Robust if its loop outer regions are a Tree Robust cycle

basis of G.

Proof. In proving Theorem 3.1 the idea was to show that a loop outer region can be *reduced* if it contains a *unique* edge; the fact that the loops form a FCB means that the loops can be reduced in an order such that each loop has a unique edge. However, reducing a loop region to its constituent edges requires *Merge* operations that can only be performed if the outer regions have no factors. This condition was guaranteed in the proof of Theorem 3.1 by initially removing all of the uniform factors from the SRG. In a Tree Robust SRG, only a subset of the factors are uniform and can be removed. Thus, we need a stronger tool. Using the *Factor Move* operator it is easy to show that:

Lemma C.4. *A loop outer region can be* <u>*reduced*</u> *if it contains at least one* <u>*unique*</u> *edge not covered by a factor.*

The desired result follows by incorporating this Lemma into the same sequence of reduction operators used in the proof of Theorem 3.1. □

To simplify notation, in the following proofs let $G_{\pi(i)} = \{C_{\pi(1)}, ..., C_{\pi(i)}\}$.

Lemma C.5. *A TR basis $\mathcal{B}$ is fundamental.*

Proof. (Proof by contrapositive): Assume that $\mathcal{B}$ is not fundamental. Then there exists no ordering π of the cycles in $\mathcal{B}$ such that $C_{\pi(i)} \setminus G_{\pi(i-1)} \neq \emptyset$ for $2 \leq i \leq \rho$. This implies that there is no ordering π for which $\left\{C_{\pi(i)} \setminus G_{\pi(i-1)}\right\} \setminus T \neq \emptyset$ for any spanning tree T. Thus, the basis is not TR. □

<u>Theorem 3.3</u>: Let $\mathfrak{B}^{|k|}$ denote all size k subsets of cycles in $\mathcal{B}$. A FCB $\mathcal{B}$ is Tree Robust iff $I(\mathcal{B}_k)$ is cyclic and not-empty for all $\mathcal{B}_k \in \mathfrak{B}^{|k|}$ for $1 \leq k \leq \rho$.

Proof. ($I(\mathcal{B}_k)$ is cyclic for all subsets of $\mathcal{B} \Rightarrow$ TR) First, we note that since the unique edge graph is cyclic for all subsets of cycles, the unique edge graph is cyclic for all partial orderings of the cycles as well.

Let $\mathcal{B}_{\pi(i)} = \mathcal{B} \setminus \{C_{\pi(i+1)}, ..., C_{\pi(\rho)}\}$ denote the set of cycles not appearing in the partial order $\pi(i+1), ..., \pi(\mu)$.

A basis is not TR if $\exists$ some j $(2 \leq j \leq \mu)$ such that $\{C \setminus G_{\pi(j-1)}\} \setminus T = \emptyset$ for all $C \in \mathcal{B}_{\pi(j-1)}$ for all orders $\pi \in \Pi$. We show that this cannot occur given that $I(\mathcal{B}_{\pi(j)})$ is cyclic for all $\pi \in \Pi$.

For $\{C \setminus G_{\pi(j-1)}\} \setminus T = \emptyset$ for all $C \in \mathcal{B}_{\pi(j-1)}$ and all orders, we require that either: 1) $C \setminus G_{\pi(j-1)} = \emptyset$; or 2) $C \setminus G_{\pi(j-1)}$ be acyclic. Since $I(\mathcal{B}_{\pi(j)})$ is not empty for all orderings, there must exist some $C \in \mathcal{B}_{\pi(j)}$ such that $C \setminus G_{\pi(j-1)} \neq \emptyset$. And since $I(\mathcal{B}_{\pi(j)})$ is cyclic for all orderings, there cannot exist some tree T that covers all edges in $I(\mathcal{B}_{\pi(j)})$.

(TR $\Rightarrow I(\mathcal{B}_k)$ is cyclic for all subsets of $\mathcal{B}$). Assume that $I(\mathcal{B}_k)$ is acyclic and consider some spanning tree T that 'covers' all of the edges in $I(\mathcal{B}_k)$ (i.e. $I(\mathcal{B}_k) \setminus T = \emptyset$). Clearly the basis $\mathcal{B}$ would not be tree exact w.r.t. to T and therefore not TR. We now must show that there exists some ordering such that $\mathcal{B}_{\pi(k)} = \mathcal{B}_k$. Assume that such an ordering does not exist. Then there must exist some $j > k$ for which $C \setminus G_{\pi(j)} = \emptyset$ for all $C \in \mathcal{B}_{\pi(j)}$. This would mean that the basis is not fundamental. However, from the previous Lemma we know that if $\mathcal{B}$ is not fundamental, it is not TR. $\square$

<u>Corollary 3.4</u>: An FCB is TR iff $\mathcal{B}_k$ is TR for all $\mathcal{B}_k \in \mathfrak{B}^{|k|}$ for $1 \leq k \leq \rho$.

Proof. ($\mathcal{B}$ is TR $\Rightarrow \mathcal{B}_k$ is TR for all k) Assume there is some $\mathcal{B}_k \subseteq \mathcal{B}$ that is not TR. Then there exists some $I(\mathcal{B}_k)$ that acyclic. We know that $\mathcal{B}$ is TR iff $I(\mathcal{B}_k)$ is cyclic for all subsets of $\mathcal{B}$. Therefore, $\mathcal{B}$ is not TR.

($\mathcal{B}_k$ is TR for all $k \Rightarrow \mathcal{B}$ is TR) Follows immediately from proof of Theorem 3.3. $\square$