Statistical Methods for Characterizing Genomic Heterogeneity in Mixed Samples

A Dissertation

Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy in

Biomedical Engineering

December 5th, 2016

by

Fan Zhang

Fan Zhang

APPROVED:

[illegible]

Patrick Flaherty, Ph.D.
Assistant Professor, Advisor
Department of Mathematics and Statistics
University of Massachusetts Amherst

[illegible]

Marsha Rolle, Ph.D.
Associate Professor, Committee Chair
Department of Biomedical Engineering
Worcester Polytechnic Institute

[illegible]

Dirk Albrecht, Ph.D.
Assistant Professor
Department of Biomedical Engineering
Worcester Polytechnic Institute

Andrew C. Trapp

Andrew C. Trapp, Ph.D.
Associate Professor
Foisie School of Business
Worcester Polytechnic Institute

[illegible]

Manuel Garber, Ph.D.
Associate Professor
Bioinformatics and Integrative Biology
Director, Bioinformatics Core
University of Massachusetts Medical School

Abstract

Recently, sequencing technologies have generated massive and heterogeneous data sets. However, interpretation of these data sets is a major barrier to understand genomic heterogeneity in complex diseases. In this dissertation, we develop a Bayesian statistical method for single nucleotide level analysis and a global optimization method for gene expression level analysis to characterize genomic heterogeneity in mixed samples.

The detection of rare single nucleotide variants (SNVs) is important for understanding genetic heterogeneity using next-generation sequencing (NGS) data. Various computational algorithms have been proposed to detect variants at the single nucleotide level in mixed samples. Yet, the noise inherent in the biological processes involved in NGS technology necessitates the development of statistically accurate methods to identify true rare variants. At the single nucleotide level, we propose a Bayesian probabilistic model and a variational expectation maximization (EM) algorithm to estimate non-reference allele frequency (NRAF) and identify SNVs in heterogeneous cell populations. We demonstrate that our variational EM algorithm has comparable sensitivity and specificity compared with a Markov Chain Monte Carlo (MCMC) sampling inference algorithm, and is more computationally efficient on tests of relatively low coverage ($27\times$ and $298\times$) data. Furthermore, we show that our model with a variational EM inference algorithm has higher specificity than many state-of-the-art algorithms. In an analysis of a directed evolution longitudinal yeast data set, we are able to identify a time-series trend in non-reference allele frequency and detect novel variants that have not yet been reported. Our model also detects the emergence of a beneficial variant earlier than was previously shown, and a pair of concomitant variants.

Characterization of heterogeneity in gene expression data is a critical challenge for personalized treatment and drug resistance due to intra-tumor heterogeneity. Mixed membership factorization has become popular for analyzing data sets that have within-sample heterogeneity. In recent years, several algorithms have been developed for mixed membership matrix factorization, but they only guarantee estimates from a local optimum. At the gene expression level, we derive a global optimization (GOP) algorithm that provides a guaranteed ϵ-global optimum for a sparse mixed membership matrix factorization problem for molecular subtype classification. We test the algorithm on simulated data and find the algorithm always bounds the global optimum across random initializations and explores multiple modes efficiently. The GOP algorithm is well-suited for parallel computations in the key optimization steps.

To my parents who loved me unconditionally,

my husband, Chuangqi, who always supported me,

and my son, Ian, who I hope one day will understand what I do.

Acknowledgements

It has been a wonderful experience studying at WPI and it is my pleasure to acknowledge many people here who helped me with my graduate study and research.

First, I would like to express my gratitude to my advisor, Dr. Patrick Flaherty, who always supported and guided me on my research projects and career development over the past few years. He taught me how to interpret real world biomedical data sets accurately and efficiently using his expertise in hierarchical Bayesian modeling and large-scale biomedical data analysis. I really enjoy working with him while doing statistical derivation, coding, writing papers and grants, and presenting at conferences. His motivation and dedication in developing accurate statistical methods to reveal underlying mechanisms in the genetic diseases and then improving patient care has been a constant inspiration for my progression in scientific research.

