Probabilistic Inference in Modern Genetic Linkage Analysis

Ming Su

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2013

Reading Committee:

Elizabeth A. Thompson, Chair

C. J. Richard Shi, Chair

Marina Meila-Predoviciu

Program Authorized to Offer Degree:
Electrical Engineering

University of Washington

Abstract

Probabilistic Inference in Modern Genetic Linkage Analysis

Ming Su

Co-Chairs of the Supervisory Committee:
Professor Elizabeth A. Thompson
Department of Statistics

Professor C. J. Richard Shi
Department of Electrical Engineering

Exact inference can be computationally intractable. When it is infeasible or impractical, one often resorts to approximate methods. This theisis focuses on the algorithms for approximate inference and their applications in modern genetic linkage analysis. The first part of the thesis concentrates on an application of inference algorithms in modern genetic linkage analysis. We develop a computationally efficient method for multipoint linkage analysis on extended pedigrees for trait models having a two-locus Quantitative Trait Locus (QTL) effect. The method is implemented in the program, *hg_lod*, which uses Markov Chain Monte Carlo (MCMC) method to sample realizations of descent patterns conditional on marker data, then calculates the trait likelihoods by an efficient exact probabilistic inference algorithm. *hg_lod* can handle data on large pedigrees with a lot of unobserved individuals, and can compute accurate estimates of lod scores at a much larger number of hypothesized locations than can any existing method. In the second part of this thesis, we focus on the theory of general probabilistic inference algorithms. We propose a convex relaxation method as an approximate inference algorithm. We give a convergence analysis of this method by deriving sufficient conditions for its convergence from a mathematical program-

ming point of view. We then show that this method is amenable to coarse-grained parallelization and propose techniques to parallelize it optimally without sacrificing convergence.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I am most grateful to my advisor, Professor Elizabeth Thompson, who has provided me with the opportunity to work on algorithm development for genetic linkage analysis. During my supervision under Professor Thompson, I am very much impressed by and gained from her highly motivated research style, scientific rigorousness and technical sharpness. As an advisor, Professor Thompson is highly responsible and she is always ready for discussion when I have new research ideas. Professor Thompson also taught me the background of genetic linkage analysis and supervised me closely on the project of two-locus linkage analysis, with a great degree of patience and encouragement. Again, I would really want to thank Professor Thompson, without whom this dissertation would not be completed at all.

I want to thank my committee members for attending my final exam and providing useful feedbacks. The committee includes: Professor Elizabeth Thompson, Professor CJ Richard Shi, Professor Marina Meila, Professor Jame Burke and Professor Jeng-Neng Hwang. I also want to thank Professor Maya Gupta, who attended my general exam and provided some helpful comments.

I would like to thank Professor Marina Meila who read the draft of my dissertation with great responsibility and carefulness, and gave very valuable technical comments.

I thank Professor Allen Wijsman for a lot of research discussion which helped me in shaping the project on two-locus genetic linkage analysis.

I want to thank Professor Yun Ju Sung from Washington University School of Medicine, for providing simulated data so I can obtain comparison results on the project of two-locus genetic linkage analysis.

I would like to thank Professor Rina Dechter from University of California at Irvine for providing me with the opportunity to work on the algorithm development for genetic linkage analysis. Professor Dechter also provided useful comments and software tools for comparison.

I would like to thank Professor Paul Tseng for teaching me in-depth optimization theory and providing helpful discussion on my work of the convex relaxation method.

I want to thank Yang Su from Institute of System and Mathematics at Chinese Academy of Science for providing useful comments on my proofs in the work of the convex relaxation method.

I would like to thank my peer students and postdocrate researchers in Professor Thompon's group. They have helped me by discussing my research ideas with me and/or providing simulated data and useful scripts. These includes but not limited to: Charles Cheung, Chris Glazner, Chaozi Zheng, Zheng Cai and Hoyt Koepke.

I would like to thank Professor Richard Shi for serving as my co-advisor in the Department of Electrical Engineering. Professor Shi offered me research assistantship at my early years at University of Washington.

I also want to thank Professor Bruce Weir from the Department of Biostatiscs for offerring me financial support during my last three years at UW so I can have this dissertation finished.

DEDICATION

To my parents

Chapter 1

INTRODUCTION

1.1 Overview

Human genetic linkage analysis concerns finding evidence of any gene affecting an observable trait linked to a set of known genetic markers, and estimating the location of this trait gene on the genome. It makes use of likelihood-based inference methods to analyze observed genetic/trait data of a number of related individuals. Modern days, the data involved in linkage analysis are often collected at a large number of genomic locations on large pedigrees, with possibly a high degree of missingness. In reality, a trait may be determined by multiple genes, and/or the interaction between genetic and environmental factors. However, the first objective of linkage analysis is to find the gene which contributes most to the trait.

Modern genetic data have put an increasing computational burden on linkage analysis. In this thesis, we develop novel computational methods in the framework of graphical models to facilitate efficient linkage analysis. In linkage analysis, the likelihood calculation is often cast to a standard inference problem on probablistic graphical models, either explicitly or implicitly constructed. Algorithms for exact computation of likelihoods have been developed by geneticists in early years of linkage analysis and can be shown to be equivalent to exact graphical model inference algorithms separately developed in computer science. As modern genetic data start imposing greater computational challenges, exact algorithms become infeasible or inpractical. This gives rise to Markov chain Monte Carlo (MCMC) approaches. In the first part of this thesis, we develop a new computationally efficient method for two-locus multipoint linkage analysis, based on the MCMC sampler developed by Tong

and Thompson [2008]. Meanwhile, Belief Propagation (BP) type of iterative approximate methods have seen dramatic success in a wide spectrum of applications. In the second part of this thesis, we study the convergence property of an iterative approximate method then develop a parallelization scheme to speed up its performance.

Section 1.2 introduces the linkage analysis and the formulation of lod score. In Section 1.3, we give the necessary genetic background for linkage analysis. The probabilistic model used in modern linkage analysis is introduce in Section 1.4. Then we present the algorithms for exact likelihood computation, and discuss their complexities and limitations in Section 1.5. Section 1.6 presents the approach to the estimation of lod scores using MCMC methods, with an emphasis on the marker data-based sample approach. Finally, Section 1.7 summarizes the contribution of this thesis.

1.2 Linkage Analysis

Linkage analysis produces a quantitative measure, the lod score [Morton, 1955], evaluated at each hypothesized location of the trait gene within the genome region of interet. First we assume a genetic model $\Gamma(\lambda)$ from which data are generated. Note that the model is indexed by the hypothesized location λ of the trait gene, and the null hypothesis $\Gamma(\infty)$ represents the case where the trait gene is outside the genome region under consideration, e.g., on a different chromosome. The likelihood of the model at each λ can be computed and contrasted against the null. Specifically, we evaluate lod scores by taking difference of the log-likelihoods between $\Gamma(\lambda)$ and $\Gamma(\infty)$:

$$\text{lod}(\lambda) = \log_{10} \frac{\Pr(Y; \Gamma(\lambda))}{\Pr(Y; \Gamma(\infty))}, \tag{1.1}$$

where $\Pr(Y; \Gamma)$ is the likelihood of the full model Γ and is also the probability of the observed data Y given the model. Note that conventionally, base 10 log is used in lod score instead of the natural log more often seen in log-likelihood difference [Morton, 1955].

The lod score has been the primary statistic for genetic linkage analysis since its introduction by Morton [1955]. The lod scores from an analysis are usually plotted horizontally along the genome structure for comparison purpose. Genomic locations with high lod scores (≥ 3.0) are suggested to have high chance of being the location of the trait gene while very low lod score ($\leq$ -2.0) will rule out the presence of the trait gene in the tested genome region. Note that these the thresholds are arbitrary and may be modified depending on the data used [Chen and Storey, 2006]. In general, the threshold should be raised when testing multiple genomic locations.

Observed data Y consists of marker data Y_M and trait data Y_T. Marker data Y_M are DNA measured on individuals using biological assays, at a large number of genomic locations which are previously known and used as reference points in locating the trait gene. Trait data Y_T are observable characteristics, which can be either discrete, e.g., blood type, or continuous, e.g., height. In reality, not all pedigree members can be observed, especially for individuals in early generations of the pedigree.

Assuming the genetic model is known *a priori*, the linkage problem boils down to evaluation of likelihoods. In traditional linkage analysis, the exact computation of likelihood is limited either by the size of the pedigree or the number of jointly analyzable genetic locations. As modern linkage analysis often deals with much larger datasets with potentially a lot of missing data, exact computation becomes impratical or infeasible. Although a fair amount of research effort has dramatically improved the efficiency of exact likelihood computation, the Markov-Chain Monte Carlo (MCMC) approach to estimating lod scores becomes the method of choice in modern linkage analysis, due to its computational efficiency and its support to complex trait models [Su and Thompson, 2012].

1.3 Pedigrees, Genetic Variants, and the Inheritance of Genome

1.3.1 Pedigree

A pedigree specifies the familial relationship of a set of related individuals. Specifically, a pedigree consists of a set of individuals and the specification of the two parents for each individual, with exception for founders, who are individuals whose parents are not given. All individuals who are not founders are called nonfounders. In order to represent a pedigree graphically, every individual in the pedigree is designated a unique identifier and a pedigree is often depicted graphically using a marriage node graph [Cannings et al., 1978]. The two individuals who form a couple and share at least one common offspring, are connected to a marriage node, and the marriage node is connected to each shared offspring of the couple.

We will use an example pedigree throughout in this chapter, which is shown in Figure 1.3, where the marriage nodes are shown as bullets and the identifiers are shown inside squares (males) or circles (females). The two nodes drawn upward to a marriage node are parent nodes while the downward ones are offspring. A parent individual may be connected to multiple marriage nodes. The pedigree in this example is relatively small (23 members) but it has some features that are common in large and complex pedigrees that arise in analysis of real data. Notice the presence of loops in the pedigree which can be caused by the marriage of related individuals (e.g., J and K), or other complex genealogical relationships, e.g., double-first cousins (C and D).

1.3.2 DNA and Genetic Variants

Each human cell nucleus contains two copies of double-stranded DNA material, i.e., the maternal genome and the paternal genome, with the former inherited from the DNA of the individual's mother, and the latter from the DNA of the individual's father. These two genomes are organized into 23 pairs of homologous chromosomes

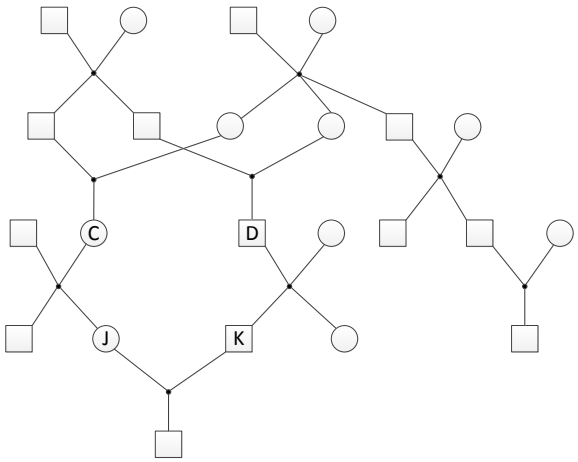


Figure 1.1: A 23-member pedigree.

(22 pairs of autosomes and 1 pair of sex chromosome). Each human being has two homologous copies of DNA. This is called diploidy and is not to be confused with the double-strandedness of the DNA. Organisms with a different number of homologous DNA copies in their cell nucleus are also found in nature, e.g., there are haploid organsim whose cell nuleus contains only one copy of DNA.

A chromosome is a structure of linearly ordered and concatenated DNA base pairs (bp). Each location on the chromosome is called a locus (loci in plural form). All possible DNA variants at a locus are called the alleles of this locus. An allele takes an allelic types with certain probability known as the allele frequency. The genotype at a single locus is defined as the pair of alleles that an individual carries at this locus. A haplotype is made of the alleles at multiple loci along one chromosome and the two homologous haplotypes make up a multilocus genotype. The multilocus genotype defined in such a way contains the information known as phase, that tells which alleles are on the same chromosome. Concatenating a set of single-locus genotypes will only give an unphased multilocus genotype.

Nowadays many locations on human genomes have been identified with a significant degree of DNA variation. The loci at these genomic locations are called genetic

markers, some of which may be functional genes while others may be only of structual purposes. The DNA contents at genetic markers can be effectively measured using biological technologies. The relative order among markers and their positions on the genome have been the goal of genetic map construction project, and are declared as known today. In linkage analysis, genetic markers are used as reference points to determine the genomic location of the trait locus. Observed genetic variants at the marker loci are unphased and are also known as phenotype, which could also mean the observed trait value.

Currently there are two types of DNA markers: microsatellite markers and Single Nucleotide Polymorphisms (SNP) markers. Microsatellite markers are spaced sparsely across the entire genome and usually only several hundred such markers will be genotyped in a genetic study. For a microsatellite marker, there might be many alleles. In contrast, a SNP marker has only two alleles but the number of SNP markers is much greater than that of microsatellite markers. Hence in practice SNP data usually convey same amount of genetic information as do microsatellite marker data. While microsatellite data are still being used in many of today's studies, the cost-effective SNP data have also achieved popularity, especially in linkage detection and localization of trait loci [Wilcox et al., 2005]. However, caution often needs to be used when dealing with very dense SNP markers, as accurate modeling of Linkage Disequilibrium (LD) effect is lacking in linkage programs for analysis with large number of loci [Wijsman et al., 2006].

1.3.3 Inheritance

An individual inherits DNA material from his or her parent through a biological process known as meiosis. The two genomes, namely, the paternal genome and the maternal genome, in an individual's cell nucleus are products of two meioses, one taking place in the individual's father, the other in the individual's mother. Each of the offsprint's two parental genomes is a blend of the DNA content of the maternal and

paternal genomes of the corresponding parent, i.e., at certain loci, the offspring's DNA is copied from the parent's maternal genome, and at other loci, the offspring's DNA is copied from the parent's paternal genome. This maintains the diversity in human biological characteristics. Mendel's First Law states that all meioses are independent, including those from a single parent to multiple offsprings and those occurred in two different parents. Mendel's Second Law states that the above phenomenon happens independently at each location on the genome, i.e., at each locus, the DNA material transmitted from parent to offspring is a randomly chosen one of the two parental DNA copies of the parent. However, inheritance is highly dependent among adjacent loci on the same chromosome. Specifically, DNA at nearby loci has a very high probability of being copied to an offspring from the same parental chromosome, and in fact chromosomes are inherited in chunks with length of order 10^8 bp. Finally, DNA on different chromosomes is believed to be inherited independently.

1.3.4 Identity By Descent

In a fixed set of related individuals, e.g., a pedigree, at each locus, all DNA segments of nonfounders are inherited from founders, i.e., they are all copies of founders' DNA segments. Two DNA segments that are copies of the same founder DNA segment are said to be *identical by descent (ibd)*. *ibd* DNA segments can be either of the same individual or different individuals. Note that *ibd* is always defined relative to the founders of a pedigree. By definition, the DNA segments of founders are not *ibd*. DNA are of same allelic types if they are *ibd* and of independent type if they are not.

At a locus, we can assign a unique identifier to each distinct founder genome. Such identifiers are called *founder genome labels* (FGL). Note that in a pedigree, at a particular locus, each DNA can be labeled using the FGL it is *ibd* with. The *ibd* relashionship among all DNA segments is a deterministic function of the descent pattern at this locus and can be illustrated by an FGL graph [Thompson, 2011] (also known as descent graph [Sobel and Lange, 1996]). An FGL graph can be depicted by

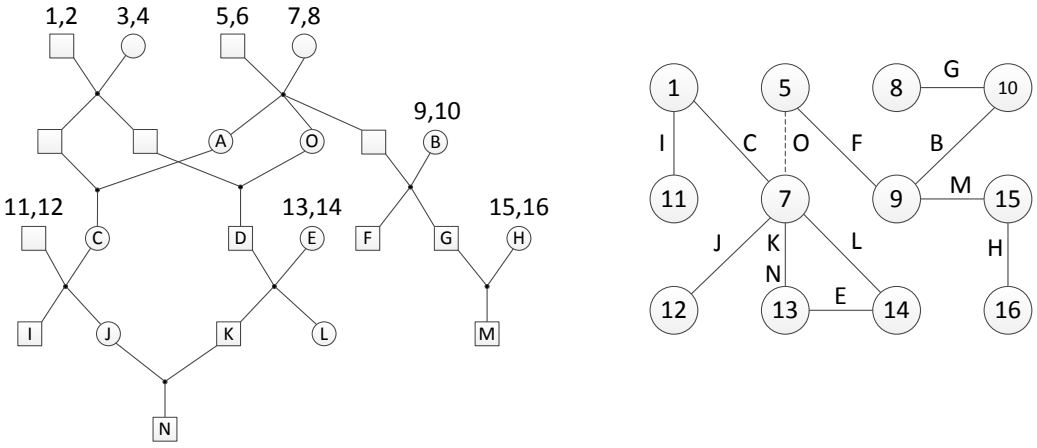


Figure 1.2: The FGL graph under a particular inheritance pattern.

allocating one node for each FGL, then for each pedigree member, drawing an edge between two FGL nodes representing two haploid DNA of this individual.

Figure 1.2 (left) shows the 23-member pedigree with paternal and maternal FGLs drawn besides founder nodes, separated by comma. The labeled nodes are observed pedigree members. The FGL graph given a particular inheritance pattern is shown at right in Figure 1.2. Notice that the observed status of one individual can change the structure of the FGL graph, e.g., if O is not observed, then the FGL graph will be separated into two disconnected components.

Note that the labeling of founder genomes is arbitrary, and two FGL graphs under different descent patterns may have same topology with respect to the edge labelings. We will refer to a node-unlabeled FGL graph as *ibd* graph and it is an important computational tool which will be discussed in Section 1.6. Figure 1.3 shows two FGL graphs under different inheritance patterns with the left one being the left component of the FGL graph in Figure 1.2. Both FGL graphs correspond to the same *ibd* graph. Given an inheritance pattern, each distinct *ibd* graph represents an equivalence class in the set of all FGL graphs. Examples given by Thompson [2011] show that across a set of inheritance patterns, the number of distinc *ibd* graphs can be much smaller

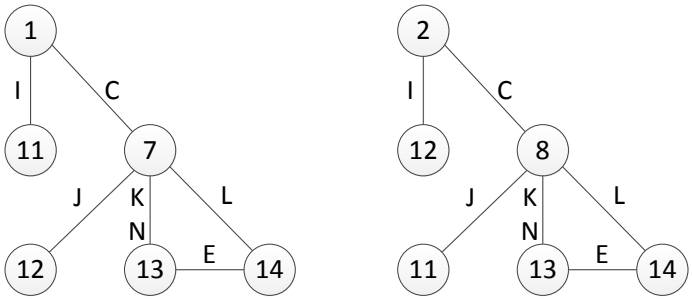


Figure 1.3: Equivalent FGL graphs under different inheritance patterns. FGL carried by individual C, I and J switch from {1,7} to {2,8}, {1,11} to {2,12} and {7,12} to {8,11}, respectively.

than the number of FGL graphs. For example, for a 47-member pedigree, out of 1000 realized FGL graphs, there are only 370 *ibd* equivalence classes.

1.3.5 Recombination and Genetic Map

Now we look at the biological process of meiosis in which an offspring inherits DNA from one of his or her parents. Prior to meiosis, in the parent's cell nucleus, each chromosome of a homologous pair duplicates to produce two identical sister chromatids connected to each other at the centremere. Two duplicated chromosomes then get tightly aligned, and may exchange DNA material at various loci. Then each of the duplicated chromosomes will divide, resulting in four gametes each of which (a sperm or an egg) containing a haploid copy of the genome.

Since DNA at neighbouring loci on a parental chromosome are very likely to be transmitted together to an offspring, each chromsome in the gamete cell is made of alternating segments of DNA different parental sources, separated by crossovers, which are locations at which the parental source switches. A recombination event is said to occur between two loci if the DNA at these two loci are transmitted from different parental chromsomes of the offspring's parent. Note that a recombination event occurs when there is an odd number of crossovers between these two loci. The

probability of this event is the recombination fraction ρ between two loci. Under a given meiosis model, the recombination fraction between two loci is given by the genetic map function $\rho(d)$, where the genetic distance d is defined as the expected number of crossovers between two loci. Note that since the genetic distance is defined as an expectation, it is additive thus provides a linear scale along the chromosome. For example, given three loci i, j and k, and the genetic distance between i and j, and that between j and k, the genetic distance between i and k is a sum of the former two. Even though two loci separated far in physical distance, measured in number of base pairs, will also have a long genetic distance between them, the genetic distance is not necessarily proportional to the physical distance. The unit for genetic distance is Morgan and centiMorgan (0.01 Morgan), which is about 10^6 bp on average.

When deriving the map function, a simple but sound assumption which is frequently made is the absence of genetic interference [Haldane, 1919]. Under this assumption, the occurrence of one crossover at a locus does not affect the likelihood of another at nearby loci, thus the occurence of crossover along the chromosome is simply a Poisson process at rate 1 (per Morgan) and the number of crossovers $W(d)$ has a Poisson distribution with mean d. It's straighforward to see that there is a recombination event between two loci if $W(d)$ is odd. Then the recombination fraction can be computed

$$\rho(d) = \sum_{k \text{ odd}} \frac{d^k e^{-d}}{k!} = \frac{1}{2}(1 - e^{-2d}). \tag{1.2}$$

This is the Haldane map function [Haldane, 1919]. A different map function might be derived under a different assumption for the meiosis process. Given the way the genetic distance is defined, any map model arranges loci in an linear order along the chromosome.

1.4 Model for Linkage Analysis

In this section, we describe the genetic model from which the observed data are assumed to be generated. The model $\xi = (\Gamma_M, \beta, \lambda)$ consists of the marker model Γ_M, the trait model β and the hypothesized location λ of the trait locus.

The marker model Γ_M can be further broken down to the population model and the transmission model. The population model speficies for each marker locus the allele frequencies while the transmission model specifies the recombination fraction between any two marker loci. Actually, the marker model are often given as a set of ordered marker loci and the genetic distances between them. The recombination fractions between marker loci as well as that between marker loci and the hypothesized location λ can be derived using an assumed map function.

The trait model consists of population allele frequencies for the trait locus and the trait penetrance which specifies the probability of an individual's phenotype conditional on the genoypte he or she carries. The penetrance probability may depend on other covariates, such as the individual's age, sex, and/or other environmental effects. Under the assumption that there is only one major gene affecting the trait in the genome region under consideration, the penetrance is often given as a table of probabilities with one entry for each phenotype-genotype configuration.

1.5 Exact Likelihoood Computation

With the full genetic model defined, the equation for lod score (Eq.1.1) can be rewritten

$$\mathrm{lod}(\lambda) = \log_{10} \frac{\Pr(Y_T, Y_M; \Gamma_M, \lambda, \beta)}{\Pr(Y_T, Y_M; \Gamma_M, \lambda = \infty, \beta)}. \tag{1.3}$$

For each hypothesized location λ of the trait locus, a lod score can be achieved by evaluating the two likelihoods in Eq.1.3. The null hypothesis in the denominator assumes that the trait locus is unlinked to any of the marker loci, implying that Y_M

does not provide any informatoin about the actual genotype at the trait locus, so the probability of Y_T is marginally determined

$$\Pr(Y_T, Y_M; \Gamma_M, \lambda = \infty, \beta) = \Pr(Y_M; \Gamma_M)\Pr(Y_T; \beta). \tag{1.4}$$

We want to evaluate the lod score for a set of values of β and for each value of β, the two likelihoods for the null hypothesis need to be computed only once while the nominator of Eq.1.3 needs to be calculated for each hypothesized λ.

The three likehoods to be computed in obtaining a lod score essentially bear the same latent structure, only differing in the number of loci involved and the type of phenotypic data. So we will focus on the computation of a general quantity $\Pr(Y)$, where $Y = \{Y_{i,j} | i = 1, \cdots, m, j = 1, \cdots, l\}$ and $Y_{i,j}$ is either an observed genotype or a trait value of individual i at locus j.

Choosing approporiate latent space is a key in exact likelihood computation because the computational efficiency of the exact computation highly depends on the latent structure underlying the observed data. In the following, we will introduce two types of latent variables each leading to a traditional algorithm for exact likelihood computation. We will derive the computational complexities of these algorithms and discuss their limitations. Finally, based on the concept of graphical models, we show that both of these algorithms belong to a broader class from which more efficient algorithms and their computer implementations are developed to facilitate the exact computation of likelihoods.

1.5.1 The Elston-Stewart and Related Algorithms

Based on an intuitive choice of latent variables, the unobserved phased multilocus genotype G, we have

$$\begin{aligned} \Pr(Y) &= \sum_G \Pr(Y|G)\Pr(G) \\ &= \sum_G \prod_{i \text{ obs}} \Pr(Y_i|G_i) \prod_{i \text{ fou}} \Pr(G_i) \prod_{i \text{ nonfou}} \Pr(G_i|G_i^p, G_i^m), \end{aligned} \tag{1.5}$$

where G_i, G_i^p and G_i^m are genotypes of individual i, father and mother of individual i, respectively. In Eq.1.5, $\Pr(G_i)$ is directly provided by the marker population allele frequencies and $\Pr(G_i|G_i^p, G_i^m)$ is specified by the transmission model. When Y_i is observed genotype, $\Pr(Y_i|G_i)$ is simply 1 or 0 depending on whether Y_i is compatible with the latent G_i. When Y_i is a trait value, $\Pr(Y_i|G_i)$ are determined by penetrance parameters.

If performed directly, the summation in Eq.1.5 is exponential in the size of the pedigree. However, notice that Eq.1.5 implies a conditional independence structure where conditional on the genotypes of the parents, the probability of an individual's genotype is independent of genotypes of others. This allows the summation to be carried out sequentially. This method was first proposed by Elston and Stewart [1971], where one nulcear family is processed during each step of the sequential summation. Their method was later implemented in LIPED, the first widely available computer software for exact likelihood calculation [Ott, 1976], which played an important role in gene-mapping studies. Cannings et al. [1978, 1980] generalized the method to arbitrary pedigrees with more complex models and started to use the term pedigree peeling to refer the sequential approach. For complex pedigrees where loops might be present, multiple nuclear families might need to considered jointly.

Now we discuss the computational complexity of the pedigree peeling algorithm. If locus j has k_j alleles, there are in total $N = \prod_{j=1}^{l} k_j$ possible haplotypes, and $\frac{1}{2}N(N+1)$ possible phased genotypes for an individual. For a loop-free pedigree, we need to enumerate all possible genotype combinations of members in a nuclear family (two parents and one offspring), resulting in a complexity of order N^6, and hence exponential in the number of loci. For complex pedigrees, intermediate probability functions involving a greater number of individuals might need to be constructed, resulting in a complexity of an order higher than N^6. In general, the peeling algorithm is linear in pedigree size, but exponential both in the number of loci and pedigree complexity.

Many sophisticated computer implementations of the Elston-Stewart algorithm have been developed over years. The LINKAGE package [Lathrop et al., 1984, 1985] allows multiple marker loci to be analyzed jointly. It can handle highly polymorphic microsatellite data at multiple loci in relatively large pedigrees and have enabled successful localization of genes for many Mendelian disorders. Cottingham et al. [1993] used improved software engineering techniques, such as caching and replacement of floating point with integer operations, to speed up LINKAGE by about one order of magnitude. In the implementation of VITESSE [OConnell and Weeks, 1995], it was shown that in certain cases merging of allele in a single meta-allele is possible and that this could dramatically reduce the number of distinct genotypes where speed-up can be achieved.

1.5.2 The Lander-Green and Related Algorithms

The pedigree peeling algorithm imposes limitation on the number of loci that can be analyzed jointly. State-of-the-art implementations of the Elston-Stewart algorithm in the VITESSE and FASTLINK [Cottingham et al., 1993] computer packages can handle no more than 5-10 marker loci at a time. However, when data at more markers are available, the multipoint analysis, i.e., the joint analysis of data at multiple loci, is more powerful than single-locus analysis [Lander and Green, 1987; Lathrop et al., 1984, 1985].

Lander and Green [1987] develped an alternative method for multipoint analysis of data on a large number of loci. Their method is based on the use of inheritance vectors [Lander and Green, 1987] $S_{\cdot,j} = \{S_{i,j}, i = 1, \ldots, m\}$, $j = 1, \cdots, n$, where $S_{i,j}$ specifies for meiosis i at locus j, the offspring's received DNA is

$$\begin{cases} \text{parent's maternal DNA} & \text{if } S_{i,j} = 0, \\ \text{parent's paternal DNA} & \text{if } S_{i,j} = 1. \end{cases}$$

$S_{\cdot,j}$ specifies for all nonfounder members of the pedigree, the descent of DNA at locus j, while $S_{i,\cdot} = \{S_{i,j}, j = 1, \ldots, l\}$ specifies the descent of DNA at all loci for meiosis i.

In Section 1.3.5, we have mentioned that with the absence of genetic interference, crossover occurs as a Poisson process, so numbers of recombination events in disjoint chromosomal intervals are independent. Under this assumption, the inheritance vectors $S_{\cdot,j}$ are first-order Markov along the chromosome and we have

$$\begin{aligned}\Pr(Y) &= \sum_S \Pr(Y|S)\Pr(S) \\ &= \sum_S (\prod_{j=1}^{l} \Pr(Y_{\cdot,j}|S_{\cdot,j}))\Pr(S_{\cdot,1}(\prod_{j=2}^{l} \Pr(S_{\cdot,j}|S_{\cdot,j-1})),\end{aligned} \tag{1.6}$$

The probabilities $\Pr(S_{i,j} = 1)$ and $\Pr(S_{\cdot,j}|S_{\cdot,j-1})$ are given by the transmission model. For meiosis i, a recombination event occurs between two loci, j and j', if $S_{i,j} \neq S_{i,j'}$. The probability of this event is $\Pr(S_{i,j} \neq S_{i,j'}) = \rho(d)$, where $\rho(\cdot)$ is the map function and d is the genetic distance between j and j'. From Mendel's First Law we know that the vectors $S_{i,\cdot}$, $i = 1, \cdots, m$ are independent, so we have

$$\Pr(S_{\cdot,j}|S_{\cdot,j-1}) = \rho_{j-1}^{R_{j-1}}(1 - \rho_{j-1})^{m-R_{j-1}}, \tag{1.7}$$

where $R_{j-1} = (\sharp i : S_{i,j} \neq S_{i,j-1})$. Also marginally $\Pr(S_{i,j} = 0) = \Pr(S_{i,j} = 1) = 1/2$.

Intuitively, $\Pr(Y_{\cdot,j}|S_{\cdot,j})$ can be evaluated by summing over all configurations of $G_{\cdot,j}$ that are compatible with $S_{\cdot,j}$. Since $S_{\cdot,j}$ specifies the descent of DNA from founders to nonfounders, each allelic assignment to founder genotypes will produce a compatible configuration of $G_{\cdot,j}$. Thus $\Pr(Y_{\cdot,j}|S_{\cdot,j})$ can be computed by summing over all allelic assignments to the founder genomes at locus j. This can be done using a form of peeling on FGL graphs introduced in Section 1.3.4.

Notice that Eq.1.6 is in the form of a hidden Markov model (HMM), where the transmission model governs the hidden layer of $S_{\cdot,j}$ and the population model and the penetrance model determine the probability of $\Pr(Y_{\cdot,j}|S_{\cdot,j})$. Under this model, a standard HMM algorithm, such as Baum's [Baum et al., 1970] can be applied. Since

there are m meioses, each $S_{\cdot,j}$ can take 2^m values, so the computational complexity of Lander-Green method is of order $2^m \times 2^m \times l = l \times 4^m$ [Lander and Green, 1987], which is exponential in the pedigree size and linear in the number of loci. Given its complexity, Lander-Green algorithm has been the choice of analyses on data for a large number of markers collected on many relatively small pedigrees.

Note is that the complexity analysis above only involves the updating of $S_{\cdot,j}$, while the complexity of computing $\Pr(Y_{\cdot,j}|S_{\cdot,j})$ is not included. Practically, FGL graphs often arise as disjoint and simple components, whose computation does not affect the overall complexity of Lander-Green algorithm.

Popular implementations of the Lander-Green algorithm include the computer packages GENEHUNTER [Kruglyak et al., 1996; Markianos et al., 2001], ALLEGRO [Gudbjartsson et al., 2000], and MERLIN [Abecasis et al., 2002]. All these packages can handle very large numbers of markers and allow the estimation of individual haplotypes or the analysis of quantitative and discrete traits, providing parametric and nonparametric linkage tests.

1.5.3 A Graphical Model-Based Unifying Framework

In this Section, we present a framework that provides an unifying view on conditional independence structures arising in linkage problems, from which more efficient algorithms are developed to facilitate exact likelihood computation.

While Elston-Stewart algorithm and Lander-Green algorithm were developed by geneticists for linkage analysis, the underlying computation, i.e., marginalization, is a prevailing theme in computer science, statistics and many other fields. The graphical representation of the dependence structure among a set of random variables is called a graphical model, a powerful tool for many computations in probabilistic inference, including marginalization. The sequential summation approach for graphical models is known as Variable Elimination (VE) which has received numerous research efforts since 1990 [Dechter, 1998].

While any order can be assumed in the marginalization process, some do better than others. Finding the optimal elimination order in which the shortest computation time can be achieved, is NP-hard in itself [Arnborg, 1985; Arnborg et al., 1987]. While in some applications an intuitive elimination order may exist, people often resort to heuristics to explore sub-optimal orders.

Recall that the latent variables used in Elston-Stewart algorithm and Lander-Green algorithm are genotypes and inheritance vectors, respectively. Now we consider a model for linkage data with both genotype and inheritance vector variables. We show that with the freedom in this full model, more flexible and problem-specific elimiation orders can be developed, resulting in performance improvement over Elston-Stewart algorithm and Lander-Green algorithm. First we introduce the Bayesian network modeling of the linkage data. A Bayesian network (BN) is a directed graphical model with nodes representing the set of random variables and edges their conditional independencies [Lauritzen and Spiegelhalter, 1988]. Conditional on the parents of a node, i.e., the nodes having outgoing edges to this node, this node is independent of all others. Figure 1.4 shows the BN for a nuclear family at 3-loci.

Based on the BN model introduced above, Fishelson and Geiger have developed VE algorithm for linkage analysis which outperforms leading linkage software with regards to speed and memory requirement [Fishelson and Geiger, 2002]. Their method allows for arbitrary elimination order and indeed consists of routines to find sub-optimal summation sequence at runtime for the remaining portion of the problem. Whenever memory usage is approaching a preset limit, the algorithm switches to a type of computation, known as conditioning, to reduce the memory requirement. In conditioning, rather than summing over all variables, the assignment to a subset of variables are held fixed and rest variables are summed over, then the procedure repeats for the next assignment to the conditional variables. Performance improvements can also be attributed to several implementation features, including preprocessing steps to reduce time and memory requirement, compact representation of multilocus

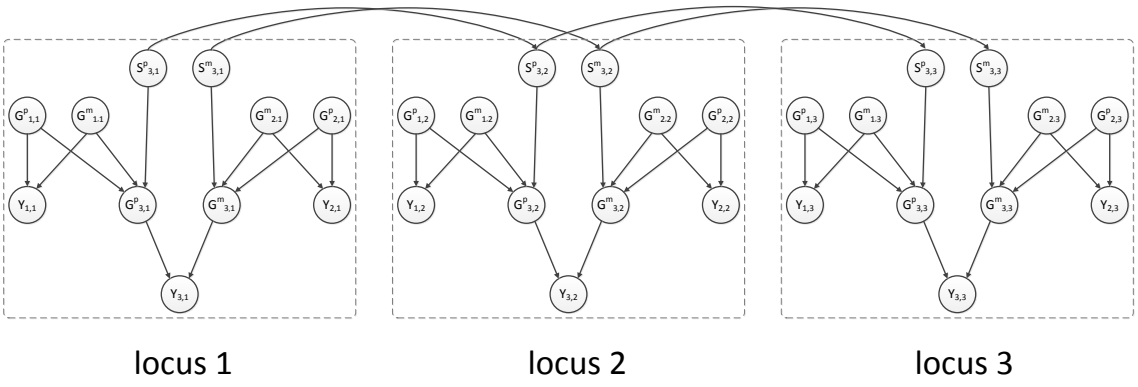


Figure 1.4: A bayesian network for a nuclear family at 3-loci. Nodes $G^p_{i,j}$ and $G^m_{i,j}$ represent the paternal and maternal haplotype of individual i at locus j, respectively. Nodes $S^p_{i,j}$ and $S^m_{i,j}$ represent the indicator for the meiosis to individual i's paternal and maternal haplotype at locus j, respectively

genotypes and conditional probability tables, etc. Silberstein et al. [2006] further developed Superlink-Online which makes the use of parallel computing technology. In Superlink-Online many phases of their algorithm are parallelized. The most important idea behind the online Superlink algorithm is that it performs the computation tasks associated with assignments to the conditioning variables in parallel. A similar approach developed by Allen and Darwiche [2008] uses Recursive Conditioning to achieves orders of magnitude faster performance for certain linkage instances.

1.6 MCMC Estimation of Lod Scores

We have presented the algorithms for exact likelihood computation and their popular computer implementations. In this section, we discuss the rise of Markov Chain Monte Carlo (MCMC), as an approximate method, in meeting the modeling and computational requirements arising from today's linkage studies. We will first discuss the motivation of using MCMC sampling methods in linkage analysis. Then we show how estimation of multipoint lod scores can be achieved using MCMC realizations of inheritance vectors, with an emphasis on the sampling approach based only on marker

data.

1.6.1 Motivation

As both the number of markers used in joint analysis and the size of the pedigrees increase a lot in nowaday's analyses, exact computation is faced with greater computational challenges than ever. The computational burden is further amplified by more complicated trait models which are used to address the current studies of complex diseases. Approximate methods are often used to obtain an estimate of the lod score when exact comutation becomes infeasible. MCMC sampling methods in genetic linkage analysis have received a lot of research effort, and some of the current publicly avaible MCMC programs, e.g., SimWalk [Lange and Sobel, 1991; Sobel et al., 2001] and MORGAN [Thompson, 2000; Thompson et al., 1993] have successfully enlarged the scale of genetic data that can be practically analyzed.

As opposed to Lander-Green algorithm, where all configurations of the latent inheritance pattern are exhaustively enumerated, an MCMC method first samples MCMC realizations of the inheritance patter, e.g., inheritance vectors at all marker loci, and subsequently computes Monte Carlo (MC) estimation of multipoint lod scores by averaging over such realizations. These MCMC realizations of latent variables are usually extracted from a dense set of MCMC scans when the Markov chain enters the steady state. Note that in theory, the longer the MCMC is run, the more accurate is the obtained estimation, which makes MCMC method an anytime algorithm. In practice, the number of MCMC scans is often set based on a number of factors, the scale and the quality of the data, the mixing property of the method and sometimes experience. The conditional independence structure in linkage analysis has provided a variety of MCMC updating rules, ranging from the single-site genotypic updating samplers of [Sheehan, 2000] to the most educated multiple-meiosis and locus sampler developed by Tong and Thompson [2008].

In MCMC methods, inheritance patterns can be sampled conditional either on

marker data only [Lange and Sobel, 1991] or both marker and trait data [George and Thompson, 2003]. While a well-approximated trait model will be more likely to produce accurate results, MCMC sampling of latent inheritance based only marker data has become the most promising approach in today's analysis of complex diseases. There are several reasons for this. First, the sampling of inheritance patterns cannot benefit from nowaday's traits under study which are too complex to be modeled sufficiently accurately. Second, very dense markers become available and they jointly provide very strong information about the inheritance pattern. Third, with the exclusion of trait data, MCMC sampling can be performed more straightforwardly. Fourth, since the inheritance patterns are obtained independently of trait data, analyses with different trait models can be performed without resampling of the realizations. This feature also facilitates sensitivity analysis of trait model parameters [Su and Thompson, 2012].

1.6.2 MCMC estimation of lod score

Now we present the estimation of multipoint lod scores based on the MCMC method that samples the inhertitance vectors S_M conditional on marker data Y_M. This approach was first proposed by Lange and Sobel [1991] and there have been several marker-based MCMC sampling approaches developed over the years [Sobel and Lange, 1996; Thompson, 2000, 2005].

Note that given the decomposition in Eq.1.4, Eq.1.3 can be rewritten as

$$\begin{aligned} \mathrm{lod}(\lambda) &= \log_{10}\frac{\Pr(Y;\Gamma_M,\lambda,\beta)}{\Pr(Y_M;\Gamma_M)\Pr(Y_T;\beta)} && (1.8)\\ &= \log_{10}\frac{\Pr(Y_T|Y_M;\Gamma_M,\lambda,\beta)}{\Pr(Y_T;\beta)}. && (1.9) \end{aligned}$$

The conditioning of Y_T on Y_M can be established via S_M, the inheritance vectors at marker loci,

$$\Pr(Y_T|Y_M;\Gamma_M,\lambda,\beta) = \sum_{S_M}\Pr(Y_T|S_M;\lambda,\beta)\Pr(S_M|Y_M;\Gamma_M). \qquad (1.10)$$

The MCMC estimation based on Eq.1.10 is thus

$$\tilde{\Pr}(Y_T|Y_M;\Gamma_M,\lambda,\beta)=\frac{1}{N}\sum_{i=1}^{N}\Pr(Y_T|S_M^i;\lambda,\beta), \tag{1.11}$$

where S_M^i is the ith MCMC realization of S_M and there are a total number of N such realizations used in the estimation. For each realization of S_M, $\Pr(Y_T|S_M;\beta,\lambda)$ can be achieved by peeling the FGL graph which is a deterministic function of S_M. Note that this gives an example of where exact computation can be utilized in an MCMC method. With today's dense marker data, the marker loci are the only locations which need to be considered as hypothesized trait locus. Thus lod scores are only estimated at marker loci, at which inheritance patterns, thus *ibd* graphs, are directly available from the MCMC sampling.

The relative ease in computing $\Pr(Y_T|S_M)$ also gives rise to MCMC method. Peeling on the FGL graph given by a particular realization of inheritance vectors is limited neither by the pedigree size nor by the number of linked loci. The computing time depends on the complexity of the FGL graph , i.e., the number of FGL, whose assignment needs to be considered jointly. Howeve, usually the majority of the FGL graphs can be partitioned into components which can be efficiently peeled.

Distinct ibd graphs provides a set of equivalence classes for FGL graphs and FGL graphs within same class are equivalent in likelihood computation. This results in a significant reduction in computing $\Pr(Y_T|Y_M)$ because we only need to evaluate $\Pr(Y_T|S_\lambda)$ for each equivalence class then multiply by the count of FGL graphs in each set. Inheritance patterns have a high chance to remain constant or only slightly differ at adjacent loci , so do *ibd* graphs. Koepke et al. [2011] developed a C library named **IBDgraph** that takes inheritance vectors as input and produces a set of equivalence classes, i.e., distinct *ibd* graphs. **IBDgraph** has the capability to construct *ibd* graphs either at a single-locus or across a number of loci. The latter can be used to achieve further computational reduction. **IBDgraph** is available for download at

http://www.stat.washington.edu/thompson/Genepi/genepi.shtml.

1.7 Contribution of This Thesis

In this thesis, we explore the direction of approximate inference methods for linkage calculation. Our work can be divided into two parts: MC-based approach and iterative BP-variants.

In Chapter 2, we develop a computationally efficient method for multipoint linkage analysis on extended pedigrees for trait models having a two-locus QTL effect. Given its computational efficiency, this method can handle data on large pedigrees with unobserved individuals, and can compute accurate estimates of lod scores at a much larger number of hypothesized locations than can any existing method. When comparing to a state-of-art linkage program for trait models with two major loci, using simulated data, results show that our implementation is orders of magnitude faster while the accuracy of QTL localization is retained. The efficiency of our method also facilitates analyses with multiple trait models, e.g. sensitivity analysis. This work has been published in Su and Thompson [2012]

In Chapter 3, we review the background of BP-like iterative algorithms, in a region-based framework. In Chapter 4, we propose a message passing algorithm, called convex relaxation method, as a natural generalization of Unified Propagation and Scaling (UPS) algorithm. We give a convergence analysis of the convex relaxation method. We derive sufficient conditions for the method's convergence from a mathematical programming point of view. This work has been published in Su [2010]. In Chapter 5, we show that this method is amenable to coarse-grained parallelization and propose techniques to parallelize it optimally without sacrificing convergence. Experiments on a shared memory system demonstrate that near-ideal speedup is achieved with reasonable scalability. This work has been published in Su [May 2011].

Chapter 2

COMPUTATIONALLY EFFICIENT MULTIPOINT LINKAGE ANALYSIS ON EXTENDED PEDIGREES FOR TRAIT MODELS WITH TWO CONTRIBUTING MAJOR LOCI

We have developed a computationally efficient method for multipoint linkage analysis on extended pedigrees for trait models having a two-locus QTL effect. The method has been implemented in the program, *hg_lod*, which uses MCMC method to sample realizations of descent patterns conditional on marker data, then calculates the trait likelihood for each realization by efficient exact computation on graphical models. Given its computational efficiency, *hg_lod* can handle data on large pedigrees with a lot of unobserved individuals, and can compute accurate estimates of lod scores at a much larger number of hypothesized locations than can any existing method. We have compared *hg_lod* to *lm_twoqtl*, the first publically available linkage program for trait models with two major loci, using simulated data. Results show that our method is orders of magnitude faster while the accuracy of QTL localization is retained. The efficiency of our method also facilitates analyses with multiple trait models, e.g. sensitivity analysis.

2.1 Introduction

In genetic linkage analysis, lod scores can be calculated for pedigrees with trait and marker data on observed members. The size of the pedigree becomes a choice in the study design of such analyses. In general, large and complex pedigrees are more informative than small and simple pedigrees in terms of power to detect linkage, accu-

racy of gene localization and accuracy of parameter estimates, especially for complex traits [Wijsman and Amos, 1997]. In addition, multipoint analysis has higher power than does single markers analysis [Wijsman and Amos, 1997]. For multipoint analysis on extended pedigrees with data on a large number of markers, lod scores are typically estimated by sampling-based methods, because exact methods, such as the Elston-Stewart algorithm [Elston and Stewart, 1971] and the Lander-Green algorithm [Lander and Green, 1987], become infeasible, due to their limitation on either the size of the pedigree or the number of markers that can be analyzed jointly. Lange and Sobel [1991] developed a Monte Carlo approach to the estimation of lod scores on extended pedigrees. In this approach, Markov Chain-Monte Carlo (MCMC) is used to sample realizations of multi-locus descent states conditional on marker data, then the trait likelihood contribution for each realization is computed by pedigree peeling. More recently, marker based realizations of descent patterns across a chromosome have been generated for subsequent use in trait analyses [Thompson, 2011]. In this chapter, we show how these realizations may be used not only for single-locus trait models, but also in a new computationally efficient method to estimate lod scores on extended pedigrees for trait models with two contributing major loci. Previously, only sampling-based approximate methods existed for trait likelihood computation in analyses with two loci [Sung and Wijsman, 2007; Sung et al., 2007b]. We propose an efficient way of computing trait likelihoods which increases both the computational efficiency and the accuracy of lod score estimation.

2.1.1 Background of Linkage Analysis with Two-Locus Trait Model

Multipoint linkage analysis for trait models with more than one major locus is important because it is generally hypothesized that complex traits are jointly affected by multiple genomic loci, potentially on the same chromosome. It has been shown that both parametric and nonparametric analyses with two-major-locus trait model can yield higher power for linkage detection and more accurate localization of trait loci

than analyses with single-locus models, for both discrete [Knapp et al., 1994; Schork et al., 1993; Strauch et al., 2003] and quantitative traits [Sung et al., 2007a].

Several programs and methods have been developed to facilitate linkage analyses using two-locus model. However these approaches all have their own limitations. The program TMLINK [Lathrop and Ott, 1990] can perform parametric multipoint linkage analysis for only a small number of loci using a trait model with two unlinked loci. The program GENEHUNTER-TWOLOCUS [Strauch et al., 2000] can perform both parametric and nonparametric analyses using a trait model with two loci, either linked or unlinked. Biswas et al. [2003] used a Bayesian parametric approach to detecting linkage with possible locus heterogeneity. These approaches are limited either in the number of markers in the multipoint analysis or in the size of the pedigree. Biernacka et al. [2005] used IBD sharing in affected sib pairs to detect linkage of two linked trait loci. Lin and Schaid [2007] generalized this approach to affected sib pairs to affected relative pairs. These IBD sharing approaches often require a large number of affected relative pairs which are often not available in genetic studies.

The MCMC-based MORGAN program *lm_twoqtl* was the first publically available program for lod scores with a parametric model including two Quantitative Trait Loci (QTL) and a polygenic component. This is a more complex trait model than can be analyzed with previously available programs [Sung et al., 2007b]. The program uses MCMC method so it can analyze extended pedigrees with a large number of marker loci. Sung et al. [2007b] showed that in an analysis using a model with known parameters, *lm_twoqtl* yielded higher power in linkage detection and more accurate localization than the Variance Components (VC) approach in SOLAR [Almasy and Blangero, 1998]. However, for large pedigrees with a lot of missing data, it becomes very computationally costly to achieve accurate estimates of lod scores using *lm_twoqtl*.

2.1.2 Proposed Method

We propose a method that is more computationally efficient than *lm_twoqtl*. Rather than using the importance sampling approach in *lm_twoqtl*, we compute exact trait likelihoods given inheritance vectors using an efficient graphical model algorithm. In addition, to sample the MCMC realizations of inheritance vectors conditional on marker data, we use the multiple-meiosis sampler [Tong and Thompson, 2008] whereas a single-meiosis sampler was used in the early implementation of *lm_twoqtl* [Sung et al., 2007b]. For large pedigrees with a lot of missing data, the multiple-meiosis sampler can achieve MCMC standard errors smaller than that of the single-meiosis sampler, using same amount of computation time [Tong and Thompson, 2008]. However note that our method is less general than *lm_twoqtl* because it doesn't include polygenic component.

We have implemented the proposed method in a new lod score program *hg_lod*. Using previously simulated data [Sung et al., 2007b], we compare the performance of *hg_lod* and *lm_twoqtl*, for analyses using a two-major-locus additive model. Results show that *hg_lod* achieves the same level of power and accuracy in localizing QTL using much less computation time. We also evaluate the linkage information, especially that for localizing the weak QTL, that is conveyed by different amount of data by examining results on subsets of the whole dataset. Results shows that there is a fair amount of uncertainty in the inferred positions of the weak QTL, among subsets of data.

To demonstrate the effectiveness of *hg_lod* for SNP data, we use previously simulated marker data [Wijsman et al., 2006] to generate trait data using the trait model used by Sung and Wijsman [2007]. This model has no polygenic component but has epistatic effect to allow the interaction between two loci. In addition, the environmental effect takes a larger portion in this model than it does in model used by Sung and Wijsman [2007]. Results show that with *hg_lod*, both trait loci can be accurately

localized.

The efficiency of our method also facilitates sensitivity analyses to be performed across a wide range of parameter values. As for the VC method, the marker-data-based approach decouples the computation of trait likelihoods from the sampling. This allows analyses with different trait models be performed without re-sampling the realizations of multi-locus descent states. Results shows that the localization of two trait loci are relatively insensitive to the allele frequencies. The power of the linkage detection drops continuously as parameter values are gradually deviated from the simulation truth.

2.2 Lod Score Methods for Two-Locus QTL Data

The computational complexity of traditional methods for exact lod score, i.e., Elston-Stewart algorithm and Lander-Green algorithm, is exponential either in the number of loci or the pedigree size. For multipoint analysis on extended pedigrees with data on a large number of markers, exact algorithms become inpractical, and lod scores are typically estimated by MCMC methods [Lange and Sobel, 1991; Thompson, 2011]. In addition, two-locus trait model makes the MCMC approach the only computationally efficient choice for linkage analysis.

2.2.1 Analysis Model

Our analysis model assumes that the quantitative trait is affected by two major genes. The vector of trait values Y_T can be decomposed as follows:

$$Y_T = Q + E,$$

where Q is a two-locus QTL effect, and E is the residual effect. Q is modeled using discrete genotypic mean for each of the 9 two-locus genotypes. When there is no epistasis, $Q = Q_1 + Q_2$, where Q_1 and Q_2 are single-locus QTL effects, each specified by 3 genotypic means. E is modeled as normally distributed with mean 0 and variance

$I{\sigma_e}^2$. We assume the trait model parameters, i.e., the genotypic means and σ_e, are known. We assume that the marker data Y_M are observed without error and that the genetic map positions and the population allele frequencies of the marker loci are known.

2.2.2 MCMC Estimation of Lod Scores at Pairs of Loci

For two-QTL model, the lod score at a pair of hypothesized map positions (λ_1, λ_2) is

$$\mathrm{lod}(\lambda_1, \lambda_2) = \log_{10} \frac{\mathrm{P}_{\lambda_1,\lambda_2}(Y_T, Y_M)}{\mathrm{P}_{-\infty,\infty}(Y_T, Y_M)}$$

where the null hypothesis $(-\infty, \infty)$ denotes that the trait loci are not linked either to the marker loci or to each other. Since the trait data are independent of the marker data under the null hypothesis, we have

$$\frac{\mathrm{P}_{\lambda_1,\lambda_2}(Y_T, Y_M)}{\mathrm{P}_{-\infty,\infty}(Y_T, Y_M)} = \frac{\mathrm{P}_{\lambda_1,\lambda_2}(Y_T|Y_M)}{\mathrm{P}(Y_T)}.$$

Using inheritance vectors as latent variables, we have

$$\begin{aligned} \mathrm{P}_{\lambda_1,\lambda_2}(Y_T|Y_M) &= \sum_S \mathrm{P}_{\lambda_1,\lambda_2}(Y_T, S|Y_M) \\ &= \sum_S \mathrm{P}_{\lambda_1,\lambda_2}(Y_T|S)\mathrm{P}(S|Y_M) \\ &= \sum_S \mathrm{P}(Y_T|S_{\lambda_1}, S_{\lambda_2})\mathrm{P}(S|Y_M). \end{aligned} \tag{2.1}$$

The above derivation uses the fact that conditional on the inheritance vectors, the probability of the trait data is independent of the marker data and the trait data depend on the inheritance vectors only through the ones at the QTL. At the hypothesized position (λ_1, λ_2), the computation in Eq. 2.1 can estimated using MCMC methods by first sampling S conditional on Y_M, then averaging over N realizations using the inheritance vectors, i.e.,:

$$\widetilde{\mathrm{P}}_{\lambda_1,\lambda_2}(Y_T|Y_M) = \frac{1}{N}\sum_{i=1}^{N} \mathrm{P}(Y_T|S^i_{\lambda_1}, S^i_{\lambda_2}). \tag{2.2}$$

2.3 Models For Likelihood Computation

2.3.1 FGL graph and ibd Graph

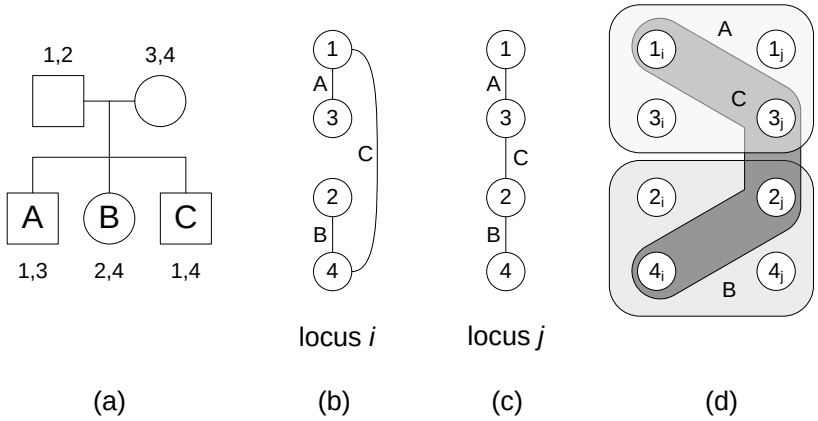


Figure 2.1: Two-locus FGL graph

We first show that how $\Pr(Y_T|S_\lambda)$ can be evaluated at a single locus λ, under a single-locus trait model. As we have introduced in Chapter 1.1, the descent of the founder genes to non-founders is determined by the inheritance vectors, and given inheritance vectors at one locus, the descent pattern can be illustrated by an FGL graph [Thompson, 2011]. Figure 2.1(a) shows a nuclear family with all three offspring observed. For each individual, the two FGL at locus i, paternal then maternal, are also shown. The FGL graph at locus i is shown in Figure 2.1(b), where the FGL are numbers and edges are labeled with the observed individuals. Since the inheritance pattern changes along the chromosome, the FGL graph will change accordingly. Figure 2.1(c) shows the FGL graph at another locus j, potentially linked to locus i. At locus j, individual C receives FGL 2 and 3 instead of 1 and 4.