I would also like to thank Dr. Andrew C. Trapp, who first introduced me to the field of optimization and has mentored me on my research earnestly. He has given me valuable advice on my global optimization project and we published a paper together based on this project. As my dissertation committee member, he was generous with his time in discussing problems that I had in my project. Furthermore, he helped revise my manuscript throughout and examined the inference process very carefully.

I appreciate Dr. Dirk Albrecht, my qualifying exam and dissertation committee member, for his comments on visualization of computational methods for my dissertation proposal. His comments on how to visualize advanced methods for my dissertation proposal were very helpful. I thank Dr. Marsha Rolle for her support for my work as a teaching assistant and my PhD study. Her thoughts on biomedical motivation and significance were very useful. I would like to thank Dr. Kwonmoo Lee, from whom I learned a lot about biomedical imaging analysis and I really enjoyed being a teaching assistant for him in his class on biomedical data analysis. I would like to thank Dr. Yitzhak Mendelson, who gave me many insightful comments on my dissertation proposal. I am grateful to Dr. Manuel Garber, who agreed to be my dissertation committee member. His valuable comments on my project of rare variant detection were very helpful. Moreover, I would like to recognize Dr. Terri Camesano, who helped with making a reasonable timeline for my whole PhD study. Her advice in balancing scientific research and family was very impressive.

Beyond these professors, I learned a lot from my lab members and other graduate students in the biomedical engineering department. Yuting He, Hachem Saddiki, and Josh Harvey brought their ideas in my projects to help them move forward quickly. Late night discussions with them on deriving hypothesis, generating simulated data sets, and debugging code were impactful to me. I also want to thank Ross Lagoy because he taught me how to write a coherent and logical scientific paper.

Finally and most importantly, I would like to acknowledge my husband, Chuangqi Wang, because without his support I would not be able to overcome all the difficulties throughout my PhD study. Special thanks are given to him for his willingness and patience to listen when things were not moving forward smoothly. Furthermore, I must express my gratitude to my parents for their endless love and support to me. Their encouragement has become my incentive to work persistently and make progress in the future.

Contents

List of Figures

List of Tables

List of Abbreviations

ACC	Accuracy
ECM	Expectation-conditional maximization
ELBO	Evidence lower bound
EM	Expectation maximization
FDR	False discovery rate
FN	False negative
FP	False positive
FPR	False positive rate
FWER	Family-wise error rate
GOP	Global optimization
KL	Kullback-Leibler
MAP	Maximum a posteriori
MCC	Matthews correlation coefficient
MCMC	Markov Chain Monte Carlo
NGS	Next-generation sequencing
NPV	Negative predicted value
NRAF	Non-reference allele frequency

PPV	Positive predicted value
PUBD	Primal upper bound
RLBD	Relaxed lower bound
ROC	Receiver operating characteristic
RVD	Rare variant detection
SLSQP	Sequential Least SQuares Programming
SNVs	Single nucleotide variants
TCGA	The Cancer Genome Atlas
TN	True negative
TP	True positive
VAF	Variant allele frequency
WES	Whole-exome sequence
WGS	Whole-genome sequence

Chapter 1

Introduction

The progression from human genome sequence to bedside of patients has shown accomplishments of human genomics research from the basis genomes knowledge to health care applications [1]. Since 2011, genomic discovery is progressively boosting the science of biomedicine. Beyond 2020, it will improve effective treatment and help identify the emergence of chemotherapy resistance [1]. Recently, sequencing technology provides large-scale data sets that have the potential to quantify the genomic changes that drive development of many diseases. By analyzing the genomic data sets, researchers have revealed tumor heterogeneity in the clinical samples at diagnosis and treatment. However, statistically accurate and efficient methods are needed to translate the complex and mixed data sets into knowledge.