Suppose at a particular locus, the set of all FGL is K and the allelic type of FGL k is $\mathcal{A}(k)$ with population allele frequency $q(\mathcal{A}(k))$. Let E denote the set of edges and $M(\varepsilon)$ be the set of observed individuals on edge ε. Let $k_1(\varepsilon)$ and $k_2(\varepsilon)$ be the

endpoints of edge ε. With the assumption that the founder genotypes are in Hardy-Weinberg equilibrium at each locus and linkage equilibrium among loci, the allelic types of distinct founder genomes are independent. So the probability of observed trait data conditional on the inheritance vector at locus λ, can be computed

$$\begin{aligned}\Pr(Y_T|S_\lambda) &= \Pr(Y_T|\text{FGL-graph}) \\ &= \sum_{\mathcal{A}} \Big(\prod_{k\in K} q(\mathcal{A}(k)) \Big) \Big(\prod_{\varepsilon\in E} \Big(\prod_{m\in M(\varepsilon)} \Pr(Y_m|\mathcal{A}(k_1(\varepsilon)), \mathcal{A}(k_2(\varepsilon)) \Big) \Big). \quad (2.3)\end{aligned}$$

The computation requires summation over $\mathcal{A}$ which is all possible assignments of allelic types to the FGL in the graph. It was first proposed by Thompson and Heath [1999] that this computation can be achieved by "peeling" the allelic assignment of types to founder genes.

Notice that the labeling of founder genomes is arbitrary, any two FGL graphs having same topology with respect to the edges are equivalent in likelihood computation. The node-unlabeled version is known as *ibd* graph. Each distinct *ibd* graph corresponds to an equivalence class in the set of all FGL graphs. Examples given by Thompson [2011] show that across the set of MCMC realizations, the number of equivalence classes can be much smaller than the number of FGL graphs. This results in a significant reduction in computing $\Pr(Y_T|Y_M)$ because we only need to evaluate $\Pr(Y_T|S_\lambda)$ for each equivalence class.

Now we consider the computation of $\Pr(Y_T|S_{\lambda_1}, S_{\lambda_2})$ under the two-locus model. We will need to specify the inheritance patterns at two loci jointly using two-locus FGL graphs, where we have two sets of distinct founder genome labels, one for each locus. Since each observed individual can receive up to 4 FGL at two loci, individuals sharing same FGL are represented as hyperedges, which are sets of FGL, instead of regular edges. Thus two-locus FGL graphs are in fact hypergraphs. Figure 2.1(d) shows the two-locus FGL graph at loci (λ_1, λ_2). FGL are suffixed to indicate their chromosomal positions and hyperedges are represented as shaded areas.

Similar to the single-locus case, there are equivalent two-locus FGL graphs, and

the two-locus *ibd* graph can be similarly defined. In a two-locus *ibd* graph, nodes associated with different loci are distinguishable, while nodes associated with the same locus are not. Since there is no straightforward way to directly testing the equivalence of two-locus FGL graphs, we propose a way of constructing equivalence classes in the set of all two-locus FGL graphs by combining information from both loci. First we find the set of equivalence classes for each locus. We then take pairwise intersections between equivalence classes from different loci. We now use an example to demonstrate how this works. Suppose that we have 6 realizations of inheritance vectors, from which we construct two sets of single-locus FGL graphs, one for each locus, and one set of two-locus FGL graphs. Within each set, graphs are numbered from 1 to 6. Across all three sets, graphs with the same number are from the same realization. Assume that the set of equivalence classes for the first locus is $\{\{1,2,3\},\{4,5,6\}\}$, and for the second locus is $\{\{2,4\},\{1,3\},\{5,6\}\}$. Taking pairwise intersections produces the set of equivalence classes for the two-locus graphs, i.e., $\{\{2\},\{4\},\{1,3\},\{5,6\}\}$.

There is no existing peeling program for two-locus FGL graphs. The most related work is by Sung et al. [2007b] and Sung and Wijsman [2007] whose approach to approximate the likelihoods is equivalent to importance sampling on two-locus FGL graphs. However, in this work, we aim to achieve a more accurate MCMC estimation by exact computation of the likelihood components, i.e., $\mathrm{P}(Y_T|S_{\lambda_1},S_{\lambda_2})$, using the Junction Tree algorithm that was developed for general graphical models.

2.3.2 Exact Computation on Factor Graphs

The hypergraph structure of two-locus FGL graphs fits well into a standard graphical model, the factor graph [Kschischang et al., 2001], for which many efficient inference algorithms have been developed over the years. Other popular graphical models, such as the Bayesian network [Jensen, 1997] and the Markov random field [Kindermann and Snell, 1980], can be equivalently transformed to the factor graph [Kschischang et al., 2001]. A factor graph $G=(X,F,E)$ is a bipartite graph consisting of random

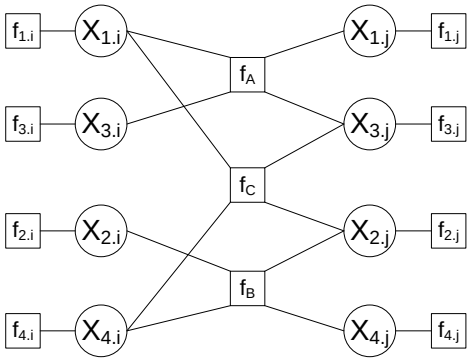


Figure 2.2: Factor graph representation of the two-locus FGL graph

variable nodes $X = \{X_1, X_2, \ldots, X_n\}$, factor nodes $F = \{f_1, f_2, \ldots, f_m\}$, and edges E. Each factor node corresponds to a function $f_k(S_k)$, where $S_k \subseteq \{X_1, X_2, \ldots, X_n\}$. A factor node f_k is connected to all variables in S_k. The factor nodes give a factorization of a joint function $g(X_1, X_2, \ldots, X_n) = \prod_{k=1}^{m} f_k(S_k)$.

The two-locus FGL graph in Figure 2.1(d) can be converted to the factor graph shown in Figure 2.2. The variable nodes correspond to the FGL, i.e., each variable node is a random variable whose sample space is all possible alleles at this locus. The factor nodes that are connected to only one variable node corresponds to the allele frequencies, i.e., $f_k = q(X_k)$, $k \in \{1.i, \ldots, 4.j\}$. Each multi-connection factor node corresponds to a hyperedge, and its factor function is the probability of the trait data of the observed individuals on that hyperedge, conditional on the genotype, e.g., $f_A = \Pr(Y_A|X_{1.i}, X_{3.i}, X_{1.j}, X_{3.j})$. The joint function

$$\begin{aligned} g(X) = {} & q(X_{1.i}) \cdot q(X_{2.i}) \cdot q(X_{3.i}) \cdot q(X_{4.i}) \cdot q(X_{1.j}) \cdot q(X_{2.j}) \cdot q(X_{3.j}) \cdot q(X_{4.j}) \\ & \cdot \mathrm{P}(Y_A|X_{1.i}, X_{3.i}, X_{1.j}, X_{3.j}) \cdot \mathrm{P}(Y_B|X_{2.i}, X_{4.i}, X_{2.j}, X_{4.j}) \\ & \cdot \mathrm{P}(Y_C|X_{1.i}, X_{4.i}, X_{2.j}, X_{3.j})), \end{aligned}$$

which is exactly in the form of the summand of Eq.2.3 and the summation over X is a standard graphical model computation, which can be carried out using the Junction Tree algorithm [Lauritzen and Spiegelhalter, 1988]. The Junction Tree algorithm is

a generalization of the peeling algorithms and there exist advanced implementations which can be easily integrated into other software programs. However when there are major-gene QTL, the polygenic effect cannot be handled by peeling [Thompson, 2000, Section 6.6], so in comparing to previous work, in our analysis model we combine the variance of the polygenic component into that of the environmental component.

2.4 Implementation

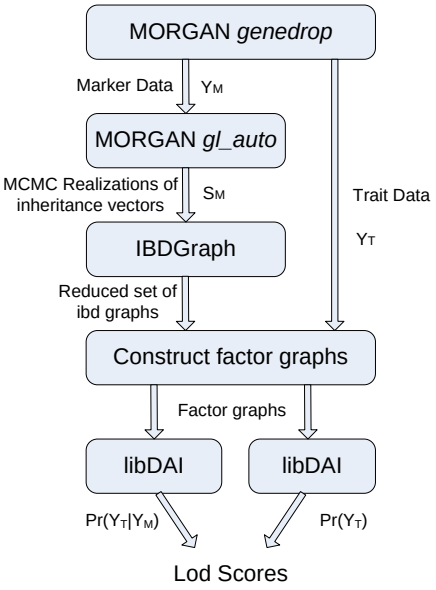


Figure 2.3: Flowchart of *hg_lod*

The lod score method described above was implemented as a Java program called hg_lod. hg_lod makes calls to several standalone programs, as shown in the flow diagram in Figure 2.3. The input and output data for the programs are labeled on the transitional arrows. A brief description of the used programs is as follows:

- ***gl_auto*** is a MORGAN-3 program that implements the locus-meiosis sampler with multiple meiosis update [Tong and Thompson, 2008], to realize inheritance

vectors conditional on marker data. ***gl_auto*** produces output in easy-to-parse FGL format in a compact way such that the size of the output is proportional to the number of recombination events.

- **IBDgraph** [Koepke et al., 2011] is a C library that takes *gl_auto* output and produces the set of equivalence classes, i.e., distinct *ibd* graphs at a single locus. The library is available for download at http://www.stat.washington.edu/thompson/Genepi/genepi.shtml.

- **libDAI** is an open source C++ program with a collection of algorithms for making probabilistic inference in discrete graphical models [Mooij, 2010]. libDAI supports factor graph format and it is easy to integrate into one's own program. It is also organized in a way that one can easily experiment with novel algorithms by making modifications at various code levels.

We tested hg_lod by running several linkage problems from the MORGAN gold standard test suite. For one-locus models, hg_lod produces exactly same lod scores as those from the MORGAN-2.9 program *lm_markers*, using the same MCMC realizations. For two-locus models, we test on examples with small pedigrees for which $\Pr(Y_T|S_i, S_j)$ can be computed exactly by brute force summation. For these small examples, the lod scores from hg_lod are exactly same as the lod scores produced by MORGAN program lm_twoqtl.

2.5 Simulation and Analysis of Data

We have two simulated data sets, both using a 52-member 5-generation pedigree (ped52) that contains 20 unobserved individuals. In both examples, there are 40 replicates of data simulated on this pedigree. The pedigree is shown in Figure 2.4 with unobserved individuals marked by a slash. In both examples, the marker data Y_M and the QTL genotypes are simulated using the program *genedrop* in MORGAN.

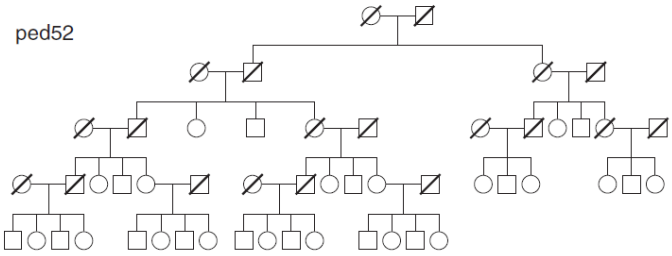


Figure 2.4: Pedigree used in Example 1 and 2

Then Q is generated from the simulated QTL genotypes. Example 1 were simulated using a model including two linked QTL with additive effect, a polygenic component and an environmental component. Example 2 includes two linked QTL in epistasis and an environmental component. There is no polygenic component in example 2. Simulation parameter values are shown Table 2.1. **Example 1** is same as the example 1 used by Sung et al. [2007b] for comparison purpose. Example 1 contains eight simulated microsatellite markers spaced 10 centi-Morgan (cM) apart, from 0 cM to 70 cM, each with six alleles (frequencies 0.3, 0.2, 0.15, 0.16, 0.1, and 0.1). The strong QTL (QTL 2) is at 55 cM and the weak QTL (QTL 1) is at 15 cM. Their parameters of the analysis model are shown in Table 2.1.

Sung et al. [2007b] computed lod scores both the markers and the midpoints between each pair of consecutive markers. To compute lod scores at the non-marker positions, we simulate inheritance vectors at these positions using the realized ones at markers. Let $S_{\bar{j}}$ be the inheritance vector at a non-marker locus $\bar{j}$, i.e., the midpoint of two consecutive marker loci j and $j+1$. Note that inheritance vectors are independent among meioses and in the absence of genetic interference in the transmission of DNA from parents to offspring, the inheritance vectors form a Markov Chain along the chromosome, so that $S_{i,\bar{j}}$ only depends on the inheritance vectors for the same i and

Table 2.1: Trait model parameter values.

		Geno Means[2]					Variance Components[4](%)				
Example		AA	Aa	aa	Allele Freq.[3]		QTL1	QTL2	epi.	poly.	env.
1	QTL1	-2	0	2	0.9	0.7	0.72 (11)	3.78 (58)	0 (0)	1 (15)	1 (15)
	QTL2	-3	0	3							
2	BB	3	0	-3	0.87	0.79	2.42 (7)	10.6 (31)	6.1 (18)	0 (0)	15 (44)
	Bb	1	3.5	6							
	bb	-1	7	15							

[1] Values for example 1 and 2 are same as those used by Sung et al. [2007b] and Sung and Wijsman [2007], respectively.

[2] For example 1, the alleles for both QTL are coded A and a. For example 2, the alleles for QTL1 are coded A and a, while the alleles for QTL2 are coded B and b. A geno mean is the mean QTL effect for a genotype.

[3] Allele frequency are for the small alleles, i.e., a or b.

[4] Contribution of each variable in the analysis model to the overal variance.

at two flanking markers, i.e.,

$$\Pr(S_{i,\bar{j}}|S) = \Pr(S_{i,\bar{j}}|S_{i,j}, S_{i,j+1}) = \frac{\Pr(S_{i,\bar{j}}, S_{i,j}, S_{i,j+1})}{\Pr(S_{i,j}, S_{i,j+1})}.$$

where S is the set of inheritance vectors at marker loci. Then we can use Eq. 2.2 where each of λ_1 and λ_2 can be either a marker position or a mid-point between two markers.

Example 2 is constructed to explore the effect of using SNP markers as opposed to the sparser microsatellite markers. It has 200 simulated SNP markers spaced 0.5 cM apart, from 0 to 99.5 cM. These simulated marker data were originally used by Wijsman et al. [2006]. The trait value was simulated using the trait model used by Sung and Wijsman [2007], which is epistatic. The strong QTL (QTL 2) is at 150th marker (74.5 cM) and the weak QTL (QTL 1) is at the 50th marker (24.5 cM). The parameters for the analysis model are shown in Table 2.1. Lod scores are computed

at markers every 5 cM apart, starting from the 10th marker (4.5 cM) to the 200th marker (99.5 cM).

Sung et al. [2007b] used the realizations from every 10th MCMC scans to calculate $\widetilde{\mathrm{P}}(Y_T|Y_M)$ due to the high cost of computing $\mathrm{P}(Y_T|S_{\lambda_1}, S_{\lambda_2})$. To compare with the work of Sung et al. [2007b], we use the realizations from every 10th MCMC scan for example 1. For example 1, we also exploited a different number of Monte Carlo sample sizes listed in Table 2.2. For example 2, we choose to use the realizations from every 30th MCMC scan out of a total of 3×10^4 scans. This, in practice, produces accurate results in a reasonable amount of time.

2.6 Results

2.6.1 Example 1

Lod scores for 40 replicates of data on the 52-member pedigree at an array of hypothesized locations are shown in the contour plot in Figure 2.5 (a). The highest lod score is 45.420 and it is achieved at (30 cM, 55 cM), shown as a dot in the figure. It is close to the true QTL locations, which is at the intersection of the dotted lines, (15 cM, 55 cM). The strong QTL location is estimated accurately, while the weak QTL position is 10 cM away from the true weak QTL location. However, the three hypothesized positions on the left of the highest lod score position exhibit high lod scores very close to the highest one. These are 45.418, 45.302 and 44.987 at 25 cM, 20 cM and 15 cM, respectively.

To evaluate the linkage information when only a smaller amount of data is available, we examine the lod scores from combining subsets of the 40 replicates. For each size from {10,15,20,25}, we generate 1000 random subsets with that size. We denote the data having a subset of size n as $n$ped52, e.g., data containing 10 replicates will be 10ped52. The highest lod score position for each subset is shown in Figure 2.6, for sizes 10, 15, 20 and 25, respectively. Highest lod score positions are shown as dots

Table 2.2: Comparison of *hg_lod* and *lm_twoqtl* with different MCMC sample sizes using Example 1.

MCMC	Time (min.)[1]				Highest		MCMC		
Sample	Samp.		Trait Lik.		Lod Score		Standard Error[2]		Positions
Size	lm	hg	lm	hg	lm	hg	lm	hg	
2×10^3	-	0.12	-	0.026	-	44.94	-	0.120/0.402	({20,25,30},55)
2×10^4	-	1.3	-	0.26	-	45.42	-	0.109/0.228	({15,20,25,30},55)
1×10^5	-	6.3	-	1.28	-	45.79	-	0.076/0.160	({15,20,25,30},55)
2×10^5	4.7	11.0	249	2.54	52.03	45.95	0.139/0.253	0.058/0.134	({15,20,25,30},55)

[1] Overall sampling time (*gl_auto*) and the average trait likelihood computation time for a single hypothesized position are listed.

[2] Batch mean estimate and actual MC standard error are listed, separately by back slashs. The MC standard errors shown are the largest among all replicates. The actual MC standard error is obtained by running 100 independent analyses for the replicate with the largest MC standard error.

and many of them are coincident. To depict the multiplicities, we plot the histogram along each axis. For example, out of 1000 10ped52, the number of subsets whose weak QTL positions are at 0 cM, 15 cM and 30 cM, is 173, 103 and 282, respectively. Out of 1000 10ped52, there are 890 subsets having strong QTL position at the true position. As the size of the subsets increases, the distributions of both strong and weak QTL positions become more centralized at (30 cM, 55 cM), the highest lod score position obtained using 40 replicates.

We also examine the impact of different MCMC sample sizes. We compare the computation time, MCMC standard errors, and highest lod score and its position with the work of Sung et al. [2007b], which used lm_twoqtl with 2×10^5 MCMC scans. The results are shown in Table 2.2. For 2×10^5 scans, *hg_lod* is about 28 times faster than *lm_twoqtl* with an MCMC standard error about two times smaller. It also can

be seen that for both programs, the estimated MCMC standard error is roughly two times smaller than the actual MCMC standard error. With much smaller number of MCMC sample size, i.e., 2×10^4, we obtain the MCMC standard error that roughly equals the one obtained with *lm_twoqtl* using 2×10^5 scans. For *hg_lod*, reducing the MCMC sample size has monotonic but negligible drops in the magnitude of the highest lod score, while the QTL are still accurately localized.

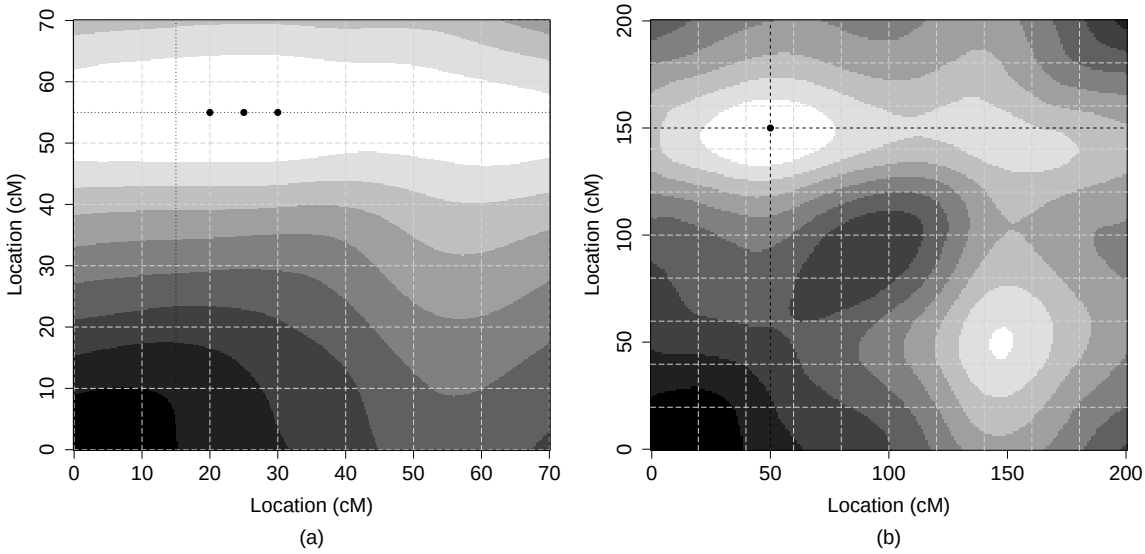


Figure 2.5: Contour plot of the lod scores for (a) example 1, and (b) example 2. The horizontal axis is the QTL 1 position, and the vertical axis is for QTL 2. Light color indicates high lod score, while positions of highest lod scores are marked by bold points

2.6.2 Example 2

The contour plot of two-dimensional lod scores is shown in Figure 2.5 (b) and other statistics are listed in Table 2.3. The highest lod score is 49.22 and it is attained at the true QTL position, which is $(50, 150)$. Comparing to example 1, the highest lod position is significantly distinguished from its neighboring positions in the magnitude

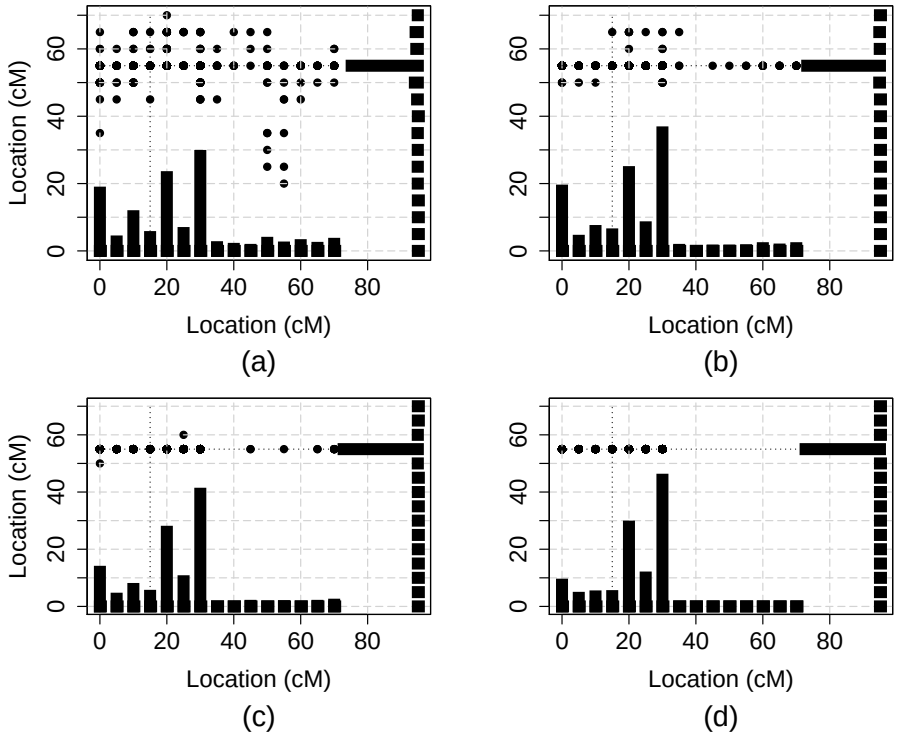


Figure 2.6: Distribution of highest lod score positions for 1000 (a) 10ped52, (b) 15ped52, (c) 20ped52 and (d) 25ped52

of the lod scores. Lod scores for positions to the left and right are about 43 and 40, respectively. Interestingly, there is another high peak at position $(150, 50)$, which is diagonally-symmetric to the true QTL position. The lod score at this position is about 39. Since the inheritance vectors for all 200 SNP markers are sampled, the sampling time is much longer than that of the run in example 1 with similar MCMC sample size. In Table 2.3, we report the estimated MCMC standard error for the smallest and the largest among all replicates. The average MCMC standard error is about two times smaller than that of the run in example 1 with similar MCMC sample size. For the replicate with largest estimated MCMC standard error, the actual standard error is about three times larger.

Table 2.3: Results for example 2.

	Time (min.)		Highest	MC Standard Error				
Sample Size	Samp.	Trait Lik.	Lod Score	Min.	Mean	Max.	Actual[1]	Position
3×10^4	25	0.19	49.22	0.004	0.018	0.054	0.18	(50,150)

[1] The actual MCMC standard error is obtained by running 100 independent analyses for the replicate with the largest MCMC standard error.

2.7 Sensitivity Analysis

The trait model in the previous analyses has the same parameter values used to generate the trait data, except that the polygenic variance is combined with the environmental variance. We are also interested in the variability of lod scores and highest lod score positions when the parameter values of the trait model differ from the simulation truth. Given the high computational efficiency of our method, we can re-evaluate lod score with many different parameter values. Further computational reduction is achieved with the elimination of the need to re-sample the inheritance vectors, given that our MCMC method samples the inheritance vectors conditional only on the marker data. We can reuse the MCMC realizations from previous computations and only recompute the likelihoods on the FGL graphs using different trait model parameter values. In this work, we only perform single-parameter sensitivity analyses by varying the parameter of interest one at a time while keeping other parameters at their true values.

We use Example 2 for sensitivity analysis and the results are shown in Figure 2.7. The parameters under analysis are the allele frequencies of QTL 1 and QTL 2, and the environmental variance. The sensitivity of the estimation of the QTL location can be expressed by a pair of percentages by normalizing the parameter range over which the estimation is close to the simulation truth by the allowable range of the

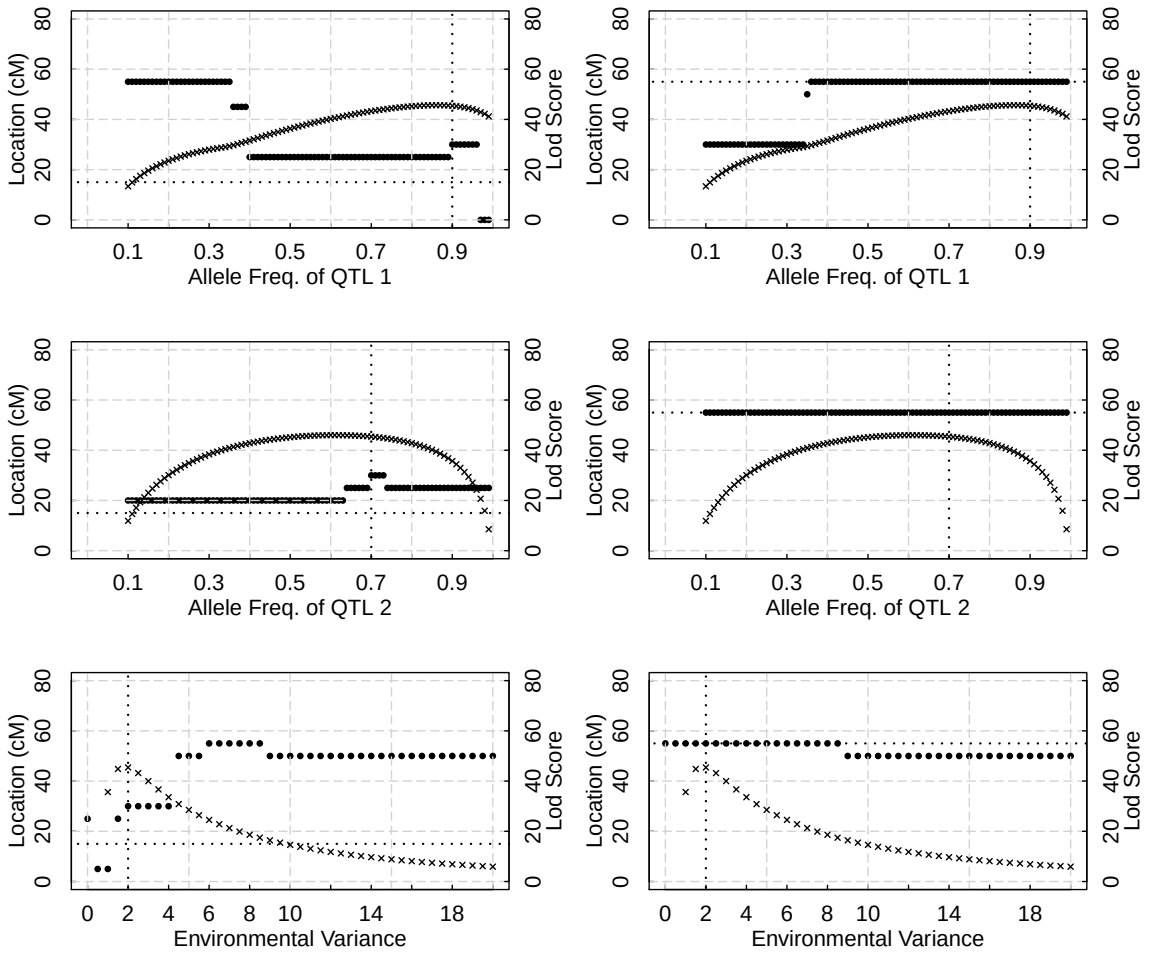


Figure 2.7: Sensitivity results for the allele frequency of QTL 1 (first row), QTL 2 (second row) and the environmental variance (third row) as a function of the parameter values. The allele frequencies of QTL 1 and of QTL 2 ranges from 0.1 to 1 with stepsize 0.01. The environmental variance ranges from 0 to 20 with stepsize 0.5. In the first column of Figure 2.7 are shown the highest lod scores (crosses) and the QTL 1 positions (solid points) where the highest lod score is attained, versus the parameter values. The second column shows the highest lod scores (crosses) and the QTL 2 positions (solid points) where the highest lod score is attained. The true parameter values and the true QTL positions are indicated by dotted lines.

parameter. Suppose the parameter range over which the estimated location is close is [A, B], the simulation truth is S, and the allowable range of the parameter is [L, H],

then the pair of percentages will be $[(A-S)/(S-L), (B-S)/(H-S)]$. Large ranges indicate insensitivity. When the varying parameter is QTL 1 allele frequency, the range for QTL 1 and QTL 2 are [-56%, 67%] and [-61%, 100%], respectively. When the varying parameter is QTL 2 allele frequency, the ranges for QTL 1 and QTL 2 are [-100%, 100%] and [-100%, 100%], respectively. When the varying parameter is the environmental variance, the ranges for QTL 1 and QTL 2 are [-100%, 11%] and [-100%, 100%], respectively. It can be seen that the estimated positions of both QTL are quite insensitive to the allele frequencies.

There are two interesting observations in the sensitivity analysis. The first one is that outside the insensitive region, the estimated position of one QTL coincides the estimated position of the other QTL in the original analysis. This can be seen for QTL 1 when the varying parameter is QTL 1 allele frequency and the value is less than 0.35, for QTL 2 when varying parameter is QTL 1 allele frequency and its value is less than 0.35, and for QTL 1 when the varying parameter is the environmental variance and its value is greater than 4. The second observation is that for certain parameter values, the estimated position of QTL 1 is closer to the simulation truth than the one in the original analysis. This happens when the allele frequency of QTL 1 is between 0.89 and 0.4, when the allele frequency of QTL 2 is outside of [0.7, 0.74], and when the environmental variance is less than 2.

Even though both QTL 1 and QTL 2 positions are insensitive in a reasonably large region around the true parameter value, the lod scores continuously change with the parameter values. For all three parameters, the lod score remains significantly high within the entire parameter range and peaks at the value that is very close to the simulation truth.

We have only considered single-parameter sensitivity, although in reality there is uncertainty about the true values of all parameters. We also did not show sensitivity results for the genotypic means, since these are jointly constrained. While our approach permits the computation of lod scores under multiple models, multi-parameter

sensitivity analysis becomes computationally intensive, and the results hard to display. For specific joint parameter changes of interest, defining a line in the parameter space, the same analysis as we present for single parameters could be undertaken. Rather than to demonstrate any particular robustness, our goal in this work is to show how our approach enables the computation of lod scores for multiple complex trait models without reconsideration of the marker data. This in turn enables any sensitivity analysis that might be needed in a real-data analysis.

In an analysis of real trait data, it is usual to start with a simple model for linkage detection, and then refine the joint two-locus model based on the location information obtained from the linkage detection. In some cases, the second QTL will only become apparent when the first is already taken into account. Linkage detection is far more robust to trait model misspecification than is the localization of the genes [Clerget-Darpoux et al., 1986].

2.8 Discussion

Based on the framework of graphical modesl, we have developed a new program, *hg_lod*, for linkage analysis of a two-locus QTL model. The program can handle general trait models, including two QTL effects and an environmental effect. *hg_lod* uses similar ideas in the MORGAN program, *lm_twoqtl*, which is the first available software for two-locus linkage analysis for large pedigree with a lot of missing data. *hg_lod* dramatically reduces the computation time which is a main issue in *lm_twoqtl*. As pointed out by Sung et al. [2007b], the capability of simultaneous detecting of two QTL is critical in accurate localization, especially when two QTL are linked and their contributions to the overall phenotypic variance are comparable. Also discussed by Sung et al. [2007b], a full model-based approach is expected to perform better than allele sharing methods and VC methods. These facts give rise to our program and whenever there are a large amount of data on extended pedigrees, our program provides a efficient tool for linkage analysis.

The lod score plots in Figure 2.5 both exhibit some diagonal symmetry which can be understood as due to the symmetry of the roles of two QTLs in the trait model. In both examples, the allele frequencies of both QTLs do not differ much (0.9 vs. 0.7 in example 1 and 0.87 vs. 0.79 in example 2). In example 1, the genotypic means of both QTLs have very similar values. In example 2, the genotypic means for two genotypes of which the roles of two QTLs are swapped have similar values (7 for Aabb and 6 for aaBb). In Figure 2.5(b), there is a second peak at the mirror point to the highest lod score location, where the locations of the two QTLs are swapped. Note that there is a second peak in Figure 2.5(b), which is missing in Figure 2.5(a). This is explained by the SNP data providing more information on the latent inheritance, so that the symmetry in the trait model is more clearly reflected in the lod scores.

In our analysis model we combine the variance of the polygenic component into that of the environmental component, since the polygenic effect cannot be peeled in the presence of two major-gene QTL because to do so requires summing over configurations of the full set of FGLs which contradicts what peeling does. In the example in this chapter, the penalty associated with this simplification causes a 12% decrease in the highest lod score, compared with *lm_twoqtl*. There is also uncertainty induced in the neighborhood where the highest lod scores are attained. Along the axis for QTL 1, the weak QTL, we obtain a flat region rather than a single peak. However the true QTL 1 position is almost always included in this flat region. When there is no polygenic component as in example 2, we achieve a sharp peak at the true QTL position even though the distribution of the variance between two QTL still follows a strong-weak pattern. In case of dealing with a model with a large polygenic component, one might first obtain a quick identification of QTL positions using *hg_lod*, then perform a fine localization in the region of high lod score using *lm_twoqtl* and taking into account the polygenic effect.

Results in Figure 2.6 show the linkage information carried by different subsets of data. Let us regard the size of data, i.e., number of pedigree replicates, as a variable.

The uncertainty of linkage information associated with each value of this variable is visualized as two histograms in Figure 2.6, one for each QTL. Not surprisingly, there is more uncertainty in the localization of QTL 1 than that of QTL 2. As the data size increases, the histogram for QTL 2 quickly converges to a single peak. Only about half of the data is sufficient to localize the strong QTL. The estimated position for the weak QTL converges slower and it ends up with two peaks, neither being the true QTL 2 position.

A major advantage of *hg_lod* is that it enables lod scores to be computed efficiently for multiple trait models with no additional MCMC sampling. We have performed single-parameter sensitivity analysis for example 1. The analyzed parameters are the allele frequencies and the variance of the environmental effect. The localizations of both QTL are insensitive to the misspecification of the allele frequency for the strong QTL. The localizations of both QTL are still insensitive, yet to a lesser degree, to the allele frequency of the weak QTL. The specification of the value of environmental variance is important in accurate localization of the weak QTL. The sensitivity analysis also demonstrates the effectiveness of the MOD-score method [Risch, 1984] for the particular trait model of example 1. The lod score is always maximized at the true parameter value for the allele frequency of QTL 1 and the variance of the environmental effect. The lod score is maximized at a value that is very close to the truth for the allele frequency of QTL 2.

Table 2.2 shows that when the MCMC sample size increases by a factor of 100, the MCMC standard error only decreases by a factor of about 3, suggesting that the MCMC runs mixed poorly using these microsatellite markers. By contrast, Table 2.3 shows very small MC error using the SNP data. Once the marker data are dense enough to determine the relevant inheritance information lod score will not change with the addition of denser SNPs [Wijsman et al., 2006]. In fact, using dense SNP data will not only fail to increase the signal strength, but may introduce noise due to linkage disequilibrium (LD) and MCMC mixing problems [Sieh et al., 2005;

Wilcox et al., 2005]. Although the optimal SNP density will depend on the specific dataset, on extended multi-generation pedigrees which provide substantial inheritance information typically 1 SNP per cM is sufficient to capture this information [Wijsman et al., 2006]. In our example, the SNPs are 0.5 cM apart, so that LD is not an issue, and the multiple-meiosis sampler in *lm_twoqtl* still works well to achieve good MCMC mixing and very small MC standard error.

Chapter 3

REGION-BASED APPROXIMATE INFERENCE METHOD

Exact inference algorithms such as belief propagation (BP) and junction tree algorithm can be very efficient as they reduce the exponential complexity to be only in the tree-width. For graphs with large tree-width, approximate inference methods, such as loopy BP and generalized BP are used as an alternative. Theoretical analysis of convergence issues of loopy BP and generalized BP algorithm have been under study in the framework of region graph and some relationship was established between BP-like algorithms and constrained optimization. In this chapter, we introduce region graphs and their structural properties which closely related to the convexity of the underlying optimization problem. This chapter will serve as a background for the development of the algorithms in next two chapters.

3.1 Variational Methods on Markov Random Field

In this chapter, we focus on a type of graphical model, known as Markov random fields (MRF). No generality is lost because there always exists equivalent transformation between Markov random field and other common graphical models, such as Bayesian networks and factor graphs [Kschischang et al., 2001]. In addition, algorithms that developed for one type of graphical model usually have mapped to other common graphical models too. Formally speaking, a Markov random field (also known as Markov network) is a triple $(\mathcal{G}, X, \Psi)$, where $\mathcal{G} = (V, E)$ is a undirected graph and X is a discrete random vector indexed by V. Ψ is the set of potential functions $\Psi_c(X_c) : S_c \to \mathbb{R}$, where c is a clique in $\mathcal{G}$ and S_c is the sample space of the random

vector X_c. The joint probability distribution of X factorizes over the cliques of $\mathcal{G}$:

$$P(X) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \Psi_c(X_c)$$

where Z is the normalization constant (also known as partition function) and $\mathcal{C}$ denotes the set of all cliques in the graph.

In probabilistic inference, desired quantities include the partition function and the marginal distributions over subsets of X, which require summation over an exponential number of configurations. The Cluster Variation Method (CVM) is one type of approximate inference technique that cast the inference problem to a minimization problem [Pelizzola, 2005]. CVM uses the concept of Gibbs free energy which is defined as follows:

$$\begin{aligned} F(P) &= -\sum_{\alpha \in \mathcal{C}} \sum_{X_\alpha} P(X_\alpha) \log \psi_\alpha(X_\alpha) + \sum_X P(X) \log P(X) \\ &\equiv E(P) - S(P), \end{aligned}$$

where $E(p)$ and $S(p)$ are referred as energy and entropy, respectively. In a variational point of view, the Gibbs free energy can be viewed as a function of the probabilities. Minimization of the free energy function is equivalent to minimizing the Kullback-Leibler (KL) divergence. The KL divergence between two probability distributions $Q(X)$ and $P(X)$ is defined as

$$D_{KL}(Q \parallel P) = \sum_X Q(X) \log \frac{Q(X)}{P(X)}.$$

Note that $P(X) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \Psi_c(X_c)$, we have

$$\begin{aligned}\sum_X Q(X)\log\frac{Q(X)}{P(X)} &= -\sum_X Q(X)\log\frac{1}{Z}\prod_{c\in\mathcal{C}}\Psi_c(X_c) + \sum_X Q(X)\log Q(X) \\ &= \sum_X Q(X)\log\frac{1}{Z} - \sum_X Q(X)\log\prod_{c\in\mathcal{C}}\Psi_c(X_c) + \sum_X Q(X)\log Q(X) \\ &= \log Z - \sum_X Q(X)\log\prod_{c\in\mathcal{C}}\Psi_c(X_c) + \sum_X Q(X)\log Q(X) \\ &= \log Z + F(Q).\end{aligned}$$

Assume $P(X)$ is known, we search for a probability distribution $Q(X)$ that minimizes $D_{KL}(Q \parallel P)$. When Q and P are identical, we have the global optimal solution where the KL divergence is zero. The optimal value of the free energy equals the negative of logarithm of the partition function.

However the expression and evaluation of the entropy term still requires summation of an exponential number of configurations. In CVM, the exponential entropy term is approximated by a sum of marginal entropies. Each of these marginal is defined on a subset of nodes (variables) and these subsets are called regions, which will be introduced in next Section. We have

$$\begin{aligned}-S(p) \equiv \sum_X p(X)\log p(X) &\approx -\tilde{S}(p) \\ &= \sum_{\gamma\in\mathcal{R}} c_\gamma S_\gamma(p) = \sum_{\gamma\in\mathcal{R}} c_\gamma \sum_{X_\gamma} p(X_\gamma)\log p(X_\gamma)\end{aligned}$$

Here $\mathcal{R}$ denotes a collection of *regions* and the parameter c_γ is called *Moebius number* or *overcounting number*. We will introduce overcounting number later in next section. The approximate variational free energy is defined as

$$\tilde{F}(P) = E(p) - \tilde{S}(p).$$

The approximate free energy consists of marginal distributions that are not known. These marginals sometime are what we would like to evaluate. By turning the

marginal distributions into variables, we have an approximate free energy function. We call these marginal distribution variables pseudo-marginals.

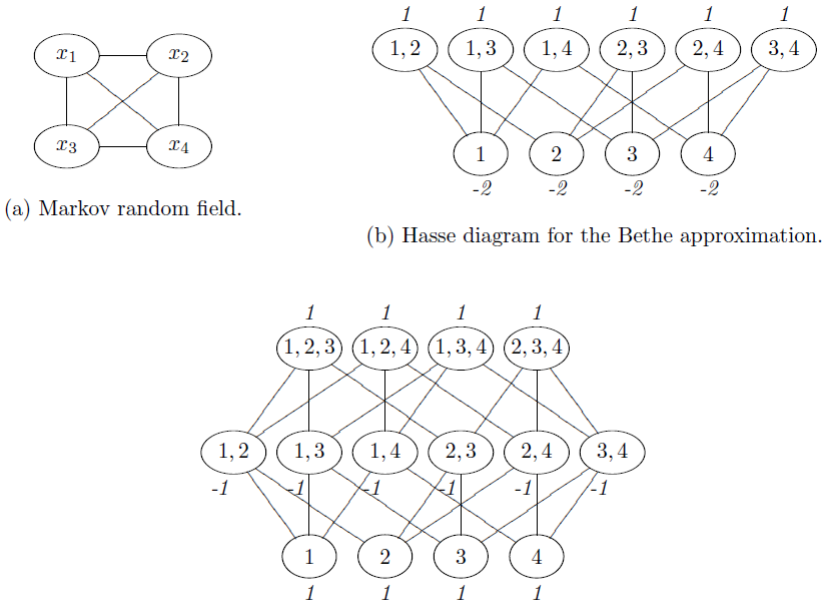
Minimization of the approximate free energy function is equivalent to minimizing an approximate form of the KL divergence by replacing the entropy part with an approximate entropy. The closer the approximation is, the exacter solution we can get. The free energy function is usually not optimized using gradient method. Instead, it is minimized by some local updating of the pseudo-marginal variables, such as in BP algorithm. The algorithm terminates when the pseudo-marginal variables reaches a locally consistent state. The solution can be used as an estimation of the partition function and true marginal distributions.

3.2 Region Graphs

A region γ is simply a subset of nodes. The random variables associated with region γ is denoted by X_γ. There are two types of regions, namely, outer regions and inner regions. If the nodes associated with a region β are included in a subset of nodes associated with region α, then we say α is a super-region of β and β is a sub-region of α. Regions that have no super-regions are called outer regions. An inner region corresponds to the intersection of regions. An inner region could be the intersection of multiple outer regions, multiple inner regions or a set of outer and inner regions.

Let $\mathcal{O}$ denote the set of all outer regions and $\mathcal{I}$ denote the set of all inner regions. We have $\mathcal{R} = \mathcal{O} \bigcup \mathcal{I}$. A particular selection of regions can be visualized by a region graph, where each node represents a region and the edges between nodes correspond to the inclusion relationship of the regions, i.e., if $\gamma \subset \gamma'$, then there is an edge between γ and γ'. Let us use $\mathcal{E}$ to denote the set of edges in a region graph and $R(\mathcal{O}, \mathcal{I}, \mathcal{E})$ to denote a region graph. Figure 3.1 shows two types of common regions graphs. Figure 3.1(b) has inner regions that are not overlapping. In Figure 3.1(c), inner regions do overlap.

The overcounting number also known as Moebius number is defined through the



(a) Markov random field.

(b) Hasse diagram for the Bethe approximation.

(c) Region graph for the Kikuchi approximation.

Figure 3.1: Example region graph

Moebius formula:

$$c_\gamma = 1 - \sum_{\gamma' \subset \gamma} c_{\gamma'}$$

and by definition the overcounting number of an outer region is 1.

Using notation of Bethe region graph, we can define pseudomarginals below.

Definition For a Bethe region graph $B(\mathcal{O}, \mathcal{I}, \mathcal{E})$, the pseudo-marginal of a region $\gamma \in \mathcal{R}$ is

$$q_\gamma \in [0,1]^{|S_\gamma|}.$$

where S_γ is the sample space of the random vector X_γ. Let x_γ be a point in the sample space S_γ. It can be used to index q_γ such that $q_\gamma(x_\gamma)$ is a component of q_γ. The pseudo-marginals of the Bethe region graph is a vector of the form $q = (\ldots, q_\alpha, \ldots, q_\beta, \ldots)$, where α is an index running over $\mathcal{O}$ and β is an index running over $\mathcal{I}$. Our notations

also allow us to use a subset of regions to index the pseudo-marginal vector. $q_{\mathcal{S}}$ is the collection of q's components that are associated with the regions in $\mathcal{S}$.

Because these variables represents probability distributions, they have to satisfy the positivity and normalization constraints:

$$\begin{cases} q_\gamma(X_\gamma) \geq 0 \quad \forall \gamma \in \mathcal{R} \; \forall X_\gamma \\ \sum_{X_\gamma} q_\gamma(X_\gamma) = 1 \quad \forall \gamma \in \mathcal{R} \end{cases}$$

3.3 Bethe Approximation

The Bethe approximation requires that the inner regions are intersections of only outer regions, i.e., there is no inner regions that are intersections of other inner regions. Under this condition, the Moebius number can be readily computed:

$$c_\gamma = \begin{cases} 1 & \text{if } \gamma \in \mathcal{O} \\ 1 - \deg(\gamma) & \text{if } \gamma \in \mathcal{I} \end{cases}$$

where $\deg(\cdot)$ denotes the degree of a node in the region graph. Figure 3.2 shows an example of Markov network and two possible region graphs associated with it.

Bethe free energy is an approximation to the free energy with marginal entropies:

$$-S(p) \equiv \sum_X p(X) \log p(X) \approx \sum_{\gamma \in \mathcal{R}} c_\gamma S_\gamma(p)$$

The Bethe free energy function can be expressed as a function of the pseudo-marginals:

$$\begin{aligned} F_\beta(q) = & \sum_{\alpha \in \mathcal{O}} \sum_{X_\alpha} q_\alpha(X_\alpha) \log \frac{q_\alpha(X_\alpha)}{\psi_\alpha(X_\alpha)} \\ & + \sum_{\beta \in \mathcal{I}} c_\beta \sum_{X_\beta} q_\beta(X_\beta) \log q_\beta(X_\beta). \end{aligned}$$

The Bethe free energy function is usually minimized by BP-like algorithms [Heskes et al., 2003; Teh and Welling, 2001b; Yedidia et al., 2000; Yuille and Rangarajan,

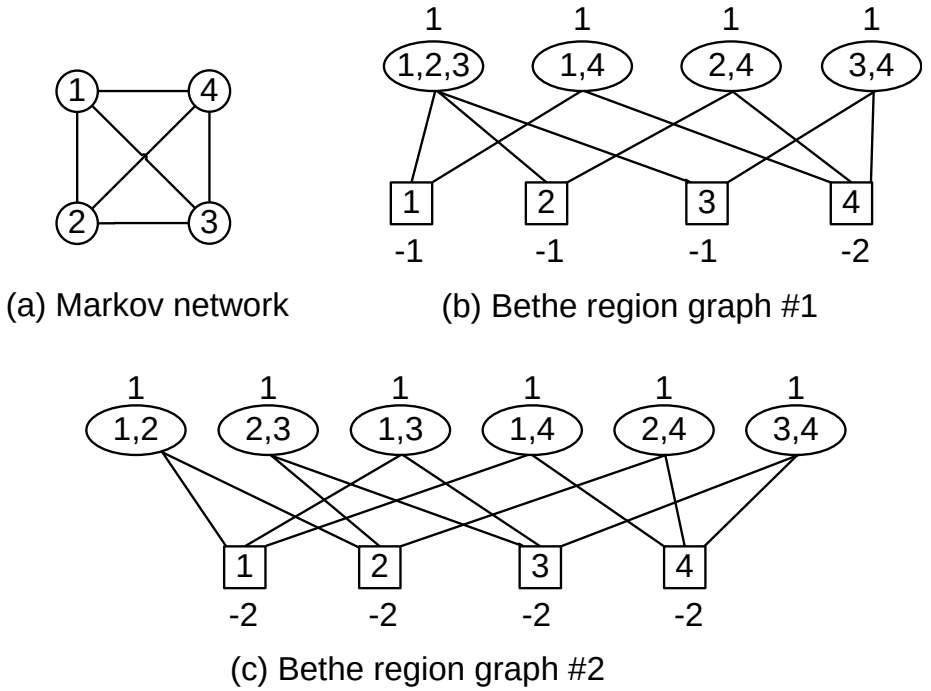


Figure 3.2: Markov network and region graphs. In (b) and (c), the ovals represent outer regions, and the boxes represent inner regions. The overcounting numbers are shown beside the regions.

2003]. It can be shown that the solution of the BP algorithm is a stationary point of the Bethe free energy function [Yedidia et al., 2000]. When the Bethe region graph contains at most one cycle, the Bethe free energy function equals the Gibbs free energy and is a strictly convex function [Yedidia et al., 2000]. The BP algorithm finds the global optimal solution which are true marginal distributions of the outer regions (pair of nodes) and single-node inner regions.

3.4 Kikuchi Approximation

In Kikuchi approximation, region graphs are allowed to have intersections of the inner regions [Heskes, 2006; Heskes et al., 2003; Pelizzola, 2005; Yedidia et al., 2000]. There are multiple alternative region graph representations of a single Markov random

field. A typical Kikuchi region graph can be constructed as follows. The minimal choice of the outer regions is the original set of cliques $\mathcal{C}$, but we can also choose to combine cliques and construct larger ones, similar to the process of triangulation. For convenience, we redefine the potentials correspondingly, i.e., such that there is precisely one potential per outer region. Given these outer regions, we construct new regions by taking the intersections of these outer regions, the intersections of intersections, and so on, until no more intersections can be made. Figure 3.1(c) shows an example region graph for the Markov random field in Figure 3.1(a). The associated overcounting number for each node is depicted besides the nodes.

Similarly the Kikuchi free energy function is a function of the pseudo-marginals of the regions in the Kikuchi region graph, and we minimize it to obtain an approximate to the exact free energy and marginal distributions. The Bethe free energy can be considered as a special case of the Kikuchi free energy. The Kikuchi free energy function is minimized using the so-called Generalized Belief Propagation (GBP) algorithm, which is given in Yedidia et al. [2000]. As its name indicated, it is generalized from BP and uses more complex message passing schemes between inner and outer regions.

Note that both the Kikuchi and Bethe free energy are approximations rather than bounds. Give the solution of the approximate inference, there is no theoretical way to assess how close the approximate solution is to the true one.

Different selections of regions yields accuracy to the different level. For sparsely connected graph, approximation using small regions works well. For highly connected graphs with strong interactions the CVM may not produce accurate estimates. In general, the accuracy can be improved by including larger regions in the approximation. The computational complexity grows exponentially to the size of the regions, making it a trade-off between region size and accuracy. In addition, how to group nodes into region also affects the approximation accuracy. Welling [2004] have proposed a sequential approach where new regions are added bottom-up to the region

graph. The method uses a novel measure to choose new region to add but its application to general problems are not demonstrated. The selection of regions is still remain a manual art and relies on common sense.

3.4.1 Convergence

It has been shown that BP algorithm converges for graphs that either are singly-connected or contains only one loop [Weiss, 2000] . For singly-connected graph, it is guaranteed to converge no matter what message passing scheme is exploited, but converges in two rounds if the messages are passed sequentially, starting from a root node, following a breath-first-search manner. The convergence condition has been proved by establishing a connection between the fixed point of the algorithm and the stationary point of the Bethe free energy function. For both singly-connected graphs and graphs with only one loop, the associated Bethe free energy function is convex over the positivity and normalization constraints.

The convergence of GBP is studied in Yedidia et al. [2000]. It has been shown that the relationship between the fixed point and the stationary point still holds in GBP. Heskes has given sufficient conditions for a Kikuchi free energy function to be convex along with methods to check for convexity [Heskes, 2006]. The condition requires that the overcounting numbers satisfy a set of inequalities.

The works that shows the connection between BP fixed points and stationary points of the free energy function are from a dual point of view, because BP is a dual rather than primal algorithm to solve the Bethe variational problem, i.e., beliefs are infeasible before convergence. Wener [2010] shows that the BP is indeed primal for a single function of reparameterizations. That means, BP seeks for a stationary point of a single function, without any constraints. This function has a very natural form: it is a linear combination of local log-partition functions, exactly as the Bethe entropy is the same linear combination of local entropies.

Chapter 4

CONVEX RELAXATION METHOD AND ITS CONVERGENCE ANALYSIS

As we introduced in Chapter 1, likelihood computation in linkage analysis is equivalent to probabilistic inference in graphical models. However, exact inference method can be computationally intractable for modern genetic data. When exact inference is infeasible or impractical, one often resorts to BP-like approximate inference methods. While loopy BP algorithm performs well in practice when minimizing Bethe free energy, its convergence is not guaranteed for cyclic graphs. Convergent algorithms are also developed for Bethe free energy to replace loop BP, however with much greater computational complexity. Both convergent and nonconvergent algorithms for minimizing convex approximation of Bethe free energy also exist. In this chapter, we focus on Bethe free energy and propose a message passing algorithm, called convex relaxation method, as a natural generalization of Unified Propagation and Scaling (UPS) algorithm to serve as a convergent alternative of loopy BP. Our aim is to obtain the same level of accuracy achieved by loop BP, with comparable computational efficiency. We first give a convergence analysis of the proposed method. We derive sufficient conditions for the method's convergence from a mathematical programming point of view. We then conduct experiments on simulated large scale *ibd* graphs.