1.1 Overview

Research has been focusing on rare variant detection in heterogeneous samples in recent years, as clinical samples are often heterogeneous. Primary tumor samples can be heterogeneous mainly because of contamination from normal cells and genetic sub-populations in a tumor tissue [2]. Immune genomics samples, like leukemic cell free DNA, are heterogeneous, which could be used to detect clonal chromosome abnormalities [3]. Therefore, in these and many other fields where the sample has high impurity, sensitive variant detection tool is more than desirable.

Intra-tumor heterogeneity can contribute to treatment failure and drug resistance [4]. For example, primary breast tumors have been classified into five genomic subtypes, luminal A, luminal B, normal like, basal like, and HER2 overexpression, based on gene expression data [5, 6]. Furthermore, The Cancer Genome Atlas (TCGA) group has revealed four molecular subtypes, classical, proneural, neural, and mesenchymal in glioblastoma tumors [7]. Thus, detection of these distinct genomic subtypes within a tumor is needed because it will lead to an improved combinatorial chemotherapy.

The long term-goal of my research is to develop statistical and computational algorithms to interpret massive biomedical and biological data sets to improve the diagnosis and treatment of cancer. To achieve this goal, the overview of my current research focuses on statistical methods development for characterizing genomic heterogeneity in mixed samples. My background in computer science, statistics, and genetics helped me contribute to two interesting research projects in large-scale clinical data analysis.

First, for single nucleotide level analysis, we develop a probabilistic statistical model and a variational inference algorithm for rare variant detection in next-generation sequencing data. This method is demonstrated to be sensitive for rare variant detection and it has comparable sensitivity and specificity when compared with other state-of-the-art methods. It is helpful to detect genomic variants that can cause genetic diseases or drug resistance.

Second, for gene expression level analysis, we develop a deterministic global optimization algorithm for a sparse mixed membership matrix factorization problem to identify both the underlying genetic subtype signatures and the distributions over these subtypes in mixed tumor samples. Knowing the distribution of subtypes for a given patient is critical because the mixture of genetic subtypes effects treatment. So, we will be able to predict the genomic subtypes and subtype distributions accurately using this novel algorithm.

1.2 Outline of dissertation

The remaining of this dissertation is organized as follows. In Chapter 2, we discuss the recent progress made in variant detection. We also survey and classify the state-of-the-art methods into categories. A common Bayesian probabilistic framework for single nucleotide variant detection is summarized. In Chapter 3, we propose and describe a novel Bayesian statistical model and a variational expectation maximization (EM) algorithm for rare variant detection in deep, heterogeneous NGS data. The performance of this variational EM algorithm is evaluated on both a synthetic data set and a real longitudinal data set. In Chapter 4, we develop a global optimization framework for a sparse mixed membership matrix factorization problem for genomic subtypes classification. We show empirical accuracy of the algorithm and evaluate the performance by both theoretical analysis and empirical measurements of the time complexity. In Chapter 5, we summarize the major contributions of this dissertation and discuss the challenges and opportunities for statistical methods for characterization of genomic data sets.

Chapter 2

Review of Rare Variant Detection Methods for DNA Next-generation Sequencing Data

2.1 Introduction

The overall pipeline for analyzing large-scale next-generation sequencing (NGS) data consists of five steps: quality control, preprocessing, alignment, post-alignment processing, and variant analysis [8, 9]. The last stage, variant analysis, can be further divided into three steps: variant detection, annotation, and visualization. In this review, we focus on single nucleotide variant detection in heterogeneous NGS data, which is crucial for NGS data analysis towards discovering disease-causing sequence variations and identifying known variants present at low population levels in mixed samples.

Identification of rare variants that are present at low frequency in heterogeneous samples is challenging because sequencing errors can overwhelm the true signal from the variant. While resolution required varies by application, here, we define variants with minor allele frequency of less than 1% as rare variants by convention [10, 11, 12]. For example, one study showed an oseltamivir resistance variant, H275Y, existed in a fraction of 0.18% in a H1N1 clinical sample [13]. Since it can be expensive to detect a rare variant event with allele frequency of less than 1% by deep whole-genome sequence (WGS) or whole-exome sequence (WES) on multiple samples due to the sequencing depth required, it is important to extract as much information from the data as possible to avoid missed detections and false positives. Therefore, sensitive computational methods are needed for rare variant detection.