4.1 Introduction

Probabilistic inference in graphical models is a prevalent task in statistics and artificial intelligence. The ability to perform this inference task efficiently is critical in large scale applications. In particular, such problems arise in the analysis of genetic data

on large and complex pedigrees [Cannings et al., 1978] or data at large numbers of markers across the genome [Abecasis et al., 2002]. Even after marker data have been analyzed, and the patterns of shared genome segments among relatives inferred, the subsequent analysis of trait data can involve computations on large numbers of potentially large and complex graphs [Thompson, 2011].

However, exact probabilistic inference in graphical models can be computationally intractable, indeed, NP-hard [Cooper, 1990]. The complexity of most exact inference algorithms is exponential in the tree-width of the graph [Chandrasekaran et al., 2008]. For graphs with large tree-width, one has to resort to approximate methods. Belief Propagation (BP) algorithm is a message passing algorithm that minimizes the so-called Bethe free energy function as an approximation to the KL-divergence. As extensions of the exact BP to cyclic graphs, loopy BP and generalized BP [Yedidia et al., 2000] work well in practice. However these algorithms are not expected to produce a global minimum nor are they guarantee to converge. Mooij and Kappen recently derived strong sufficient condition for the convergence of parallel loop BP by showing that the algorithmic mapping is a contraction [Joris and Kappen, 2005]. There have been efforts towards general convergent BP-like algorithms, such as Concave-Convex Procedure (CCCP) [Yuille and Rangarajan, 2003] and UPS algorithm [Teh and Welling, 2001b]. CCCP works by decomposing the free energy into concave and convex parts, and by using a discrete update rule which matches the derivatives of the two parts, it guarantees a decrease in the free energy at each iteration step until a local minimum is reached. However as a double-loop algorithm, its computational efficiency is inferior, especially in cases where loopy BP performs well.

Since the Bethe free energy function is nonconvex in general cyclic graphs, research has also been focusing on minimizing various types of approximation to Bethe free energy function. Heskes [2006]; Heskes et al. [2003] developed a double-loop algorithm that minimizes a sequence of convex upper-bounds that get tighter gradually over iterations. Then came the seminal algorithm for minimizing Tree Reweighted

(TRW) free energies by Wainwright et al. [2005]. These energies arise from a convex combination of log partition functions of spanning trees that forms a natural upper bound on the exact log partition function. Winn and Bishop [2005] developed a message passing algorithm optimizing a special variational bound. Globerson and Jaakkola [2007] proposed a convergent algorithm for tree-reweighted free energies based on orient tree. Hazan and Shashua [2008] developed a convergent algorithm as well for general convex energies by imposing strict non-negativity constraints on certain coefficients of the entropy terms. Later on Meltzer et al. [2009] provide a unifying view that relates convergence to the order in which message updates are performed. More recently, Jancsary and Matz [2011] devised convergent algorithms using tree-reweighted energies arising from a small number of spanning trees. They showed superior performance to original TRW algorithm and empirically that their approximations are often of reasonable quality.

Among these approximation, TRW has achieved appealing performance, especially in that they produce more accurate pseudomarginals. However TRW only yields an upper bound of the log partition function and in general this bound has larger error than Bethe approximation. For this reason, we keep our focus on directly minimizing the Bethe free energy. UPS algorithm has gained popularity due to its reasonably good performance and easiness to implement [Carbonetto et al., 2004; Xie et al., 2009]. In this chapter, we give a convergence analysis of a convex relaxation method, which is a natural generalization of UPS algorithm. Although it was stated that UPS algorithm is convergent by Teh and Welling [2001b], there is no proof given and the convergence result does not follow from the preceding contents therein either. The proofs given in Teh and Welling [2001a] and in Welling and Teh [2001] lack key components that guarantee global convergence. We give an alternative but complete proof for a weaker sufficient condition for the convergence, from a mathematical programming point of view.

While it has been demonstrated empirically that loopy and generalized BP work

extremely well in many applications, such as Turbo decoding, observation of divergence has been documented as well [Botetz, 2007]. Yedidia et al. [2000, 2005] have shown that these methods are not guaranteed to converge for graphs with cycles. Approximate inference methods can be understood as nonconvex constrained minimization of the so-called Bethe free energy. They have also shown that fixed points of these iterative methods are stationary points of Bethe free energy and these methods are guaranteed to converge to some fixed point for singly-connected graphs. Our aim is to find a convergent alternative of loopy BP that achieves same accuracy as loopy BP and runs within a similar time bound of loopy BP. UPS algorithm, developed by Teh and Welling [2001b], is such an alternative. They have demonstrated that UPS algorithm works at least as well as loopy BP in practice. The algorithm iterates through a sequence of strictly convex subproblems. Each subproblem is solved by Iterative Scaling (IS) algorithm, which is exact and fast. UPS algorithm can be generalized by allowing any exact algorithm, including IS, to be used in solving the subproblems. We refer to this generalization as the convex relaxation method.

The convex relaxation method can be viewed as a block coordinate descent (BCD) method for constrained optimization [Heskes, 2006; Teh and Welling, 2001b]. It is known that in general, BCD methods may not converge to any stationary point of the problem [Bertsekas, 1999]. In a well-known example [Powell, 1973], the sequence of iterates cycles through several nonstationary points in the limit. The reason for such a behavior is that at each iteration the gradient of the objective function becomes progressively closer to being orthogonal to the coordinate search direction [Bertsekas, 1999; Nocedal and Wright, 1999]. In addition, coordinate blocks for the convex relaxation method overlap with each other. In contrast to BCD methods with orthogonal blocks, such methods have no general convergence results. Any particular method of such type needs to be treated specifically. An example in statistics can be found in the work by Drton and Eichler [2006], where an overlapping coordinate search is used in maximum likelihood estimation and a specific proof for the convergence of

the algorithm is given.

Under some convexity assumption of the problem, one can ensure the convergence of the block-nonlinear Gauss-Seidel (GS) method, where coordinate blocks are orthogonal to each other, in both constrained and unconstrained case [Bertsekas, 1999; Grippo and Sciandrone, 2000; Luo and Tseng, 1992]. Consider the following problem

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad x \in X = X_1 \times \cdots \times X_m \subseteq \mathbb{R}^n, \end{aligned} \tag{4.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and the feasible set X is the Cartesian product of closed, nonempty and convex subsets $X_i \subseteq \mathbb{R}^{n_i}$, for $i = 1, \ldots, m$, and $\sum_{i=1}^{m} n_i = n$. Let $x \in \mathbb{R}^n$ be partitioned into m components with $x_i \in \mathbb{R}^{n_i}$. A naturally defined block GS method for (4.1) has the following update rule:

$$x_i^{k+1} = \arg\min_{y_i \in X_i} f(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, y_i, x_{i+1}^{k}, \ldots, x_m^{k}).$$

The results in Bertsekas [1999] and in Grippo and Sciandrone [2000] both state that for (4.1), any cluster point of the sequence $\{x^k\}$ generated by the block GS method is a stationary point of (4.1), under the assumption that $\{x^k\}$ has cluster point and $f(x)$ is component-wise strictly convex. This result is of particular interests to us because in the convex relaxation method, the Bethe free energy function is component-wise strictly convex with respect to each coordinate block. However the applicability of this result is compromised by the fact that the coordinate blocks in the convex relaxation method overlap with each other.

We first prove some linear-algebraic properties of the constraint set. Together with a mild assumption that all clique potentials are positive, these properties help us obtain desired convergence results even in the presence of the overlapping of the coordinate blocks. The rest of the chapter is organized as follows. In Section 2 and Section 3, we present the convex relaxation method and some techniques for subproblem construction, respectively. In Section 4, we give a mathematical programming formulation of the method and prove some properties of the constraint set. In Section

5, we prove the major convergence results. Section 6 shows experimental results on Ising grid models and large scale *ibd* graphs. Some discussion is given in Section 7.

4.2 The Convex Relaxation Method

The definitions and notions associated with Bethe approximation can be found in Chapter 3.

As a well-known result [Heskes et al., 2003; Pakzad and Anantharam, 2002, 2005], the following theorem is a key element in the development of the UPS algorithm. It also plays an important role in the convergence analysis of the convex relaxation method.

Theorem 4.2.1. *If each connected component of a Bethe region graph $B(\mathcal{O}, \mathcal{I}, \mathcal{E})$ is singly-connected, then the Bethe free energy function defined on this region graph is strictly convex over the constraint set.*

Let $\mathcal{I}'$ be a subset of inner regions. We can obtain a subproblem by fixing the pseudo-marginals associated with $\mathcal{I}'$ to a constant vector. We call $\mathcal{I}'$ a *condition set*. Figure 4.1 shows the messages passed between inner and outer regions. The inner region on the left is not conditioned and it sends (receives) messages to (from) the neighboring outter regions then updates its own pseudo-marginals based on the messages received. The inner region on the right is conditioned and it only sends out messages without updating its own pseudo-marginals. An alternative way to understand this is that the messages passed from the outer region to the conditioned inner region are absorbed by that inner region.

If the induced subgraph of B by $\mathcal{O} \cup \mathcal{I} \setminus \mathcal{I}'$ is a tree, then this subproblem is strictly convex. Hence, hence the unique global minimum can be easily attained by an exact algorithm, such as the one used in Teh and Welling [2001b]. The convex relaxation method (which incorporates UPS as a special case) works by first finding a sequence of such convex subproblems then repeatedly solving them until convergence.

The convex relaxation method is described in Procedure 1.

Procedure 1 The convex relaxation method

input: A region graph $B(\mathcal{O}, \mathcal{I}, \mathcal{E})$.

input: A sequence of condition sets $\{\mathcal{I}_1, \ldots, \mathcal{I}_m\}$, $\mathcal{I}_i \subset \mathcal{I}, i = 1, \ldots, m$.

output: Final value for pseudo-marginals.

Initialize the pseudo-marginals to equal probabilities.

while $\neg$ converge **do**

  for $i = 1$ to m **do**

    Run message passing with condition set $\mathcal{I}_i$.

  end for

end while

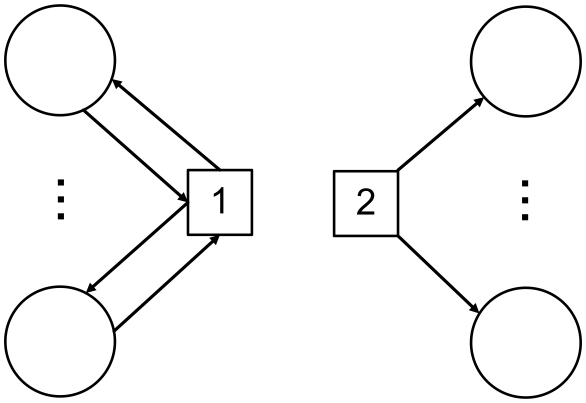


Figure 4.1: Message passing for regular (left) and conditioned (right) inner region.

4.3 Constructing Subproblems

In convex relaxation method, subproblems are defined over a sequence of tree-structured subgraphs. Simple schemes of finding these subgraphs in grid graphs are proposed in Teh and Welling [2001b]. However, these schemes are not optimal and cannot be extended to general graphs. We present a hypergraph spanning tree algorithm that

is more effective and is applicable to general graphs. With the hypergraph representation, the problem of finding these subgraphs, which otherwise requires ad hoc treatment in bipartite region graphs, becomes well-defined.

Definition A hypergraph is a pair $H(V, E)$, where V is the set of nodes and E is the set of hyperedges each of which corresponds to a subset of nodes. A path is a sequence $v_0, e_0, \cdots, v_i, e_i, \cdots, e_{k-1}, v_k$, such that $v_i \in V$, $e_i \in E$, and $v_i \in e_i$, $v_i \in e_{i-1}$.

Definition A hypergraph $H(V, E)$ is a hypertree if for every $u, v \in V$, there is a unique path from u to v in H. A hypergraph $H'(V, E')$ is a spanning tree of $H(V, E)$ if $E' \subseteq E$ and H' is a hypertree. A hyperforest is a disjoint union of a set of hypertrees.

In the hypergraph representation, nodes and hyperedges correspond to outer regions and inner regions, respectively. Specifically, an inner region can be regarded as a set, whose elements are adjacent outer regions. In convex relaxation method, every outer region appears in all subproblem. The sequence of tree-structured subgraphs corresponds to a sequence of spanning hypertrees. In general, a spanning tree in a hypergraph may not exist and even determination of its existence is strongly NP-complete [Tomescu and Zimand, 1994]. We develop a heuristic, called *hyperspan*, by extending Kruskal's minimum spanning tree algorithm for ordinary graphs. Here we are only looking for a spanning tree so hyperedges are picked from a randomly ordered list of hypereges. The algorithm works by maintaining a set of disjoint hypertrees, initially being empty graphs (a set of nodes only). The set of trees are grown by sequentially adding hyperedges from an ordered list as long as no cycle is introduced.

Since we may frequently check whether a hyperedge is connected to a hypertree and merge two hypertrees as they become connected through a newly added hyperedge, the disjoint-set data structure and union-find algorithm are used to speedup the operation. A disjoint-set data structure is a data structure that keeps track of a set of elements partitioned into a number of disjoint (nonoverlapping) subsets. Here our disjoint-set

is the set of disconnected hypertrees, each of which is represented by a set of nodes. We use the implementation of the union-find data structure in Tarjan and Leeuwen [1984], where disjoint-set forest is used. Disjoint-set forests are a data structure where each set is represented by a tree data structure, in which each node holds a reference to its parent node. The root node is used as a representative of the set. The union-find algorithm associated with the disjoint-set data structure performs these two basic operations:

- Find: Determine which set a particular element is in. Also useful for determining if two elements are in the same set.

- Union: Combine or merge two sets into a single set.

When determining whether a node belongs to a set, we simply compare the parent of the node to the root of the set tree. Merging of two set trees can be done by appending the root of the shallower tree to the root of the other. A more detailed description of union-find data structure can be found in Cormen et al. [2009]. We map our two important tasks to the union-find algorithm:

- Checking if a new hyperedge forms a cycle: check if each of the neighboring nodes of the hyperedge belongs to a distinc set.

- Merging two hyperedges: simply use the Union operation on two hypertrees.

When checking whether an hyperedge forms a cycle to existing hyperforest, we can use a hash map to keep track of the set-membership of the neighboring nodes of the hyperedge. Originally the hash map is empty, as we loop through the neighboring nodes, we first find its set-membership by retrieving the representative node then look up it in the map. If it's not in the map, it means no cycle formed and the representative node will be put in the map. The maximum of the size of the hash

Procedure 2 Hyperspan

input: A hypergraph $H(V, E)$.
input: A randomly ordered list L containing all hyperedges in E.
input: A list C of hyperedges. Initially empty.
for each node $v \in V$ **do**
  Make a Union-Find set $S(v)$ for v.
end for
for each edge $e \in L$ **do**
  Let $N = \{v|$ adjacent to $e\}$.
  Create a hashmap M.
  for each node $v \in N$ **do**
    Find $Set(v)$ (Union-Find).
    if $Set(v) \in M$ **then**
      Loop found. Break.
    else
      Add $Set(v)$ to M.
    end if
  end for
  if no loop found **then**
    Add e to C.
    Merge all $Set(v)$ for $v \in N$ (Union-Find).
  end if
end for

map will be the degree of the hyperedge. For graphs with bounded node degree, the use of hash maps can accelerate cycle detection.

We call the above algorithm *hyperspan* and its pseudocode is shown in Procedure 2. We write $s(V, E') = hyperspan(H, L)$ to mean that a spanning forest s of

Procedure 3 Greedy Sequencing

input: A hypergraph $H(V, E)$.

input: An ordered list L containing all hyperedges in E.

input: A list C of hyperedges. Initially empty.

output: A sequence S of spanning hypertrees or hyperforests.

$s(V, E') = hyperspan(H, L)$. Add s to S.

Add $E \setminus E'$ to C.

while C is not empty **do**

  $L' =$ concatenation of E' to C.

  $s(V, E') = hyperspan(H, L')$. Add s to S.

  $C \leftarrow C \cap E'$.

end while

H is obtained by applying our on hypergraph H, with H's hyperedges in list L. We apply the above heuristic repeatedly to obtain a sequence of spanning hyperforests. In this context, the convergence criterion in Su [2010] translates to a condition that every hyperedge has to appear in at least one spanning forest. The Greedy Sequencing procedure guarantees that, in the worst case, this convergence criterion is still satisfied.

A two-dimensional $m \times n$ grid graph is defined as the graph whose nodes correspond to the points in the plane with integer coordinates, x-coordinates being in the range $1, ..., n$, y-coordinates being in the range $1, ..., m$, and each node (except for those at the boundary) connects to 4 neighboring nodes with distance 1. Interestingly, the greedy sequencing algorithm is optimal for two-dimensional grid graphs as is shown by the following proposition:

Proposition 4.3.1. *For a grid graph model with arbitrary size, the greedy sequencing procedure returns a sequence of size two.*

Proof. We apply *hyperspan* using an ordered list of hyperedges of the hypergraph. After the first run of *hyperspan*, we obtain a set of hyperedges that are not in the resulting spanning forest. Denote these hyperedges as nonforest edges. Then we apply *hyperspan* again with a new list in which all these nonforest edges appear before other edges, and so will be considered in the new spanning forest before other edges. If these nonforest edges don't form a cycle themselves, they will all appear in the new spanning forest, and two runs of *hyperspan* suffice. Now, for a grid graph, in the first run of *hyperspan*, suppose the hyperedges are traversed in a row-first manner. That is, all nodes of a row are traversed consecutively and then another row is considered. Then the obtained nonforest edges are disconnected, thus not forming a cycle. Alternatively, the same argument applies if a column-first order is considered. □

4.4 Mathematical Programming Formulation

Now we formally state the Bethe free energy minimization problem [Heskes, 2006; Yedidia et al., 2000, 2005]:

$$\begin{aligned} \text{minimize} \quad & F_b(q) \\ \text{subject to} \quad & q_\gamma(X_\gamma) \geq 0 \quad \forall \gamma \in \mathcal{R} \; \forall X_\gamma \in S_\gamma \\ & \textstyle\sum_{X_\gamma} q_\gamma(X_\gamma) = 1 \quad \forall \gamma \in \mathcal{R} \\ & \textstyle\sum_{X_{\alpha \setminus \beta}} q_\alpha(X_\alpha) = q_\beta(X_\beta) \\ & \quad \forall \alpha \in \mathcal{O}, \beta \in \mathcal{I}, \alpha \supset \beta, X_\beta \in S_\beta. \end{aligned} \tag{4.2}$$

The second and third set of constraints in (4.2) are known as normalization and consistency constraints, respectively. Note that with consistency constraints, we only need the normalization constraints for either outer regions or inner regions, but not both. Indeed, we can eliminate variables which are pseudo-marginals of all inner regions because they can be obtained by marginalizing the pseudo-marginals of their including outer regions. Consistency constraints need to be written in terms of pseudo-

marginals of every pair of intersecting outer regions. This alternative formulation is given below:

$$\begin{aligned} \text{minimize} \quad & F_b(q) \\ \text{subject to} \quad & q_\alpha(X_\alpha) \geq 0 \ \ \forall \alpha \in \mathcal{O} \ \forall X_\alpha \in S_\alpha \\ & \textstyle\sum_{X_\alpha} q_\alpha(X_\alpha) = 1 \ \ \forall \alpha \in \mathcal{O} \\ & \textstyle\sum_{X_{\alpha \setminus \beta}} q_\alpha(X_\alpha) = \sum_{X_{\alpha' \setminus \beta}} q_{\alpha'}(X_{\alpha'}) \\ & \quad \forall \alpha, \alpha' \in \mathcal{O}, \alpha \cap \alpha' = \beta, X_\beta \in S_\beta. \end{aligned} \tag{4.3}$$

Note here F_b is a function of pseudo-marginals of only outer regions (the pseudo-marginals for inner regions are substituted by those of outer regions). Let us denote the feasible set of (4.3) by Q. It can be described by $Q = \{q | Aq = d, q \geq 0\}$, where A is the appropriate coefficient matrix and d is the appropriate right hand side vector. Let S_v be the sample space of random variable X_v with $v \in V$. Without loss of generality we assume that $|S_{v_1}| = |S_{v_2}| = s$, $\forall v_1, v_2 \in V$. Also assume that each inner region contains only one node. This is always possible because we can choose outer regions to be pairs of adjacent nodes, which is usually the approach taken in Ising models [Pelizzola, 2005]. These two assumptions will greatly simplify the presentation of next two propositions. The rank of matrix A is determined by next proposition.

Proposition 4.4.1. *Rank* $A = |\mathcal{O}| + (s-1) \times \sum_{\beta \in \mathcal{I}} |c_\beta|$.

Proof. Let A_n denote the submatrix associated with the normalization constraints. Since each normalization constraint contains a set of variables that are absent in other normalization constraints, the rank of A_n is just the number of normalization constraints, i.e., the number of outer regions $|\mathcal{O}|$.

For a particular $X_\beta \in S_\beta$, we have

$$\begin{aligned} \sum_{X_{\alpha_1 \setminus \beta}} q_{\alpha_1}(X_{\alpha_1}) = \cdots = & \sum_{X_{\alpha_{deg(\beta)} \setminus \beta}} q_{\alpha_{deg(\beta)}}(X_{\alpha_{deg(\beta)}}) \\ & = q_\beta(X_\beta), \end{aligned} \tag{4.4}$$

where $\alpha_1, \ldots, \alpha_{deg(\beta)}$ are the outer regions adjacent to β. Considering an equality between two outer regions as an edge connecting them, the set of equalities representing (4.4) corresponds to a set of edges connecting these outer regions. So the minimum number of such edges is the number of outer regions minus 1, i.e., $\deg(\beta) - 1 = |c_\beta|$. Consider all $X_\beta \in S_\beta$, the number of constraints is $s \times |c_\beta|$. Note we can use normalization constraints to remove the equalities for one particular X_β. So overall we have $(s-1) \times |c_\beta|$ consistency constraints for β. By induction, we can show that consistency constraints for different inner regions are independent. Thus the total number of independent equality constraints is $|\mathcal{O}| + (s-1) \times \sum_{\beta \in \mathcal{I}} |c_\beta|$. □

Consider fixing the pseudo-marginals associated with a subset of inner regions, denoted by $\mathcal{I}'$, to decrease the dimension of the feasible set. This is equivalent to adding equality constraints to (4.3). The resulting feasible set can be described by

$$\{q | \begin{bmatrix} A \\ C \end{bmatrix} q = \begin{bmatrix} d \\ f \end{bmatrix}, q \geq 0\},$$

where C is the appropriate coefficient matrix and f is the appropriate right hand side vector, for the added equality constraints. The rank of the new constraint matrix is determined by the next proposition.

Proposition 4.4.2.

$$Rank \begin{bmatrix} A \\ C \end{bmatrix} = rank\ A + (s-1) \times |\mathcal{I}'|.$$

Proof. For a particular $X_\beta \in S_\beta$, we add one equality constraint: $\sum_{X_{\alpha \setminus \beta}} q_\alpha(X_\alpha) = q_\beta(X_\beta)$, where α is some adjacent outer region of β and $q_\beta(X_\beta)$ is a fixed pseudo-marginal. Again with normalization constraints, the number of added equalities for β is $s-1$. These equalities are linearly independent. The added constraints between different inner region are also linearly independent. If the added constraints are independent of the original constraints, then the total number of independent constraints is $rank\ A + (s-1) \times |\mathcal{I}'|$.

Now we show that the added constraints are independent of the original constraints. First it can be seen that the added constraints are independent of the normalization constraints. For an inner region β, we only add $s-1$ constraints, where s is the size of the sample space of X_β. So these constraints do not imply the normalization constraint for β. Second we show that the added constraints are independent of the consistency constraints. Again for an inner region β, the added constraints force the pseudomarginals of only one adjacent outer region to sum to fixed values. This does not imply the consistency between the pseudomarginals of any pair of adjacent outer regions. □

Let $\mathcal{I}'$ be a subset of inner regions. We can obtain a subproblem of (4.2) by fixing the pseudo-marginals associated with $\mathcal{I}'$ to a constant vector p. Denote this subproblem by $\mathcal{P}(\mathcal{I}', p)$. If the induced subgraph of B by $\mathcal{O} \cup \mathcal{I} \setminus \mathcal{I}'$ is singly-connected, then by Theorem 4.2.1, $\mathcal{P}(\mathcal{I}', p)$ is convex. In such a case, we call $\mathcal{I}'$ a *condition set*.

A formal description of the convex relaxation method is given in Procedure 4.

Procedure 4 The mathematical programming formulation of the convex relaxation method

input: A sequence of condition sets $\{\mathcal{I}_1, \ldots, \mathcal{I}_m\}$.
input: q – initial value for pseudo-marginals.
output: q – final value for pseudo-marginals.
while ¬ converge **do**
  for $i = 1$ to m **do**
    $p \leftarrow q_{\mathcal{I}_i}$.
    Solve the subproblem $\mathcal{P}(\mathcal{I}_i, p)$.
    $q \leftarrow$ solution of $\mathcal{P}(\mathcal{I}_i, p)$.
  end for
end while

In Procedure 4, the objective function of each subproblem is strictly convex over

a reduced feasible set (as a result of fixing some variables). Thus the unique global minimum can be easily attained by an exact algorithm, such as the one used in Teh and Welling [2001b]. The subproblems interact with each other through shared pseudo-marginal variables. Procedure 4 is considered as converged when the value of all pseudo-marginal variables remain unchanged after any consecutive execution of all subproblems, i.e., one round of the **while** loop.

Formulation (4.3) gives us insight to some linear-algebraic properties of the feasible sets. For next two propositions we will focus on formulation (4.3). At each iteration, we reduce the original feasible set to a lower dimensional convex polyhedral set by adding a set of equality constraints. Let $\mathcal{S}$ be the smallest affine subspace in which the original feasible set lies. Also let $\mathcal{S}_i$ be the smallest affine subspace in which the feasible set of the i-th subproblem lies. Let C_i denote the coefficient matrix for the equality constraints added in the i-th subproblem. The following proposition states a necessary and sufficient condition for $\{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$ spanning $\mathcal{S}$.

Proposition 4.4.3. *The affine subspace $\mathcal{S}$ is spanned by $\{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$ if and only if $\cap_{i=1}^{m} \mathcal{I}_i = \emptyset$.*

Proof. By the Inclusion-Exclusion principle, the necessary and sufficient condition for $\{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$ spanning $\mathcal{S}$ is:

$$\begin{aligned} \text{rank } A = \sum_{i=1}^{m} \text{rank} \begin{bmatrix} A \\ C_i \end{bmatrix} - \sum_{i,j:1\le i<j\le m} \text{rank} \begin{bmatrix} A \\ C_i \\ C_j \end{bmatrix} \\ + \sum_{i,j,k:1\le i<j<k\le m} \text{rank} \begin{bmatrix} A \\ C_i \\ C_j \\ C_k \end{bmatrix} - \cdots + (-1)^{m-1} \text{rank} \begin{bmatrix} A \\ C_1 \\ \vdots \\ C_m \end{bmatrix}. \end{aligned} \tag{4.5}$$

Invoking Proposition 4.4.2, we can show (4.5) is equivalent to:

$$\sum_{i=1}^{m} |\mathcal{I}_i| - \sum_{\substack{i,j: \\ 1\le i<j\le m}} |\mathcal{I}_i \cup \mathcal{I}_j| + \sum_{\substack{i,j,k: \\ 1\le i<j<k\le m}} |\mathcal{I}_i \cup \mathcal{I}_j \cup I_k| - \cdots + (-1)^{m-1} |\mathcal{I}_1 \cup \cdots \cup \mathcal{I}_m| = 0, \tag{4.6}$$

where we use the binomial identity

$$\sum_{k=1}^{m} (-1)^{k-1} \binom{m}{k} = 1$$

to eliminate rank A from both sides of (4.5).

By a corollary of the Inclusion-Exclusion principle, (4.6) is true if and only if $\cap_{i=1}^{m} \mathcal{I}_i = \emptyset$. □

Now we make an assumption about the positivity of the clique potentials. This assumption is essential to the proof of our convergence results. This assumption satisfies the condition of Proposition 5 in Yedidia et al. [2005], which will be used to prove next proposition.

Assumption 4.4.4. *All clique potentials are strictly positive, i.e.,* $\psi_\alpha(x_\alpha) > 0 \ \ \forall \alpha \in \mathcal{C}, x_\alpha \in S_\alpha$.

Proposition 4.4.5. *If all clique potentials are strictly positive, then all local minima of the constrained Bethe free energy are interior minima ([Yedidia et al., 2005]).*

Let $\mathcal{I}_i$ denote the subset of inner regions that are conditioned on at the i-th iteration of the inner **for** loop of Procedure 4. First let us define a stationary point of $f(x)$ to be a point x_s where the first derivative $f'(x) = 0$ evaluated at x_s. The next proposition gives a sufficient condition for the equivalence of fixed points of Procedure 4 and stationary points of the free energy minimization problem.

Proposition 4.4.6. *Under Assumption 4.4.4, any fixed point of Procedure 4 is a stationary point of the constrained minimization problem if* $\cap_{i=1}^{m} \mathcal{I}_i = \emptyset$.

Proof. Suppose the algorithm converges to a fixed point $\bar{q}$, then we have for $i = 1, \ldots, m$, $\bar{q}_i$ is the unique global minimum of the subproblem at the $i-th$ inner iteration. Adapting the proof for Prop. 4.4.5, we can show that under Assumption 4.4.4 the unique global minimum of each subproblem of Procedure 4 lies in the relative interior of that subproblem's constraint set. Indeed, if we initialize the pseudo-marginals and messages to be positive, then they will remain positive during the execution of algorithm. This means for $i = 1, \ldots, m$, $\nabla F_b(\bar{q})$ is orthogonal to the smallest affine subspace in which the i-th subproblem's feasible set lies. If these affine subspaces span the smallest affine subspace in which the original feasible set lies, then $\nabla F_b(\bar{q})$ is orthogonal to the original feasible set, ensuring that $\bar{q}$ is a stationary point. Then the claim follows from the result in Proposition 4.4.3. □

The feasible set of (4.3) is a nonempty, compact polyhedral set. In addition, Assumption 4.4.4 ensures that the energy function stays finite because logarithm functions will not be evaluated at 0. Thus there exists at least one stationary point because at least the global minimum is attained. Since the feasible set is compact, according to Bolzano-Weierstrass theorem, the sequence of iterates generated by any algorithm will have at least one cluster point. If it can be proved that every cluster point is stationary, then the algorithm is guaranteed to converge.

4.5 Convergence Analysis

In this section, we prove the convergence result for the convex relaxation method. The result presented here is similar to those in Bertsekas [1999] and [Grippo and Sciandrone, 2000] but the overlapping nature of the coordinate blocks prevents an simple extension from these results. The results in Bertsekas [1999] and in Grippo and Sciandrone [2000] state that for a problem with C^1 objective function and feasible set being a Cartesian product of m orthogonal nonempty, closed convex sets, any cluster point of the sequence generated by the block GS method is a stationary point, under

the assumption that the sequence admits cluster points and the objective function is component-wise strictly quasiconvex with respect to all components.

Let us use $k \in \mathbb{Z}^+$ to index the iterations of the outer **while** loop of Procedure 4. Also we use $i \in \{0, \ldots, m\}$ to index the iterations of the inner **for** loop. Let $w(k,i)$ be the vector of pseudo-marginals obtained at the end of i-th iteration of the inner loop in the k-th iteration of the outer loop. Let $Q_i \equiv Q_i(w(k, i-1))$ be the feasible set of the i-th subproblem given the current iterate $w(k, i-1)$. $w(k,i)$ is defined recursively as follows:

$$\begin{aligned} &w(k,0) = q^k, \\ &w(k,i) = (w_{\mathcal{I}_i}(k, i-1), \arg\min_{p_{\mathcal{R}_i} \in Q_i} F_b(p_{\mathcal{R}_i})) \\ &\quad \forall i = 1, \ldots, m, \\ &w(k+1,0) = w(k,m) = q^{k+1}, \end{aligned}$$

where we use the subset of regions to index the pseudo-marginal vector p and w. In our notation, we assume that the operator $(\cdot)$ rearranges the components of its argument according to the original order in q.

We use the following proposition to show the convergence of the objective function along a converging subsequence of $\{w(k,i)\}$.

Proposition 4.5.1. *If some sequence $\{w(k,i)\}$, $i \in \{0, \ldots, m\}$, admits a cluster point $\bar{w}$, then for every $j \in \{0, \ldots, m\}$, we have*

$$\lim_{k \to \infty} F_b(w(k,j)) = F_b(\bar{w}).$$

Proof. Let $\{w(k,i)\}_{k \in \mathcal{K}}$ be the subsequence of $\{w(k,i)\}$ converging to $\bar{w}$. Since F_b is continuous, $\{F_b(w(k,i))\}_{k \in \mathcal{K}}$ converges to $F_b(\bar{w})$. Also because F_b is non-increasing, so the whole sequence $\{F_b(w(k,i))\}$ converges to $F_b(\bar{w})$. We also have

$$\begin{aligned} F_b(w(k+1,i)) \leq F_b(w(k,j)) \leq F_b(w(k,i)) \\ \forall j \in \{i+1, \ldots, m\} \end{aligned} \tag{4.7}$$

and

$$F_b(w(k,i)) \leq F_b(w(k,j)) \leq F_b(w(k-1,i)) \quad \forall j \in \{0,\ldots,i-1\}. \tag{4.8}$$

Evaluating limits on both sides of the inequalities (4.7) and (4.8) yields

$$\lim_{k\to\infty} F_b(w(k,j)) = F_b(\bar{w}), \forall j \in \{0,\ldots,m\}.$$

□

Now we show the last proposition before our final convergence result.

Proposition 4.5.2. *Let $\{q^k\}$ be a sequence of points in Q converging to $\bar{q}$. For a particular $i \in \{1,\cdots,m\}$, let $\{p^k\}$ be a sequence of vectors whose components are defined as follows:*

$$p_j^k = \begin{cases} q_j^k & \text{if } j \neq i, \\ \arg\min_{u\in Q_i} F_b(u, q_{\mathcal{I}_i}^k) & \text{if } j = i. \end{cases}$$

Then, if $\lim_{k\to\infty} F_b(p^k) - F_b(q^k) = 0$, we have $\lim_{k\to\infty} \|p^k - q^k\| = 0$.

Proof. We prove the proposition by contradiction. Assume that $\lim_{k\to\infty} \|p^k - q^k\| \neq 0$ and $\|p^k - q^k\| = \gamma^k$. There exists $\bar{\gamma} > 0$ and N such that $\|\gamma^k\| \geq \bar{\gamma}$ for all $k > N$. Let $s^k = \frac{p^k - q^k}{\|p^k - q^k\|}$ then $p^k = q^k + s^k\gamma^k$. $\|s^k\| = 1$ and s^k differs from zero only in the i-th block coordinate. Since s^k belongs to a compact set, it has a limit point $\bar{s}$. Assume $\{k_j\}$ be a subsequence of $\{k\}$ and s_{k_j} converge to $\bar{s}$.

Now let $\epsilon \in [0,1]$ and we have $0 \leq \epsilon\bar{\gamma} \leq \gamma^k$. So $q^k + \epsilon s^k\bar{\gamma}$ lies on the segment joining q^k and $q^k + s^k\gamma^k$ thus belongs to Q since Q is convex. Since $F_b(q_1,\ldots,q_i,\ldots,q_m)$ is strictly convex with respect to q_i, then we have

$$F_b(p^k) \leq F_b(q^k + \epsilon s^k\bar{\gamma}) \leq F_b(q^k).$$

Since both $F_b(p^k)$ and $F_b(q^k)$ converge to $F_b(\bar{q})$, we have

$$F_b(\bar{q} + \epsilon\bar{s}\bar{\gamma}) = F_b(\bar{q})$$

for every $\epsilon \in [0, 1]$. This contradicts the fact that $F_b(q)$ is strictly convex with respect to the i-the block coordinate. □

Now we present the convergence result.

Proposition 4.5.3. *Suppose Assumption 4.4.4 holds and the convex relaxation method (Procedure 4) satisfies $\cap_{i=1}^{m} \mathcal{I}_i = \emptyset$, the sequence of iterates generated by the method converges in the sense that*

1. *It has cluster points.*

2. *Any cluster point is a stationary point of the Bethe free energy minimization problem.*

Proof. Let $\bar{q}$ be any cluster point of $\{q^k\}$ and $\{q^k\}_{k \in K}$ be the subsequence converging to $\bar{q}$. Note that $q^k = w(k, 0) = w(k-1, m)$. By Proposition 4.5.1, we have $\lim_{k \to \infty} F_b(w(k, 1)) = F_b(\bar{q})$. Using Proposition 4.5.2 for $i = 1$, identifying $\{p^k\}$ with $\{w(k, 1)\}$, we can show that $\{w(k, 1)\}_K$ converges to $\bar{q}$. Sequentially applying Proposition 4.5.2 for $i = 2, \cdots, m-1$, we have $\{w(k, i)\}_K$ converges to $\bar{q}$ for $i = 2, \cdots, m$. Thus $\bar{q}$ is a fixed point and in addition a stationary point (by Proposition 4.4.6). □

Remark Proposition 4.5.3 guarantees that under Assumption 4.4.4, the execution of the convex relaxation method will not cycle through a series of non-solution points. When a cluster point is reached, we have found the solution to the problem. However, it is impractical to run a numeric algorithm for an infinite number of iterations to achieve a cluster point or fixed point. A common way is to monitor the amount of progress the algorithm is making at the end of each iteration, and terminate its execution when it starts making negligible progress. Usually changes in the function or variable values can be used as comparison criterion. For example, in the convex relaxation method, we can set a tolerance value ϵ, when $|F_b(q^k) - F_b(q^{k-1})| < \epsilon$ or $\|q^k - q^{k-1}\| < \epsilon$, we can view that as a sign of achieving a fixed point and terminate the

execution of the algorithm. If the original problem is not very badly scaled, changes less than a small tolerance, e.g., $\epsilon = 10^{-6}$ or 10^{-7}, usually indicate that a fixed point or a stationary point is achieved.

4.6 Experiments and Results

In many cases, loopy BP algorithm often converges quickly, i.e., no alternating between closely distanced solutions occurs. Loopy BP can serve as a good reference point for comparison purposes. In addition, as mentioned in Section 1.1, the convex relaxation method is developed to be a convergent alternative of loop BP algorithm. In this Chapter, we conduct experiments on both Ising grid models and large *ibd* graphs using loopy BP and convex relaxation method. We compare their convergence speed and the accuracy of approximation.

4.6.1 Ising Grid Models

We test UPS algorithm using two-dimensional Ising models of statistical physics [Baxter, 1982]. The Ising model can be represented using a Markov random field with grid graph $G(V, E)$ and joint distribution $P(x) \propto e^{\sum_{i \in V} \alpha_i x_i + \sum_{(i,j) \in E} \beta_{ij} x_i x_j}$, where V and E are nodes and edges. With each β in [0, 2], we randomly generated 40 problems where for each of which β_{ij}'s are uniformly drawn from [-β, β] and α_i's are uniformly drawn from $[-0.05, 0.05]$. Since we are comparing the results with the exact algorithm, we choose size of the test examples to be 10×10.

First we compare the accuracy of the approximate log partition using loopy BP (LBP), mean field (MF) BP [Jordan et al., 1999], TRW BP and UPS. The results are illustrated in Figure 4.2. Plotted are the normalized errors of log partition funtion vs. β. For loopy BP and USP which are expected to produce estimates rather than bounds, the normalized error is defined as $\frac{|Exact_{Log Partition} - Estimated_{Log Partition}|}{Exact_{Log Partition}}$. For

MF whose results are lower bounds, the error is defined as

$$\frac{Bound - Exact_Log_Partition}{Exact_Log_Partition}.$$

For tree reweighted BP which produces upper bound, the error is defined as

$$\frac{Bound - Exact_Log_Partition}{Exact_Log_Partition}.$$

From Figure 4.2 it can be seen that UPS always produces similar performance to loopy BP and both are the most accurate among the compared methods. This is within expectation since both methods directly minimize the Bethe free energy. The lower bound produced by the simplex mean field approximation starts as the least accurate and becomes steady as β approaches 1. On the other hand, TRW BP generates bounds that grows linearly with β.

We also compare the accuracy of pseudomarginals obtained from LBP, TRW and USP. The averaged l_1-errors of pseduomarginals are illustrated in Figure 4.3. Again the approximated pseudomarginals from loop BP and UPS have similar accuracy. Interestingly as β increases, although TRW BP produced less accurate estimates to the log partition function, the estimated pseudomarginals of TRW BP become more accurate than those of BP and UPS.

4.6.2 Large ibd *Graphs*

Through a simulated example, we also demonstrate the applicability of UPS to handle large scale models arising in modern genetic analysis. We obtain the simulated data from Brown et al. [2012] where a 200-generation population is simulated with each generation consisting of 3500 males and 3500 females. Each founder chromosome is given a unique founder genome label and descendant chromosomes are constructed by simulating meiosis events using a realistic model. Hence each chromosome can be specified as a list of DNA segments each represented by an FGL. At subsequent generations, the next generation is simulated by randomly sampling 3500 pairs of

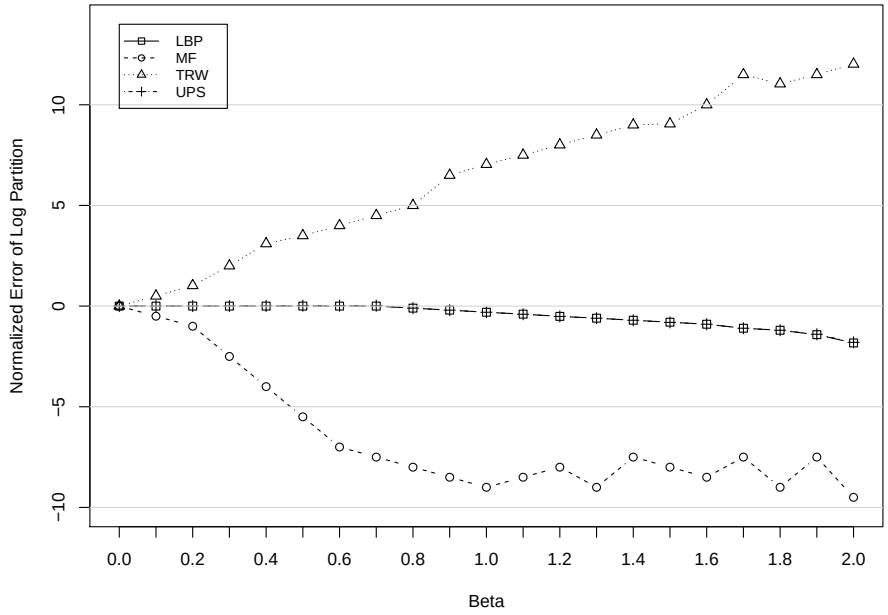


Figure 4.2: Comparing normalized approximated log partition from MF, LBP, UPS and TRW.

males and females, with each pair generating a son and a daughter. At a genomic position, the state of *ibd* (see Section 1.3.4) can be obtained by comparing the FGL segments where the FGL is the same, the chromosomes are ibd. In this example, we use individuals from the 4 most recent generations of the simulated population to construct the *ibd* graph at a randomly selected genomic location assuming all individuals are observable. The true simulated *ibd* graph is shown in Figure 4.4.

The *ibd* graph contains 361 nodes and 5370 edges. At this genomic position, only a small portion (361/14000) of founder DNA remains in the population and the *ibd* graph becomes very dense. We use the Java package **LibTW** [van Dijk et al.] to calculate the graph's treewidth. The lower bound and upper bound of the treewidth is 65 and 177, respectively. Despite the daunting density of the graph, the length of the sequence we obtained using the greedy sequencing algorithm is only 11. Using

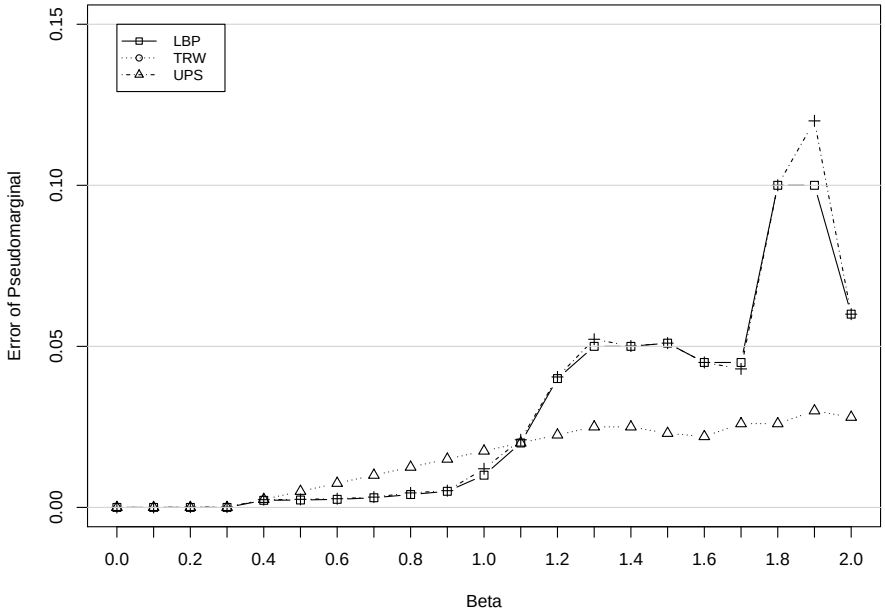


Figure 4.3: Comparing accuracy of approximated pseudomarginals from MF, LBP, UPS and TRW.

Loopy BP, UPS and TRW BP with uniformly initialized pseudomarginals all take less than 1 second and converge to the same solution. Since it is infeasible to obtain exact solution to problems with such a complexity, we are not able to assess the error of the approximation. The quick convergence of all three algorithms is due to the fact that at such a density, the Bethe free energy function has a large number of local minima such that starting from the initial solution, the algorithm will not move too far away before reaching a nearby local minimum. However this is common in multimodal function optimization and it can be improved by using more educated initial solutions.

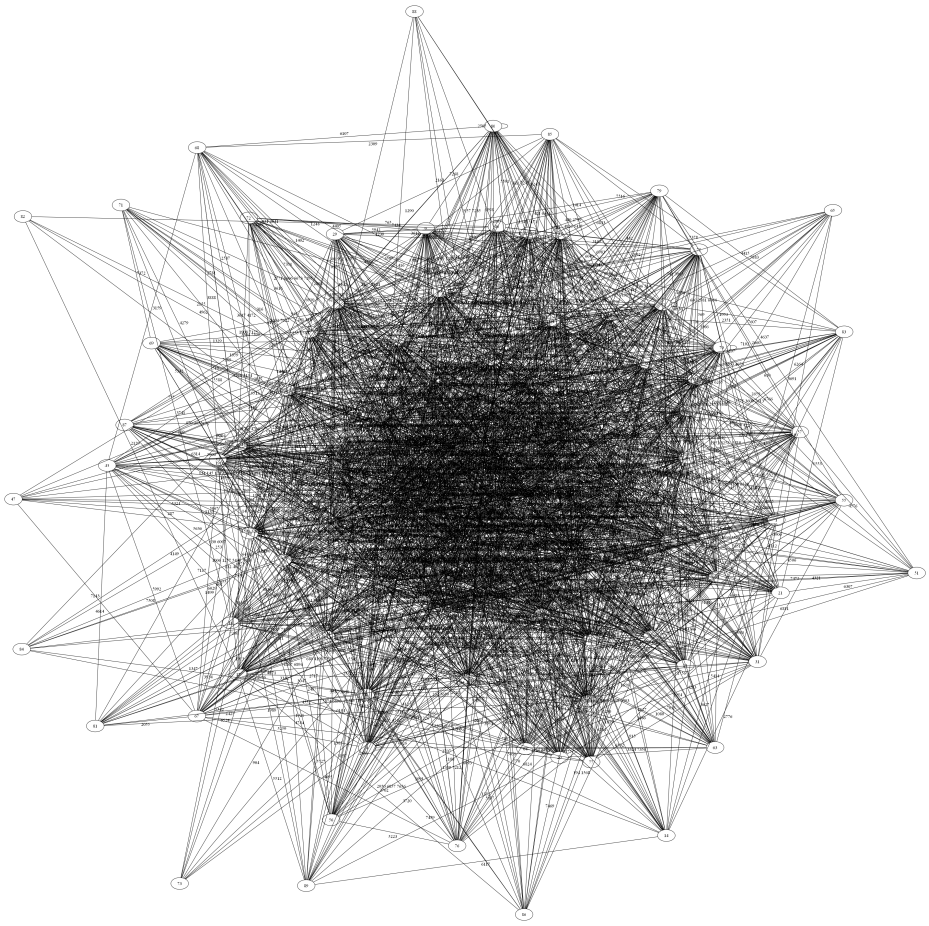


Figure 4.4: The ibd graph of the simulated example.

4.7 Discussion

We proved the sufficient condition for the convergence of the convex relaxation method. A stronger sufficient condition that eliminates the necessity of Assumption 4.4.4 is definitely more desirable because the potential functions arise in many application will have boundary conditions, i.e., $\psi_\alpha(x_\alpha) = 0 \text{or} 1 \;\; \text{for some} \alpha \in \mathcal{C}, \forall x_\alpha \in S_\alpha$. However, whether there exists such stronger sufficient condition remains unknown. This work provides insights to the analyses of the approximate methods, which directly optimize Bethe free energy or similar function. The structure of the optimization

problem, which is represented as the linear-algebraic properties of the constraint set, directly relate to the topology of the region graph. This suggests that desirable structures may exist in methods that build subproblems on a collection of subgraphs. These structures may be exploited to improve the efficiency of the algorithm.

We also proposed a heuristic for subproblem construction. This heuristic has been shown to be effective in general and is provably optimal for grid graphs. Thorough testing on a complete set of benchmarking networks will be important in evaluating the performance of the heuristic.

From the mathematical programming point of view, the convex relaxation method is a generalization of the constrained block-nonlinear GS method. Now we discuss whether our specific proof for the convex relaxation method provides a paradigm for attacking the convergence analysis of such constrained block-nonlinear GS algorithms arising from other applications. Our proof of the convex relaxation method consists of two parts. The first one is adapted from the proof of the block-nonlinear GS method, which basically requires that for each subproblem, a unique global minimum is attained. In many cases, this requirement is naturally satisfied because subproblems are constructed in such a way. Usually subproblems are convex so can be easily solved even analytically. The second part of the proof is trying to establish a relationship between the solution to the subproblems and the solution of the original problem. this relies on an important fact stated in the proof for Proposition 5 in Yedidia et al. [2005], i.e., the solution to the subproblems lies in the relative interior of the constraint sets. This might be hard to satisfy in general and its fufillment may require particular selection of the subproblems.

Chapter 5

PARALLELIZING CONVEX RELAXATION METHOD

The ability to perform probabilistic inference tasks efficiently is critical in large scale applications. Exact inference methods have been parallelized but they still suffer from worst-case exponential complexity. Message passing-based approximate inference methods have been parallelized and good speedup achieved. In this Chapter, we focus on the parallelization of the convex relaxation method proposed in Chapter 4. This BP variant has better convergence properties and is provably convergent under certain condition. Moreover it can be shown that this method exhibits embarrassing parallelism and is amenable to coarse-grained parallelization. We propose techniques to parallelize it optimally without sacrificing convergence. Experiments on a shared memory system demonstrate that near-ideal speedup is achieved with reasonable scalability. We also discuss issues arise from extending the proposed method to distributed systems.

5.1 Introduction

The ever-evolving parallel computing technology suggests that dramatic speedup might be achieved by appropriately mapping the existing sequential inference algorithms to the parallel framework. Exact inference methods, such as variable elimination (VE) and the junction tree algorithm, have been parallelized and reasonable speedup achieved. Parallel VE was proposed based on the idea of cutset conditioning [Shachter and Andersen, 1994; Silberstein et al., 2006], where assignment of the cutset variables leads to independent computations that can be done concurrently. Using the method of *pointer jumping*, Pennock [1998] proposed a parallel VE with running

time being logarithm in the number of nodes. Parallel junction tree algorithms were proposed by Kozlov and Singh [1994]; Namasivayam et al. [2006]; Xia and Prasanna [2008]. In these algorithms, both in-clique and topological parallelism [Kozlov and Singh, 1994] were extracted.

However, exact inference can be computationally intractable, indeed, NP-hard [Cooper, 1990]. The complexity of exact inference methods for a graphical model is exponential in the tree-width of the graph. For graphs with large tree-width, approximate methods are necessary. Recently a promising parallel approximate inference method was presented by Gonzalez et al. [2009a,b], where loopy BP was parallelized, but without ensuring convergence. While it has been demonstrated empirically that loopy and generalized BP work extremely well in many applications, such as Turbo decoding, divergence has also been documented by Botetz [2007]; Yedidia et al. [2000, 2005] have shown that these methods are not guaranteed to converge for loopy graphs. The convergence properties of such iterative methods have significant impact on both theory and application. The UPS algorithm [Teh and Welling, 2001b] has gained popularity due to its reasonably good performance and ease of implementation [Carbonetto et al., 2004; Xie et al., 2009]. More important, as we have shown in Chapter 4, UPS is guaranteed to converge under mild conditions.

In this chapter, we develop an effective parallel generalized inference method with special attention to UPS algorithm. Even though the generalized inference method possesses a structural parallelism that is straightforward to extract, problems of imbalanced load and excessive communication overhead can result from ineffective task partitioning and sequencing. We focus on solving these two problems and demonstrate the performance of efficiently paralleled algorithms on large scale problems using a shared memory system.

The rest of the chapter is organized as follows. In Section 5.2, we present the basic idea of extracting coarse grained parallelism from a sequential UPS algorithm. Then we propose techniques for task partitioning and load balancing in Section 5.3. In

section 5.4 and Section 5.5, we describe the computing environment and the software implementation, respectively. Experimental setup is given in Section 5.6 and some important run-time techniques are developed in Section 5.7.1. Results are shown in Section 5.8. Some thoughts about reducing communication overhead in a distributed memory system are presented in Section 5.9. Section 5.10 provides a summary discussion.

5.2 MapReduce: a coarse-grained parallelism of a hyperforest

Recently a promising parallel approximate inference method was presented in Gonzalez et al. [2009a,b], where loopy BP was optimally parallelized, but without guarantee of convergence. Here we propose a parallel convex relaxation method as a convergent alternative to parallel loopy BP.

In the greedy sequencing procedure introduced in Chapter 4 Section 4.3, if a subproblem is defined on a hyperforest rather than a hypertree, we can run Iterative Scaling (IS) on disconnected components of the hyperforest, independently. These tasks can be performed in parallel in a system with multiple processing units. This is essentially embarrassing parallelisim and it suggests a natural way of extracting coarse-grained parallelism uniformly across the sequence of subproble ms. The basic idea is, for each subproblem, to partition the hypertree into a prescribed number t of components and assign the computation associated with each component to a separate processing unit.