Significant progress has already been made towards improving the power of variant detection methods. Early variant detection method was based on genotyping subtraction between tumour and normal samples [14, 15]. Recent variant detection algorithms are developed using advanced statistical methods, including Bayesian statistics [16, 17, 18, 19, 20] and Fisher's exact [21, 22].

Statistical accuracy is one of the most important concerns for method development for rare variant detection in impure samples. Simple strategies based on setting cut-offs for counting alleles are not robust to sample variations for variant detection in low or moderate sequencing read depths. To improve robustness, statistical and probabilistic methods have been developed and are more reliable in providing measures of uncertainty for genotype inference in variant detection [23]. Specifically, Bayesian statistical methods have been popular for the development of sensitive variant detection tools by incorporating proper prior probabilities of possible genotypes to then predict the true genotype using a maximum a posteriori (MAP) probability. Several studies have used a binomial probabilistic distribution to model sequencing error distributions that can be further used to differentiate true biological variants from errors [13, 20, 24, 25].

Rare variant detection is important for several research and clinical application areas – in particular in cell-free DNA (cfDNA)-based diagnostics. Recently, advanced statistical methods have enabled the detection of rare variants in low fractions of cfDNA and the detected genetic variants can be taken as potential biomarkers for early cancer detection and anti-cancer treatment monitoring [26, 3]. It has been demonstrated that cfDNA sequencing is able to detect tumour-derived variants with high sensitivity and specificity in the frequently mutated genes in pancreatobiliary tumour samples [27]. A "liquid biopsy" extraction method has been used to identify variants at allele frequency of 2% with high sensitivity and specificity within the circulating cfDNA of tumours [28]. Furthermore, a pilot study reveals the feasibility of this idea using leukemic cfDNA in resolving DNA abnormalities [3].

The purpose of this article is to review the state-of-the-art in rare variant detection methods and to provide a framework for comparing methods. First, in Section 2.2, we discuss key attributes of variant detection methods including: accuracy, scalability, and robustness. Then, in Section 2.3, we discuss the factors that could effect the power of variant detection methods (Figure 2.1). In section 2.4, we summarize current state-of-the-art variant detection methods. Finally, we discuss the challenges and opportunities of statistical methods for variant detection for the future.

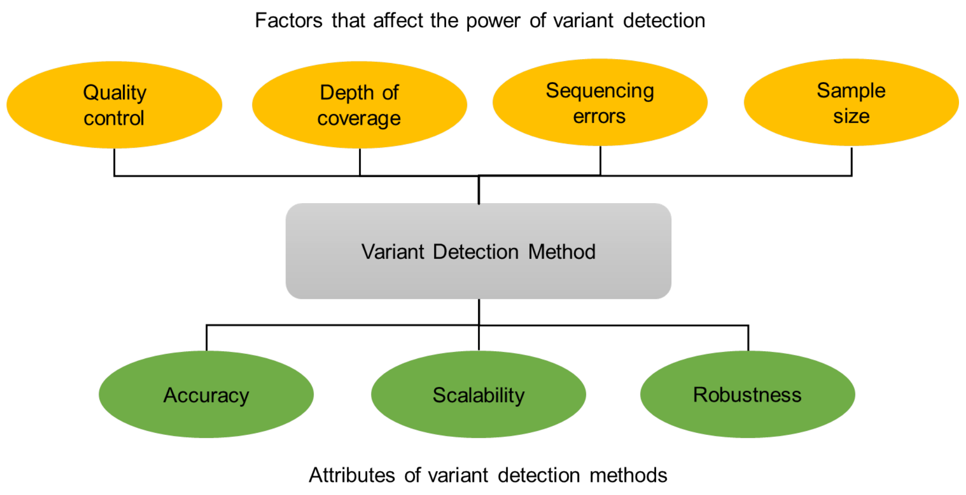


Figure 2.1: An overview of attributes of variant detection methods and factors that influence the power of variant detection. The details of attributes (accuracy, scalability, and robustness) are described in Section 2.2 and the potential factors (quality control, depth of coverage, sequencing errors, and sample size) are described in Section 2.3.