We demonstrate this parallelization scheme through the example in Figure 5.1. Note that here we only illustrate the idea of partitioning and the two resulting components are not tree. They can be turned into trees by further conditioning on more inner regions, e.g., 10 and 7 in Figure 5.1(a), respectively. At the first iteration, we have two disconnected graphs as shown in Figure 5.1(a). Processor P_1 holds the graph at the lower left corner and processor P_2 holds the one at the upper right corner. P_1 contains data for inner regions $\{5, 9, 10, 13, 14, 15\}$ while P_2 contains data

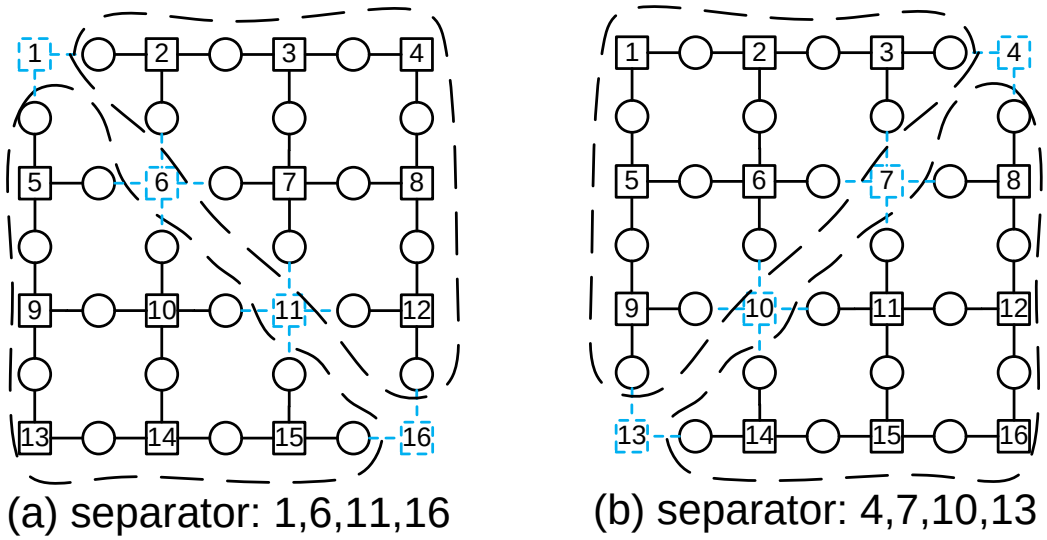


Figure 5.1: 4×4 grid graph. Boxes: outer regions. Cycles: inner regions. The separators are drawn in dashed lines.

for $\{2, 3, 4, 7, 8, 12\}$. Both processors have the separator $\{1, 6, 11, 16\}$ since the two subproblems depend on them. At next iteration, we have two different graphs as shown in Figure 5.1(b). P_1 holds the graph at the upper left corner and P_2 holds the one at the lower right corner. P_1 has inner regions $\{1, 2, 3, 5, 6, 9\}$ while P_2 has $\{8, 11, 12, 14, 15, 16\}$. Both processors need to contain the separator $\{4, 7, 10, 13\}$. At every iteration a processor needs data that are updated in other processors in previous iterations. For example, at the second iteration, P_1 needs data for inner regions 2, 3, and their adjacent outer regions. These data needs to be sent from P_2 to P_1 because they are updated in P_2 at the first iteration.

Note that synchronization is necessary at the end of each inner iteration, which can be accomplished by software barriers. During synchronization, data communication may or may not be incurred. If the underlying system is shared memory, there will be no communication cost between two consecutive inner iterations because all processing units are seeing the same copy of the data. In distributed memory systems, communication overhead is incurred (Section 5.9). This independent processing

and synchronization scheme maps to a coarse-grained **MapReduce** framework [Dean and Ghemawat, 2004]. Figure 5.2 shows the convex relaxation method flowchart in MapReduce framework. There are m inner iterations, each consists of a map and a reduce operation. $T_1, \ldots, T_t$ represent independent computation associated with t disconnected components. The **Map** operation is the procedure that each processing unit computing on its own input data obtained either from the initial input or from the previous iteration. Once all tasks reach the synchronization point (barrier), there will be a simple synchronization (shared memory system) or a data transmission between processing units (distributed system). This is known as the **Reduce** operation, where multiple independent tasks reduce to a central control thread. In case of distributed memory system, the data are also reduced from an inconsistent state where different copies of the same variable contains different value to a consistent state across all processing units.

In the next section, we present techniques for effective solving of the critical issues for parallelizing sequential applications: task partition and load balance.

5.3 Task Partitioning and Load Balancing

To partition the computation task for a subproblem, we need to partition the hypertree or hyperforest associated with the subproblem into a given number of disconnected components. Usually this number equals the the number of available processing units. There are existing methods that can be used for this task. Recall the convergence criterion (Proposition 4.4.6) states that every hyperedge has to appear in at least one spanning hypertree. This remains necessary for the partitioned hypergraph, which means no hyperedge is always allowed to be a cut edge. To prevent a hyperedge from being selected as a cut edge, we can apply a simple technique, called *edge contraction*. When a hyperedge is contracted, it is replaced by a super node, containing this edge and all nodes that are adjacent to this edge. All other edges that are previously adjacent to any of these nodes become adjacent to the super node.

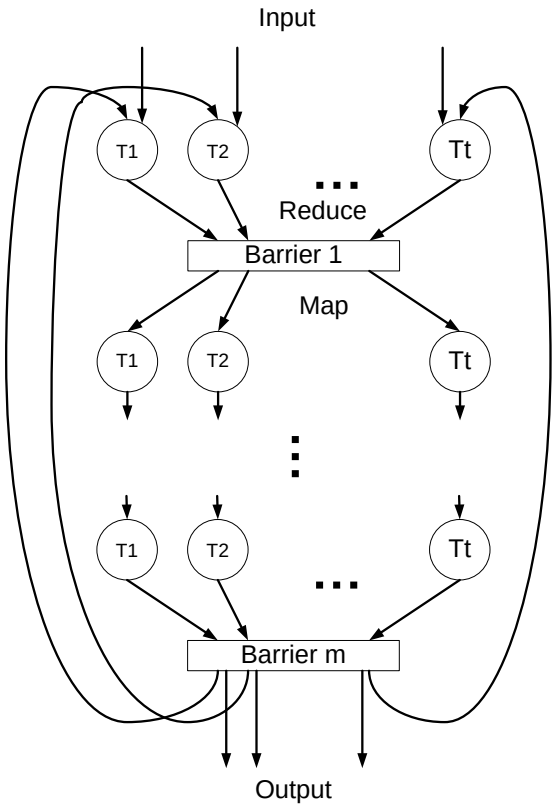


Figure 5.2: MapReduce flowchart for the UPS algorithm.

Figure 5.3 gives an example of edge contraction. The hyperedge $\{3, 5, 6\}$ is being contracted. The super node $3'$ is made up of nodes 3, 5 and 6, which are neighboring nodes of hyperedge $\{3, 5, 6\}$. Previous hyperedges $\{1, 2, 3\}$ and $\{4, 5\}$ become $\{1, 2, 3'\}$ and $\{4, 3'\}$, respectively. After we partition a hypergraph, we can select a subset of cut edges to contract, resulting in a coarsened hypergraph. Repartitioning on this coarsened graph will not have any cut placed on the contracted edges. A coarsened hypergraph can be uncoarsened by simply unfolding contracted hyperedges, which is exactly a reversed process to contraction.

Now we consider the greedy sequencing with partitioning to obtain a sequence of hyperforests. We can first run *hyperspan* on the hypergraph which gives us a set of conditioned hyperedges. Then we can partition the resulting hypertree or hyperforest.

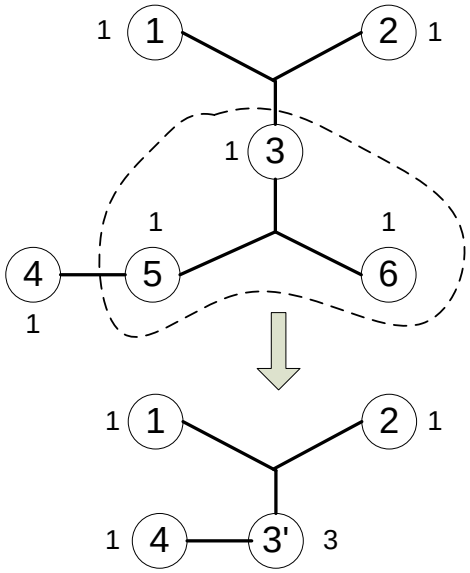


Figure 5.3: Coarsening by contracting edge 3, 4 and 5.

This also gives us a cut set of hyperedges that need to be conditioned on. Note that we need to treat the condition set and cut set differently in terms of meeting the criterion in Proposition 4.4.6. For a hyperedge in the condition set, we want it to be unconditioned in at least one subproblem. This can be achieved by assigning highest priority to it in the spanning tree algorithm. For a hyperedge in the cut edge, we have to protect it from being placed on the cut edge for every subproblems. Only by making it appear in the hypertree before partitioning will not guarantee this because it can be selected as a cut edge as a result of partitioning. So we have to maintain two lists of hyperedges, one for the conditioned hyperedges, and for the cut edges, both populated after the construction of the first subproblem. These two lists get updated during the execution of the sequencing method and when both become empty, we obtain a sequence of subproblem satisfying the convergent criterion. The greedy sequencing method with partitioning is shown in Procedure 5. For a hypergraph $H(V, E)$, we write the partioned hypergraph as $partition(H(V, E))$.

Near optimal speedup is only achieved when all tasks take approximately same

Procedure 5 Greedy Sequencing with Partitioning

input: A hypergraph $H(V, E)$.

input: An ordered list L of E.

input: A list $Cond$ of hyperedges. Initially empty.

input: A list Cut of hyperedges. Initially empty.

output: A sequence $Span$ of hyperforests.

$s(V, E') = hyperspan(H, L)$. Add $E \setminus E'$ to $Cond$.

$s(V, E'') = partition(s(V, E'))$. Add the cut edges $E' \setminus E''$ to Cut.

Add s to $Span$.

while $Cond$ is not empty || Cut is not empty **do**

L' = concatenation of $L \setminus Cond$ to $Cond$.

$s(V, E') = hyperspan(H, L')$.

Coarsen $s(V, E')$ on a subset c of Cut

$s(V, E'') = partition(s(V, E'))$. $Cut = Cut \cap (E' \setminus E'')$.

Remove $Cond \cap E'$ from $Cond$.

Uncoarsen s then add to $Span$.

end while

amount of time. Load balancing is particularly hard because there is no prior information on how many iterations IS takes. Note BP is a component of IS, and IS solution time is always a multiple of BP solution time. IS solution time is correlated with but likely not proportional to the number of nodes in the hypertree. We perform weighted partitionings. The weight of a node is 1 for a regular node. For a super node, the weight is the number of regular nodes it contains. In Figure 5.3, the weights are shown besides the nodes. From our experimentation, we observe that reasonable load balance is achieved through weighted partitioning when the average interaction between adjacent random variables is not too high. For high interaction, partitioning-based load balancing performs poorly. In Section 5.6, we show this effect

and propose some techniques to accommodate it. Since this approximate equalization of task running time is an attempt made before the execution of the algorithm, we call it static load balancing (SLB).

The hypergraph partitioning in our method is performed using a multilevel hypergraph partitioning program *hMETIS* developed at University of Minnesota [Karypis and Kumar, 1998]. Compared to alternative programs, it has much shorter solution time and more importantly, it produces balanced partitions with a significantly fewer number of cut edges. The latter implies that we have a smaller inner loop in the greedy sequencing procedure, and the convex relaxation method is expected to converge faster.

5.4 Computing Environment

Our experimenting environment is a shared memory 8-core system with 2 Intel Xeon Quad Core E5410 2.33 GHz processors. The operating system is Debian Linux. Experimental results were collected when no other users were competing for the system resource. The reason for this special treatment is that when other users are time-sharing the system, within one iteration of the parallel method, it is hard to obtain an accurate measure of actual computation time from the elapsed time of different cores, because there might be different degrees of overhead on different cores.

On such a multicore shared memory system, we adopted the common multithreading scheme, where in general, n threads are created in a n-core system and each thread is distributed to a separate core. These threads have the same memory space that is shared among the cores. In another words, any communication between threads can be made through the shared memory. For example, if the data of an inner region get updated in one core, all other cores immediately see the change and when they make reference to the data of that inner region, they will have the updated data. This is the advantage of shared memory system. On the other hand, due to the limited memory bus capacity, for a fixed number of cores, the number of concurrent access

to the memory is much less than in a distributed system, where each processor can access its own memory more freely.

5.5 Software Implementation

We chose to implement the sequential and parallel methods in Java programming language. This allows easy export of our experiments to other platforms. More importantly, it allows us to avoid dealing with the low level threading mechanism.

In our multicore configuration, our program consists of a main control thread, represented as an MPI process and a set of worker threads for the parallel tasks. The MPI threads are facilitated by MPJ Express [Taboada et al.], a publicly available Java package that allows application developers to write and execute parallel applications for multicore processors and compute clusters/clouds. The main thread first performs the sequencing and partitioning, then constructs and starts the worker threads. The main thread is idle during most of the time and it only performs the barrier function when all threads finish the computation for the current iteration. When a thread finishes its computation, it will wait for other threads.

As shown in Figure 5.2, all threads are synchronized at a software barrier point to ensure that all subproblems converge and to test for global convergence. Global convergence means that the change of value of the pseudo-marginal variables between two consecutive outer iterations is below a preset threshold. Here we use a cyclic barrier which iterates with the main algorithm and waits for all threads to finish the computation before giving permission to each thread to start next iteration's computation. Rather than a plain barrier point, a **Reduce** operation needs to be carried out to check for global convergence. For each worker thread, we associated a Boolean flag indicating whether the solution for the subproblems on this thread differs from the one from the last iteration. At the barrier point, a logic AND operation is taken on these flags to reduce them to one Boolean flag. This flag being true indicating that a fixed point is reached and the algorithm is terminated with a solution.

5.6 Experimental Setup

We test the parallel convex relaxation method using two-dimensional Ising models. When $\beta > 1$, loopy BP fails to converge even for small graphs. In the two dimensional model , the interactions are between pairs of nodes. Since we need large model to demonstrate the effectiveness of our parallel scheme, we choose test examples with size 100×100.

With $\beta = 1.1$, we randomly generated 100 problems where β_{ij}'s are uniformly drawn from [-β, β] and α_i's are uniformly drawn from $[-1, 1]$. The number of cores used ranges from 2 up to 8 to demonstrate both raw speedup and scalability. Speedup is defined as the ratio between sequential elapsed time and parallel elapsed time. With this level of interaction, the sequential run time exceeds 1 minute giving rise to parallelization, and SLB starts performing poorly, in the sense that there is always some component taking much more time than others do.

5.7 Runtime Techniques

Due to this synchronization, the slowest task will determine the overall performance. The static load balancing (SLB) introduced in Section 5.3 performs worse as β increases. In practice, we apply two runtime techniques that turn out to be very effective.

5.7.1 Overpartitioning and Dynamic Load Balancing

Instead of partitioning the graph into n components and distributing them to n threads, we partition the graphs into more components and put them into a customized task pool. The idea of partitioning the graph into more components than the number of processing units is known as *overpartitioning*. By overpatitioning, we can break the graph into smaller components and the tasks defined on these components have smaller granularity. This will help us achieve a more balanced assignment

of tasks to different cores. Usually the task assignment used with overpartitioning are static, i.e., before execution of the instance of the algoritm. This would work very effectively in cases where the relative computation times of tasks are known or can be estimated fairly accurately statically.

In our case, the computation time of tasks may not be proportional to the size of their components, so a static load balancing with overpartitioning may still result in a uneven load distribution. We propose a load balancing scheme, which instead of being static, happens at the runtime. This is indeed a task scheduling mechanism which can be carried out autonomously by each core. First we still have the overpartitioned graph and the associated task pool. At runtime, each thread fetches a task from the pool when it finishes its current task. A thread will be kept busy until there is no tasks left in the pool then it will wait for other threads finishing their computation. This maximizes the utility of each core by keeping them busy doing useful work as much as possible during an entire iteration. The number of tasks each core processed can vary but the overall computation should approximately equal (within the runtime of one task).

The dynamic load balancing scheme is implemented as a set of distributed task scheduler, one for each thread. When a thread finishes its current task, the task scheduler of this thread will fetch one task from the task pool if there is any left. The number of tasks remaining in the pool is kept in an integer counter. The task fetch action is carried out independently of other threads. We need to make this action synchronized so that the task pool is always consistent when viewed from different threads. So when a thread is accessing the task pool, it will first place a lock on the counter so that accesses attempted by other threads will be denied. The thread then fetch the task and decrement the counter before release its access to other threads. In practice this synchronization mechanism will only introduce negligible overhead since the chance of simultaneous access to the task pool from multiple threads is extremely low.

In our parallel scheme, the overpartitioning has a drawback. When we further partition the graph, there will be more hypereges selected as cut edges. This means in the convex relaxation method, for each subproblem, we have a larger condition set which will very likely result in a longer sequence of subproblems. This is because according to the criterion in Proposition 4.4.6, any conditioned inner regions (cut edges) need to be unconditioned for some iteration. The fulfillment of this requirement for larger condition sets will in general be achieved across a longer sequence of subproblems. This may decrease the convergence rate of the problem because during each inner iteration, there is an increased number of variables held fixed and fewer variables got updated. However we view this as a minor problem and its impact on the convergence speed will be dominated by the gain achieved by overpartitioning and dynamic load balancing.

5.7.2 Early Termination

We have discussed that the relative solution time of a subproblem is not proportional to the weight of its component. For example, if we have two components, with weight W_1 and W_2, solution time T_1 and T_2, respectively, then $\frac{T_1}{T_2}$ is not proportional to $\frac{W_1}{W_2}$. The time a subproblem takes is determined by the solution time of the bottleneck component, which may or may not be the largest one. Threads running on bottleneck component are called bottleneck threads. If a bottleneck thread runs a lot of longer than other threads, the overall performance is compromised.

Now we introduce a technique called early termination (ET) of the bottleneck task. With ET, we terminate a thread when all other threads become idle and no task is left in the pool. However terminating a task prematurely has two undesirable effects. First, it breaks the convergence requirement. Second, it may change the convergence rate even if the method finally converges. In both cases, the number of total outer iterations will be much greater. To mitigate both issues, we introduce a parameter P_{extra} to control the strictness of ET. P_{extra} specifies the proportion of the

extra time granted to the bottleneck thread to run after other threads finishes their computation. For example, if $P_{extra} = 1$, the bottleneck thread can run twice as long as other threads. Intuitively setting larger P_{extra} will help the algorithm converge because when the bottleneck thread is terminated, the closer it is to the convergence of this subproblem, the less likely it will affect the global convergence. In order to ensure the algorithm still converges, at runtime we can occasionally switch back to non-ET mode, especially when oscillation of messages is detected. In practice, we found that no ET actually occurs when close to the final solution, indicating that the convergence condition is indeed satisfied.

Setting the value of P_{extra} is difficult because it is impossible to estimate what is going to happen. A large P_{extra} will weaken the effectiveness of ET. However, even a P_{extra} with value 1 may drastically decrease performance. For example, suppose we have 8 cores and 8 tasks, with the first seven taking time T and the last taking time $2T$, by setting P_{extra} being 1, for 7 out of 8 cores, half of the time will be wasted. If this happens for every iteration, it will be inefficient. Setting P_{extra} is a trade-off because increasing the utility within each iteration may increase the total number of iterations. However our experience shows that even with P_{extra} not very small, the inefficient use of the parallel system will not universally occur for all iterations. The uneven task time will be diluted by the large number of total iterations.

The ET is also implemented as a distributed monitor for each thread. The monitor checks the counter variable introduced in Section 5.7.1 at the end of each iteration of the IS algorithm. When counter is detected to be 0, i.e., no task left in the pool, the monitor will trigger the thread into a count down state where the upper limit of extra execution time for this thread will the P_{extra} times the execution time of the finished threads.

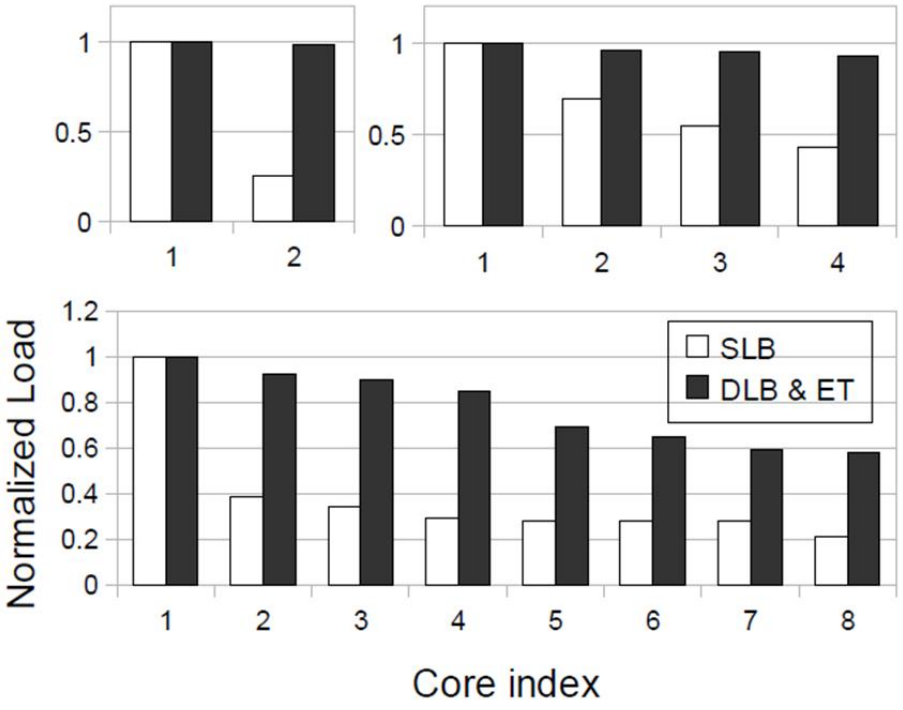


Figure 5.4: Comparison of load balance: DLB & ET vs. SLB. Plotted are normalized load vs. cores for 3 cases: 2 cores (upper left), 4 cores (upper right) and 8 cores (bottom).

5.8 Results

Using a single test example, we compare the load balance obtained using SLB with the one using DLB and ET. We consider to run the example in 3 cases: 2 core, 4 cores and 8 cores and in all 3 cases we use same overpartition number. The load L_i for a particular core i is calculated as the total computation time for the execution of algorithm, averaged by the total number of iterations. The normalized load for core i is

$$\frac{L_i}{\max_{i=1}^{N} L_i}$$

where N is number of nodes. Note that the load balance does not reflect what happens in individual iterations. It can well be that some core runs much longer in a particular

iteration than others even though the overall load are balanced.

Note that in each case, a more balanced load scheme would yield more aligned bars and as shown in Figure 5.4, very uneven load balance results from SLB, regardless of the number of cores used. As desired, this is dramatically mitigated by DLB and ET. Notice that almost perfect balance is achieved for a small number of cores and the load becomes less balanced as the number of cores scales up. This can be explained as with same number of tasks in the pool (due to the same overpartition number), on average it is always easier to distribute them uniformly using a smaller number of cores. Consider a contrived extreme case where we have 8 tasks with running time $\{4, 1, 1, 1, 1, 1, 1, 1\}$. Using 2 and 4 cores, the load is $\{5, 6\}$ and $\{4, 3, 2, 2\}$, respectively. The maximum load difference using 2, 4 and 8 cores are then 1, 2, and 3, respectively.

In Figure 5.5, we compare the average speedup of SLB to that of DLB and ET, using 100 problem instances. DLB and ET uniformly improve the speedup and the improvement becomes more prominent as the number of cores increases. With DLB and ET, the speedup is close to ideal with the number of cores up to 5. A drop from the ideal speedup is observed as the number of cores grows. This can be attributed to two factors. First, as shown in Figure 5.4, even with DLB and ET, load becomes less balanced as the number of cores increases. Second, there is an increased level of resource contention in terms of memory bandwidth. The BP algorithm frequently accesses memory. As more tasks are running in parallel, the number of concurrent memory accesses also increases. When the required throughput puts more demand on the memory bandwidth, speedup drops below the ideal value. This is one of the disadvantages discussed in Section 5.4.

Since BP algorithms are all data-dependent, so we evaluate the applicability of the proposed parallel techniques to different problem instances. In Figure 5.6, we illustrate the distribution of the speedup values of all 100 problems using box plot for each number of cores. Each box plot corresponds to the 100 problems and plotted from top to bottom are the minimum, lower quartile, median, upper quartile and maximum.

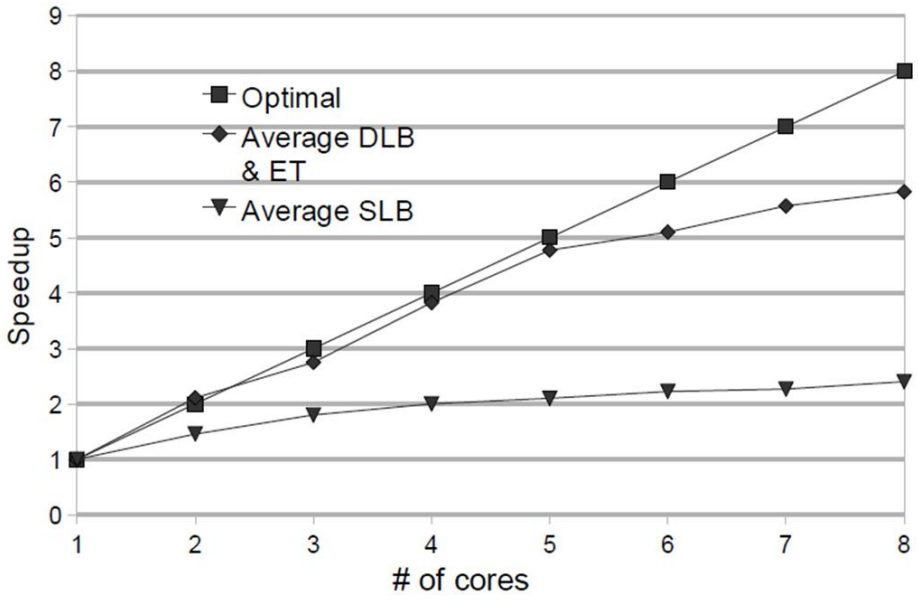


Figure 5.5: Speedup comparison: DLB & ET vs. SLB. Ideal speedup is also plotted.

It can be seen that DLB/ET has a consistent behavior over all 100 problems while SLB performs rather poorly in all cases. Relying on its capability to handle load dynamically, DLB/ET minimizes the performance gap between the hardest instance and the easiest one.

5.9 Distributed Case

5.9.1 Reducing Communication Overhead

In this section, we discuss the problem of reducing communication overhead in a distributed memory system. In a distributed system, each processor has its own memory. Different processors may acquire different copies of the same variable. For example, in Figure 5.1, data of inner region 4 is updated in processor 2 in the first iteration. Processor 1 still has the initial data value of inner region 4, so at the end of iteration 1, the updated data will be synchronized between two processors. In general, as components are assigned to separate processors, one region may belong to more

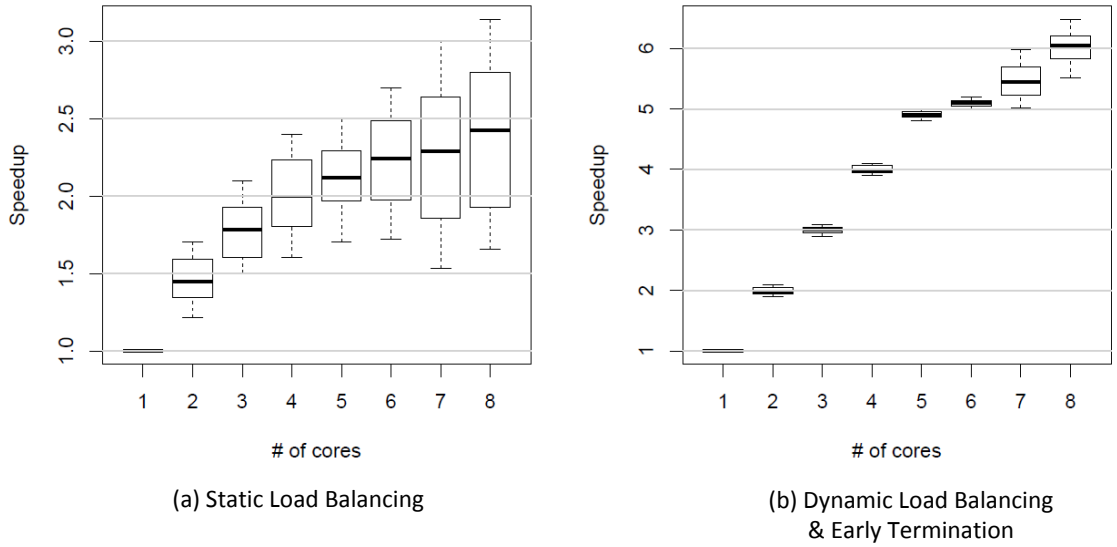


Figure 5.6: Box plots of speedup for 100 runs: DLB & ET vs. SLB.

than one processors and need to be synchronized between consecutive iterations.