2.2 Attributes of variant detection methods

2.2.1 Accuracy

Accurate detection of SNVs is essential because the detected variants may modulate chemotherapy resistance that could help discover novel variants in clinical cancer samples and lead to improved therapies. To this end, accuracy is normally considered a primary criteria for evaluating the performance of variant detection methods. Simulated sequencing data sets are generally used to evaluate accuracy to test variant detection methods since the underlying truth is known. In some cases, benchmarking data sets with well-recognized sequencing samples can also be performed to validate the methods.

Performance metrics are commonly used to understand the basic values of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) in variant detection. We list several common metrics for evaluating the accuracy of variant detection methods in Table 2.1. Receiver operating characteristic (ROC) curves can also be useful to visualize and interpret the performances of various methods as shown by [29, 30, 31]. Since each metric has its uncertainty and is related to the sequence context, selection of desired metrics relies on specific purposes. A discussion about selecting metrics for accuracy evaluation is covered in detail by [32].

Ideally, an accurate variant detection method should perform with high sensitivity and specificity with a low false discovery rate to identify variants within a practical level of mutant allele frequencies. Since it is difficult for statistical methods to be perfectly accurate in all circumstances, at a minimum, reliable variant detection methods should be capable of detecting true variants by keeping the false positive rate as low as possible within an acceptable accuracy for a particular dataset of interest.

Table 2.1: Metrics for accuracy evaluation of variant detection methods.

Metrics	**Explanation**	**Derivation**
Sensitivity	true positive rate	$\frac{TP}{TP+FN}$
Specificity	true negative rate	$\frac{TN}{FP+TN}$
FPR	false positive rate	$\frac{FP}{FP+TN}$
FDR	false discovery rate	$\frac{FP}{FP+TP}$
PPV	positive predicted value or precision	$\frac{TP}{TP+FP}$
NPV	negative predicted value	$\frac{TN}{TN+FN}$
ACC	accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$
MCC	Matthews correlation coefficient	$\frac{TP*TN-FP*FN}{\sqrt{P1*P2*N1*N2}}$

TP, true positive; FP, false positive; TN, true negative; FN, false negative; P1, TP+FP; P2, TP+FN; N1, TN+FP; N2, TN+FN.

2.2.2 Scalability

Much effort is being devoted to improve scalability in analyzing a broad sequence coverage. For example, reliable variant detection within a genome-wide sequence scale would surpass traditional low-coverage diagnostics of a few specific sites. Since detecting variants in WGS and WES data sets is mostly time-consuming, application of large-scale genomic data is hindered by lack of scalable and efficient algorithms. Advanced statistical methods with high scalability are more than desirable to detect true variant alleles.

Performing statistical inference and estimation for statistical variant detection methods, such as Bayesian and heuristic methods, requires a process of drawing conclusions of abnormally high non-reference variants from large sequencing data. The Markov Chain Monte Carlo (MCMC) sampling algorithm is comprehensively used for statistical inference, but the inherent limitation of MCMC is that the time for converging is long and the convergence can be hard to diagnose. Alternatively, a variational approximation algorithm converges faster than the MCMC sampling algorithm because it yields deterministic approximation that provides bounds on probability of interest, which accelerates the variant detection process [33]. For example, the mean field algorithm is a type of variational approximation method that has been demonstrated to be 10 to 30 times faster than MCMC sampling while producing the same accuracy [34]. A variational algorithm is used to estimate the model for chemogenomic profiling [35]. However, even though sampling-based methods are computationally slow, they can be easy distributed to multiple computing cores in parallel for faster yield. This type of trade-off between accuracy and scalability is expected, since accurate solutions may often be balanced with efficiency.

2.2.3 Robustness

The design of statistical methods typically limits the number of errors received in variant detection. Method robustness is important to evaluate the ability of variant detection in the presence of noise and contaminated sequencing samples. For example, robust Bayesian analysis is often conducted to study the uncertainty of prior distributions for model robustness assessment. A prior assignment in an empirical Bayesian method can be subjective, which may cause bias in the probabilistic distribution prediction of the allele frequency for a site in the sequence. This will result in the miscalling of a variant when comparing the allele frequencies of one site in a control to the case pairing. A good variant detection method should be insensitive to changes of priors or parameters of the model system that should generate consistent results.

2.3 Factors that affect the ability of variant detection

Next-generation sequencing data is massive and heterogeneous and many factors could influence the performance of variant detection methods, which we outline and discuss below.

2.3.1 Quality control

The quality of the data can effect variant detection, so checking the quality of the raw data and filtering the low-confidence alleles in advance will improve the accuracy of variant detection. FastQC is a standard tool that has been implemented for assessing the quality of data by generating analytical graphs. Also, low-confidence alleles can be trimmed using a standalone tool, NGS QC Toolkit [36], to prevent from making wrong variant calls. Although filtering low-confidence alleles helps with read alignment, it is possible that false positives could be introduced for high-coverage data sets [37]. Therefore, it is important to also consider the read depth of coverage in order to ensure the accuracy of variant detection in addition to quality control in the read mapping step.

2.3.2 Depth of coverage

Sequencing depth of coverage, number of times that each base pair has been sequenced, contributes to the overall result from variant detection methods because sufficient depth of coverage is necessary to support an accurate variant call. Due to the relatively high cost of sequencing, the depth of coverage is typically low (less than $10\times$) and the distribution of the read depth over each site is not generally uniform. Low depth coverage will pose the challenge for detecting the low fraction of circulating cfDNA from tumor cells and the normal DNA in the blood may further complicate the variant detection. Previous studies have shown the effect of coverage and revealed that high coverage data normally leads to high sensitivity for variant detection [38, 39, 40]. The false discovery rate of variant detection using GATK [17] decreased as the depth of coverage increased [41]. Generally, a minimum coverage for a heterozygote non-reference allele is $20\times$ [42]; the minimum coverage for a single nucleotide polymorphism is $50\times$, while some applications may need higher coverage [43]. It is reasonable that if you desire to detect a rare variant of 0.1% allele frequency, the required depth of coverage is $1,000\times$. Furthermore, unmatched sequencing read depths of case and control samples will generate increased false positives [44].

2.3.3 Sequencing errors

Intrinsic errors from next-generation sequencing platforms exist during sample processing and sequencing [32]. The non-reference allele errors in the process of library preparation, PCR amplification, and sample sequencing are not in an uniform distribution due to the influence of the operation of sequencing-by-synthesis [45, 46]. The presence of insertions/deletions, structural variants, and copy number variations may introduce false positive variants that pose the challenge for accurate variant detection [47]. It is especially critical when identifying variants with a minor variant allele frequency (VAF) making it difficult to differentiate a true rare variant (VAF $< 1\%$) from a common sequencing error. A common sequencing error rate is reported to be from 1% to 3% in the initial release of raw sequencing data [48]. However, when evaluating a synthetic DNA sequencing data set the sequencing error rate using NGS technology is less than what is reported for non-synthetic samples [13], which makes detection more difficult. To estimate the sequencing errors and show accurate estimation of error rate on mock microbial genetic marker sequencing samples, a web server, NGS-eval [49], was developed. This application helps estimate the sequencing quality of the NGS data and quantify the ability of variant detection methods.

2.3.4 Sample size

Pooled sequencing on multiple samples has enabled the identification of more rare variants than individual samples [9, 37]. It has been shown that when comparing strategies of single and multiple samples [41], GATK gives higher sensitivity for variant detection in a multiple-sample strategy than a single-sample strategy, but the specificity decreased. The reason may be that more false positives are called in larger data sets with the multiple samples [23]. However, if the coverage of the multiple samples is low, the false discovery rate for variant detection could increase compared to the case when using high coverage [40]. Another observation [50] showed that a large data set of multiple sequencing samples at low coverage (4-6$\times$) yields higher capability of rare variant detection compared to a small data set of less sequencing samples at high coverage.