If at the end of each iteration, each processor sends all data that are needed by other processors and receives from other processors the data it needs, we call this method a *fully synchronized method*. In the worst case, this synchronization can incur significant communication overhead. Alternatively, we can use partial synchronization to reduce communication overhead. At the end of each iteration, we only synchronize those regions that are going to be separators in the next iteration, if these regions have been updated in the current iteration. Compared to the fully synchronized method, the synchronization cost is minimal. The next proposition shows that the partially synchronized parallel method has exactly the same solution as the fully synchronized parallel method.

Proposition 5.9.1. *Suppose we have a Bethe region graph* $B(\mathcal{O}, \mathcal{I}, \mathcal{E})$ *and a sequence of condition sets* $\{\mathcal{I}_1, \ldots, \mathcal{I}_m\}$ *such that* $\mathcal{I}_i$, $i \in \{1, \ldots, m\}$, *separates* B *into* t *components. Then the iterates* $\{q^k\}$ *generated by the fully and partially synchronized parallel*

method are same provided both methods have same initial solution.

Proof. Let $\mathcal{P}_{ij}$ denote the regions assigned to the i-th processor at k-th iteration such that $j = ((k-1) \bmod m)+1$. Let $\mathcal{P}_i = \cup_{j=1}^{m} \mathcal{P}_{ij}$ $i = 1, \ldots, t$. Let $\mathcal{S}_{jl} = \cup_{i=1}^{t} (\mathcal{P}_{il} \setminus \mathcal{P}_{ij})$, where $l = ((j+1) \bmod m) + 1$. We have $\mathcal{R} = \mathcal{O} \cup \mathcal{I} = \cup_{i=1}^{t} P_i$.

We prove that the iterates generated by these two methods are always same, by induction on the iteration number. By assumption, at first iteration, the pseudo-marginals q of both methods are same. Now assume q_k are same at the end of iteration k, where $j = ((k-1) \bmod m) + 1$, then we have for both methods, the Bethe free energy function $F_b^k = F_b(q_{\mathcal{S}_{jl}}^k, q_{\mathcal{R}\setminus\mathcal{S}_{jl}}^k)$, where we use the subsets of regions to index the whole pseudo-marginal vector q. Let $\Phi^{k+1}(q_{\mathcal{S}_{jl}}, q_{\mathcal{R}_l\setminus\mathcal{S}_{jl}}) = F_b(q_{\mathcal{S}_{jl}}, q_{\mathcal{R}_l\setminus\mathcal{S}_{jl}}, q_{\mathcal{R}\setminus\mathcal{R}_l}^k)$. At the end of iteration $k+1$, for both methods we have

$$q_{\mathcal{R}_l}^{k+1} = \arg \min_{q_{\mathcal{R}_i}:(q_{\mathcal{R}_i}, q_{\mathcal{R}\setminus\mathcal{R}_i})\in Q} \Phi^{k+1}(q_{\mathcal{S}_{jl}}, q_{\mathcal{R}_l\setminus\mathcal{S}_{jl}}), \tag{5.1}$$

$$q^{k+1} = (q_{\mathcal{R}_l}^{k+1}, q_{\mathcal{R}\setminus\mathcal{R}_l}^k), \tag{5.2}$$

and

$$F_b^{k+1} = \Phi^{k+1}(q_{\mathcal{S}_{jl}}^{k+1}, q_{\mathcal{R}_l\setminus\mathcal{S}_{jl}}^{k+1}) = F_b(q^{k+1}),$$

where Q denotes the constraint set. For iteration $k+1$, $q_{\mathcal{R}\setminus\mathcal{R}_l}^k$ is available, because $\mathcal{R}\setminus\mathcal{R}_l$ are the separators in the current iteration and by assumption they are synchronized in both parallel methods.

We know that $F_b^{k+1} \leq F_b^k$ because $(q_{\mathcal{S}_{jl}}^k, q_{\mathcal{R}\setminus\mathcal{S}_{jl}}^k)$ is a feasible solution of Φ^{k+1} and Φ^{k+1} is convex. $B[\mathcal{R}_l]$ satisfies Theorem 3.1 in Su (2010). Thus, Φ^{k+1} is strictly convex and the solution to (5.1) in both methods is the unique global solution of Φ^{k+1}. This means that q^{k+1} is the same for these two methods. This completes our proof. □

Figure 5.7 illustrates a 1-dimensional case. It shows that at a particular point in the algorithm execution, the minimization of the strictly convex subproblem will converge to the same global minimum despite different starting points.

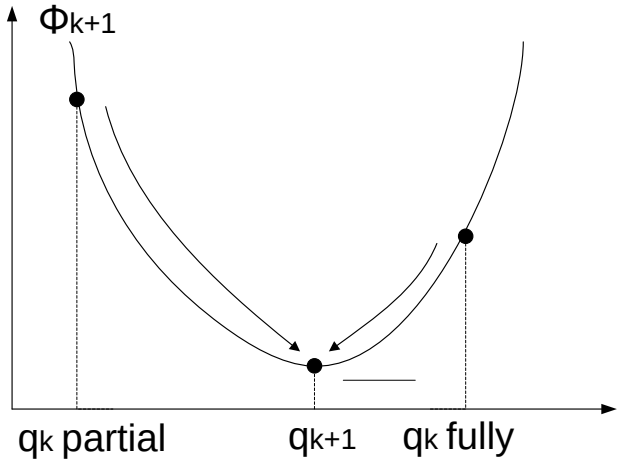


Figure 5.7: Optimizing 1-dimensional strictly convex function with different initial solutions.

Indeed, the algorithm converges with any permutation of the sequence $\{\mathcal{I}_1, \ldots, \mathcal{I}_m\}$. The communication cost between ith and jth iteration depends both on $\mathcal{I}_i$ and $\mathcal{I}_j$. However a sequence having optimal communication cost may decrease convergence rate. The theoretical analysis of how an optimal sequence can be constructed will be very difficult.

5.9.2 Implementation and Experiments

Again we manage these concurrently running threads by using MPJ Express. The Java statement for this is

MPI.*COMM_WORLD*.Allreduce(converg,0,converg,0,1,MPI.*BOOLEAN*,
MPI.*LAND*).

Its semantic is performing a logical AND operation on a set of Boolean variables, each from a thread and named **converg**. The result of the operation is stored in the **converg** variable of the main thread.

The multicore communication device uses efficient inter-thread mechanism. Data communication is also done using nonblocking send and blocking receive. We are able to use nonblocking send, which is usually faster. This is because the memory for the data being sent will only be overwritten after the barrier point is reached, but by then the receiver would have successfully received the data. Blocking receive has to be used to ensure that all data being transmitted are correctly received before the barrier point; that is, before the next iteration starts.

Instead of sending individual pseudo-marginal variables one by one, which is highly inefficient due to the messaging overhead, we group all pseudo-marginals that are going to be transmitted between two threads into one package for sending and receiving. Sender and receiver, respectively, use the predefined protocol to packing and unpacking the aggregate into individual pseudo-marginal variables.

5.10 Discussion

We showed that in the convex relaxation method, coarse-grained parallelism can be effectively extracted. We presented techniques for task partitioning, load balancing, and communication overhead reduction. This parallel implementation is at the algorithmic level, which indicates that it can be combined with other lower level parallelization techniques. Experiments on a shared memory system exhibit near-ideal speedup with reasonable scalability.

Although the convex relaxation method is essentially embarrassingly parallel, its data-dependent running time poses challenges in achieving sufficient parallel efficiency. Since the running times of partitioned tasks are not proportional to the sizes of the subgraphs, nor are they easily estimatable, we have introduced two techniques to load-balance on the fly. The first one, overpartition is fairly effective and solves the problem to some extent despite its overhead. Not to our surprise, the part of the problem left unsolved is again caused by the running time dependency on data. We achieve some improvement through the introduction of a heuristic early termination technique.

However, this technique violates the algorithm's update rule for convergence and may work poorly in practice. As a future direction, we think a more informative and elegant method should be developed to use in combination with overpartitioning. For this method, a desired property is not to break the convergent update rule of the original algorithm.

With the facility constraint, we only experiment on a limited number of cores for shared memory framework. Further exploration is necessary to demonstrate that the speedup scales up in practice on large multicore shared memory systems. However such systems are usually high cost and less accessible than a distributed systems with low cost machines with the number of cores often ranging from 1 to 8. Our experiments shows that for the test instances the proposed parallel technique is not suitable for distributed systems due to the dominant communication overhead. However this result only poses a limit on the granularity of the partitioned tasks, it doesn't eliminate the use of our method in large problems where there are an sufficient amount of computation after partitioning such that the communication overhead will be relatively small. Note that in a distributed framework, communication overhead always can be reduced by optimally mapping the tasks to computing units such that the number of messages, i.e., updated peusomarginals, is minimized.

Chapter 6

SUMMARY OF THIS DISSERTATION

This disseration work studies approximate inference algorithms for probabilistic graphical models, with a concentration on models arising from modern genetic linkage analysis. We have explored three aspects of approximat inference methods: convergence, computational efficiency and performance in practical applications.

We have conducted theoretical convergence analysis for an important approximate method. Our analysis establishes connection between fixed points of belief propagation algorithms and convergence of numerical optimization algorithms. Inspired by related works in optimization and statistics, we derive a sequence of mathematical properties for the method which lead to a mild condition for the method's convergence. We also proposed simple but effective ways to satisfy the conditions for convergence. Future work can be along the direction to derive stronger condition for convergence and the development of the associated algorithm.

Our approach to boosting computational efficiency is through parallelization. We first observed the embarrassing parallelism inherent to the algorithm. Then directed by experimentation, we worked out practical techniques to extract maximal parallelism. Software implementation and thorough experimentation on shared memory systems are also given. As discussed in Chapter 5, future work would be seeking for ways to improving the load balance in order to achieve better scalability and ways to reducing communication overhead in distributed memory systems.

To demonstrate how pratical BP-like approximate inference methods are in linkage analysis, we have developed a methodology combining MCMC sampling method and approximate inference algorithm, to solve a two-locus genetic linkage problem

that demands extensive computational power. The proposed work is implemented as a software tool and through a simulated example, we showed its performance and various analysis results, such as parameter sensitivities, etc. Traditional exact and approximate inference are not likely to fulfill the computational demand for modern linkage genetic studies where a large number of SNP sites are going to be investigated, not to mention different combinations of parameter values. As marker-based MCMC sampling approach has demonstrated to be promising in handling nowaday's data, novel methodologies making efficient use of approximate inference method in conjuction with MCMC sample should be developed to increase the analyzing capability of current linkage methods.

BIBLIOGRAPHY

Abecasis GR, Cherny SS, Cookson WO, and Cardon LR. 2002. Merlin-rapid analysis of dense genetic maps using sparse gene flow trees. *Nature Genetics*, 30:97101.

Allen D and Darwiche A. 2008. Rc_link: genetic linkage analysis using bayesian networks. *Int. J. Approx. Reasoning*, 48(2):499–525.

Almasy L and Blangero J. 1998. Multipoint quantitative trait linkage analysis in general pedigrees. *American Journal of Human Genetics*, 62:1198–1211.

Arnborg S. 1985. Efficient algorithms for combinatorial problems on graphs with bounded decomposibility. *BIT*, 25:2–23.

Arnborg S, Corneil DG, and Proskurowski A. 1987. Complexity of finding embeddnigs in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–84.

Baum LE, Petrie T, Soules G, and Weiss N. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions on markov chains. *Ann Math Stat*, 41:164–171.

Baxter RJ. 1982. *Exactly Solved Models in Statistical Mechanics*. New York: Academic.

Bertsekas DP. 1999. *Nonlinear Programming*. Athena Scientific.

Biernacka JM, Sun L, and Bull SB. 2005. Simultaneous localization of two linked disease susceptibility genes. *Genetic Epidemiology*, 28:33–47.

Biswas S, Papachristou C, Irwin ME, and Lin S. 2003. Linkage analysis of the simulated data - evaluations and comparisons of methods. *BMC Genetics*, 4(Suppl. 1):S70.

Botetz B. Efficient belief propagation for vision using linear constraint nodes. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

Brown MD, Glazner CG, Zheng C, and Thompson EA. 2012. Inferring coancestry in population samples in the presence of linkage disequilibrium. *Genetics*, 190(4): 1447–60.

Cannings C, Thompson EA, and Skolnick MH. 1978. Probability functions on complex pedigrees. *Advances in Applied Probability*, 10(1):26–61.

Cannings C, Thompson EA, and Skolnick MH. Pedigree analysis of complex models. In *Mielke J, Crawford M (eds): Current Developments in Anthropological Genetics*, pages 251–298. Plenum Press, 1980.

Carbonetto P, de Freitas N, and Barnard K. A statistical model for general contextual object recognition. In *In ECCV*, pages 350–362, 2004.

Chandrasekaran V, Srebro N, and Harsha P. Complexity of inference in graphical models. In *In Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, 2008.

Chen L and Storey J. 2006. Relaxed significance criteria for linkage analysis. *Journal of Genetics*, 173(4):2371–2381.

Clerget-Darpoux F, Bonaiti-Pellie C, and Hochez J. 1986. Effects of misspecifying genetic parameters in lod score analysis. *Biometrics*, 42:393–399.

Cooper GF. 1990. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405.

Cormen T H, Leisserson C E, Rivest R L, and Stein C. 2009. *Introduction to algorithms (3rd Edition)*. MIT Press.

Cottingham RW Jr, Idury RM, and Schaffer AA. 1993. Faster sequential genetic linkage computations. *American Journal of Human Genetics*, 53:252–263.

Dean J and Ghemawat S. Mapreduce: Simplified data processing on large clusters. In *In Proceedings of the Sixth Symposium on Operating System Design and Implementation*, 2004.

Drton M and Eichler M. 2006. Maximum likelihood estimation in gaussian chain graph models under the alternative markov property. *Scandinavian Journal of Statistics*, 33(2):247–257.

Elston RC and Stewart J. 1971. A general model for the genetic analysis of pedigree data. *Human Heredity*, 21(8):523–542.

Fishelson M and Geiger D. 2002. Exact genetic linkage computations for general pedigrees. *Bioinformatics*, 18 Suppl. 1:S189–198.

George AW and Thompson EA. 2003. Multipoint linkage analyses for disease mapping in extended pedigrees: a Markov Chain Monte Carlo approach. *Stat Sci*, 18:515–531.

Globerson A and Jaakkola T. Convergent propagation algorithms via oriented trees. In *In Uncertainty in Artificial Intelligence*, 2007.

Gonzalez J, Low Y, and Guestrin C. Residual splash for optimally parallelizing belief propagation. In *In Proceedings of 12th International Conference on Artificial Intelligence and Statistics*, 2009a.

Gonzalez J, Low Y, Guestrin C, and O'Hallaron D. Distributed parallel inference on large factor graphs. In *In Proceedings of 25th International Conference on Uncertainty in Artificial Intelligence*, 2009b.

Grippo L and Sciandrone M. 2000. On the convergence of the block nonlinear gauss-seidel method under convex constraints. *Operations Research Letter*, 26(3):127–136.

Gudbjartsson DF, Jonasson K, Frigge ML, and Kong A. 2000. Allegro, a new computer program for multipoint linkage analysis. *Nature Genetics*, 25:12–13.

Haldane JBS. 1919. The combination of linkage values and the calculation of distances between the loci of linked factors. *Journal of Genetics*, 8:229–309.

Hazan T and Shashua A. 2008. Convergent message-passing algorithms for inference over general graphs with convex free energies. *The 24th Conference on Uncertainty in Artificial Intelligence (UAI).*

Heskes T. 2006. Convexity arguments for efficient minimization of the Bethe and Kikuchi free energies. *Journal of Artificial Intelligence Research*, 26:153–190.

Heskes T, Albers K, and Kappen B. Approximate inference and constrained optimization. In *Uncertainty in Artificial Intelligence*, pages 313–320. Morgan Kaufmann Publishers, 2003.

Jancsary J and Matz G. Convergent decomposition solvers for tree-reweighted free energies. In *In Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 2011.

Jensen FV. 1997. *Introduction to Bayesian networks (1st Edition).* Springer.

Jordan M, Ghahramani Z, Jaakkola TS, and Saul L. 1999. An introduction to variational methods for graphical models. *Learning in Graphical Models*, pages 105–161.

Joris M and Kappen H. Sufficient conditions for convergence of loopy belief propagation. In *Proceedings of the 25th Conference Annual Conference on Uncertainty in Artificial Intelligence*, pages 396–40, 2005.

Karypis G and Kumar V. hMetis: a hypergraph partitioning package. `http://www.cs.umn.edu/ karypis/metis/hmetis`, 1998.

Kindermann R and Snell JL. 1980. *Markov Random Fields and Their Applications*. American Mathematical Society.

Knapp M, Seuchter SA, and Baur MP. 1994. Two-locus disease models with two marker loci: the power of affected-sib-pair tests. *American Journal of Human Genetics*, 55:1030–1041.

Koepke H, Koepke L, and Thompson EA. 2011. Efficient testing operations on dynamic structures using strong hash functions. *Journal of Computational and Graphical Statistics*, Under review.

Kozlov A and Singh J. A parallel lauritzen-spiegelhalter algorithm for probabilistic inference. In *In Proceedings of the 1994 Conference on Supercomputing*, pages 320–329, 1994.

Kruglyak L, Daly MJ, Reeve-Daly MP, and Lander ES. 1996. Parametric and nonparametric linkage analysis: a unified multipoint approach. *American Journal of Human Genetics*, 58:1347–1363.

Kschischang FR, Frey BJ, and Loeliger H. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519.

Lander ES and Green P. 1987. Construction of multilocus genetic linkage maps in humans. *Proceedings of the National Academy of Sciences of the United States of America*, 84(8):2363–7.

Lange K and Sobel E. 1991. A random walk method for computing genetic location scores. *American Journal of Human Genetics*, 49:1320–1334.

Lathrop GM and Ott J. 1990. Analysis of complex diseases under oligogenic models and intrafamilial heterogeneity by the linkage programs. *American Journal of Human Genetics*, 47(Suppl.):A188.

Lathrop GM, Lalouel J, Julier C, and Ott J. Strategies for multilocus linkage in humans. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 81, pages 3443–3446, 1984.

Lathrop GM, Lalouel J, Julier C, and Ott J. 1985. Multilocus linkage analysis in humans: detection of linkage and estimation of recombination. *American Journal of Human Genetics*, 37:482–498.

Lauritzen SL and Spiegelhalter DJ. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224.

Lin WY and Schaid DJ. 2007. Robust multipoint simultaneous identical-by-descent mapping for two linked loci. *Human Heredity*, 63:35–46.

Luo ZQ and Tseng P. 1992. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal Optimization Theory and Applications*, 72(1):7–35.

Markianos K, Daly MJ, and Kruglyak L. 2001. Efficient multipoint linkage analysis through reduction of inheritance space. *American Journal of Human Genetics*, 68: 963–977.

Meltzer T, Globerson A, and Weiss Y. Convergent message passing algorithms - a unifying view. In *In Uncertainty in Artificial Intelligence*, 2009.

Mooij JM. 2010. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11:2169–2173. http://cs.ru.nl/ jorism/libDAI/.

Morton NE. 1955. Sequential tests for the detection of linkage. *American Journal of Human Genetics*, 7(3):277–318.

Namasivayam VK, Pathak A, and Prasanna VK. Scalable parallel implementation of bayesian network to junction tree conversion for exact inference. In *Proceedings of the 18th International Symposium on Computer Architecture and High Performance Computing*, pages 167–176, Washington, DC, USA, 2006.

Nocedal J and Wright S. 1999. *Numerical Optimization*. Springer.

OConnell JR and Weeks DE. 1995. The VITESSE algorithm for rapid exact multilocus linkage analysis via genotype set-recoding and fuzzy inheritance. *Nature Genetics*, 11:402–408.

Ott J. 1976. A computer program for general linkage analysis of human pedigrees. *American Journal of Human Genetics*, 26:588–597.

Pakzad P and Anantharam V. Belief propagation and statistical physics. In *Princeton University*, 2002.

Pakzad P and Anantharam V. 2005. Estimation and marginalization using Kikuchi approximation methods. *Neural Computation*, 17:1836–1873.

Pelizzola A. 2005. Cluster variation method in statistical physics and probabilistic graphical models. *Physics A: Mathematical and General*, 38:309–339.

Pennock D. Logarithmic time parallel bayesian inference. In *Proc. 14th Conf. Uncertainty in Artificial Intelligence*, pages 431–438, 1998.

Powell MJD. 1973. On search directions for minimization algorithms. *Mathematical Programming*, 4(1):193–201.

Risch N. 1984. Segregation analysis incorporating linkage markers. I. single-locus models with an application to type I diabetes. *American Journal of Human Genetics*, 36(2):363–386.

Schork NJ, Boehnke M, Terwilliger JD, and Ott J. 1993. Two-trait-locus linkage analysis: a powerful strategy for mapping complex genetic traits. *American Journal of Human Genetics*, 53:1127–1136.

Shachter RD and Andersen SK. Global conditioning for probabilistic inference in belief networks. In *Proc. Tenth Conference on Uncertainty in AI*, pages 514–522, 1994.

Sheehan NA. 2000. On the application of Markov chain Monte Carlo methods to genetic analyses on complex pedigrees. *International Statistical Review*, 68:83–110.

Sieh W, Basu S, Fu AQ, Rothstein JH, Scheet PA, Stewart WCL, Sung YJ, Thompson EA, and Wijsman EM. 2005. Comparison of marker types and map assumptions using Markov chain Monte Carlo-based linkage analysis of COGA data. *BMC Genetics*, 6(Suppl 1):S11.

Silberstein M, Tzemach A, Dovgolevsky N, Fishelson M, Schuster A, and Geiger D. 2006. On-line system for faster linkage analysis via parallel execution on thousands of personal computers. *American Journal of Human Genetics*, 78(6):922–935.

Sobel E and Lange K. 1996. Descent graphs in pedigree analysis: applications to haplotyping, location scores, and marker-sharing statistics. *American Journal of Human Genetics*, 58(6):1323–37.

Sobel E, Sengul H, and Weeks DE. 2001. Multipoint estimation of identity-by-descent probabilities at arbitrary positions among marker loci on general pedigrees. *Human Heredity*, 52:121–131.

Strauch K, Fimmers R, Kurz T, Deichmann KA, Wienker TF, and Baur MP. 2000. Parametric and nonparametric multipoint linkage analysis with imprinting and two-locus-trait models: application to mite sensitization. *American Journal of Human Genetics*, 66:1945–1957.

Strauch K, Fimmers R, Baur MP, and Wienker TF. 2003. How to model a complex trait. 2. analysis with two disease loci. *Human Heredity*, 56:200–211.

Su M. On the convergence of convex relaxation method and distributed optimization of bethe free energy. In *Proceedings of the 11th International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2010.

Su M. Parallelizing a convergent approximate inference method. In *Proceedings of the 24th Canadian Artificial Intelligence Conference (Lecture Notes in Artificial Intelligence by Springer)*, May 2011.

Su M and Thompson EA. 2012. Computationally efficient multipoint linkage analysis on extended pedigrees for trait models with two contributing major loci. *Genetic Epidemiology*, 36(6):602–611.

Sung YJ and Wijsman EM. 2007. Accounting for epistasis in linkage analysis of general pedigrees. *Human Heredity*, 63(2):144–52.

Sung YJ, Di Y, Fu AQ, Rothstein JH, Sieh W, Tong L, Thompson EA, and Wijsman EM. 2007a. Comparison of multipoint linkage analyses for quantitative traits in the ceph data: parametric lod scores, variance components lod scores, and bayes factors. *BMC Proceedings*, 1(Suppl 1):S93.

Sung YJ, Thompson EA, and Wijsman EM. 2007b. MCMC-based linkage analysis for complex traits on general pedigrees: multipoint analysis with a two-locus model and a polygenic component. *Genetic Epidemiology*, 31:103–114.

Taboada GL, Jameel M, Ramos S, Manzoor J, and Hamid K. Mpj express. `mpj-express.org`.

Tarjan RE and Leeuwen J V. 1984. Worst-case analysis of set union algorithms. *Journal of the ACM*, 31(2):245–281.

Teh YW and Welling M. Passing and bouncing messages for generalized inference. Technical Report GCNU TR 2001-01, Gatsby Computational Neuroscience Unit, University College London, 2001a.

Teh YW and Welling M. The unified propagation and scaling algorithm. In *NIPS*, pages 953–960, 2001b.

Thompson EA. 2000. Statistical inference from genetic data on pedigrees. *NSF-CBMS regional conference series in probability and statistics*, 6.

Thompson EA. MCMC in the analysis of genetic data on pedigrees. In *F. Liang, J.-S. Wang, and W. Kendall (eds), Markov Chain Monte Carlo: Innovations and Applications*, pages 183–216. World Scientific, 2005.

Thompson EA. 2011. The structure of genetic linkage data: from LIPED to 1M SNPs. *Human Heredity*, 71:88–98.

Thompson EA and Heath SC. 1999. Estimation of conditional multilocus gene identity among relatives. *Statistics in Molecular Biology and Genetics: Selected Proceedings of a 1997 Joint AMS-IMS-SIAM Summer Conference on Statistics in Molecular Biology*, 33:95–113.

Thompson EA, Lin S, Olshen AB, and Wijsman EM. 1993. Monte Carlo segregation and linkage analysis of a large hypercholesterolemia pedigree. *Genetic Epidemiology*, 10:677–682.

Tomescu I and Zimand M. 1994. Minimum spanning hypertrees. *Discrete Applied Mathematics*, 54(1):67–76.

Tong L and Thompson EA. 2008. Multilocus lod scores in large pedigrees: combination of exact and approximate calculations. *Human Heredity*, 65:142–153.

van Dijk T, van den Heuvel JP, and Slob W. Treewidth Java Package. `http://www.treewidth.com/`.

Wainwright MJ, Jaakkola TS, and Willsky AS. 2005. A new class of upper bounds on the log partition function. *IEEE Transaction on Information Theory*, 51(7): 2313–2335.

Weiss Y. 2000. Correctness of local probability propagation in graphical models with loops. *Neural Comput.*, 12(1).

Welling M. On the choice of regions for generalized belief propagation. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence*, page 585, Arlington, Virginia, 2004. AUAI Press.

Welling M and Teh YW. Belief optimization for binary networks: a stable alternative to loopy belief propagation. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, volume 17, 2001.

Wener T. Primal view on belief propagation. In *Proceedings of the 26th Annual Conference on Uncertainty in Artificial Intelligence*, pages 651–657, Catalina Island, California, 2010. AUAI Press.

Wijsman EM and Amos CI. 1997. Genetic analysis of simulated oligogenic traits in nuclear and extended pedigrees: summary of gaw10 contributions. *Genetic Epidemiology*, 14(6):719–35.

Wijsman EM, Rothstein JH, and Thompson EA. 2006. Multipoint linkage analysis with many multiallelic or dense diallelic markers: Markov chain-Monte Carlo provides practical approaches for genome scans on general pedigrees. *American Journal of Human Genetics*, 79:846–858.

Wilcox MA, Pugh EW, Zhang H, Zhong X, Levinson DF, Kennedy GC, and Wijsman EM. 2005. Comparison of single-nucleotide polymorphisms and microsatellite markers for linkage analysis in the COGA and simulated data sets for Genetic Analysis Workshop 14: Presentation groups 1, 2, and 3. *Genetic Epidemiology*, 29 (Suppl 1):S7–S28.

Winn J and Bishop CM. 2005. Variational message passing. *Joural of Machine Learning Reseach*, 6:661–694.

Xia Y and Prasanna VK. Parallel exact inference on the cell broadband engine processor. In *In Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, 2008.

Xie Z, Gao J, and Wu X. 2009. Regional category parsing in undirected graphical models. *Pattern Recognition Letters*, 30(14):1264–1272.

Yedidia JS, Freeman WT, and Weiss Y. Generalized belief propagation. In *NIPS*, pages 689–695. MIT Press, 2000.

Yedidia JS, Freeman WT, and Weiss Y. 2005. Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51:2282–2312.

Yuille AL and Rangarajan Anand. 2003. The concave-convex procedure. *Neural Computation*, 15(4):915–936.