2.4 Classification for variant detection methods

In this section, we classified the state-of-the-art variant detection methods into two categories: probabilistic methods, and non-probabilistic, or other combination. We outlined the category and subcategory, functions, platform, and software implement for each method.

2.4.1 Probabilistic methods

The underlying concept of probabilistic methods for variant detection is by modeling uncertainty given the sequencing data. To understand how probabilistic methods are developed for variant detection, we summarize 25 variant detection methods that are built based on the probabilistic strategies. We discuss each method from three aspects: specific purpose of the method, method category, and metrics or related applications. A summary of the state-of-the-art probabilistic methods for variant detection is shown in Table 2.2.

GATK [17] is designed for detecting germline variants in homogeneous samples. It uses a simple Bayesian genotyper to calculate the posterior distribution of each genotype given mapped reads over each site [51]. It adopted a MapReduce system to facilitate processing large-scale sequencing data in parallel and has been involved in The 1000 Genomes Project and The Cancer Genome Atlas.

MuTect [16] is a method to detect germline and somatic variants with low allele frequencies at various sequencing read depths in mixed tumour samples. It is built on a Bayesian classifier to calculate a log-likelihood ratio that can be used as a threshold for variant detection in matched tumour and normal samples. It has been shown that MuTect is more sensitive than other competing methods in detecting somatic variants within low fraction of tumour cells, which enables us to discover subclonal drivers for tumour progression.

Mapping and assembly with quality (**MAQ**) [52] is a probabilistic method that uses a fixed prior for estimation of non-reference allele probabilities. **SAMtools** [53] is a revised MAQ model to manipulate genomic sequences in the SAM and BAM format. Similar to GATK, SAMtools computes the likelihood of each possible genotype using a naive Bayesian model to then identify germline variants using BCFtools [54]. It has been demonstrated for comparable accuracy in real data for allele count estimation, allele frequency estimation, and association mapping. **glftools** [55] is a revised version of SAMtools to generate genotype likelihood files. Single individual, glfSingle and multiple individuals, glfMultiples are developed for genotype calling as well.

FamSeq [56] is a family-based sequencing program for variant detection in data derived from family members. FamSeq uses a Bayesian network to yield posterior probabilities for measure of genotype calls and a MCMC sampling method to derive posterior probabilities. This method integrates Mendelian inheritance and sequencing data of family members to reduce false positives and false negatives for variant detection.

JointSNVMix [57] was developed to discover somatic variants and to distinguish germline from somatic events. It applies two novel Bayesian probabilistic models to jointly analyze the allelic count of tumour and normal samples. Concordance is used as a probabilistic threshold to measure the performance of variant detection. It has been demonstrated that joint modeling, JointSNVMix, has higher specificity than its independent analogue with guaranteed sensitivity.

Table 2.2: A summary of the state-of-the-art probabilistic methods for variant detection and the category classifications of them.

Category	Subcategory/Method	Functions	Platform	Source Code	Ref
Probabilistic	**Bayesian Decision Rules**				
	GATK	SNVs, indels	Java	https://www.broadinstitute.org/gatk/	[17]
	Mutect	SNVs	Java	http://www.broadinstitute.org/cancer/cga/mutect	[16]
	SAMtools	SNVs, indels	C	http://samtools.sourceforge.net/	[53]
	FamSeq	SNVs	C++	http://bioinformatics.mdanderson.org/main/FamSeq	[56]
	MAQ	SNVs	Perl	http://maq.sourceforge.net/	[52]
	JointSNVMix	SNVs	Python	http://compbio.bccrc.ca/software/jointsnvmix/	[57]
	Bayesian				
	Virmid	SNVs	Java	https://sourceforge.net/projects/virmid/	[58]
	EM-SNP	SNVs	R	http://www-rcf.usc.edu/ fsun/Programs/EM-SNP/EM-SNP.html	[59]
	SomaticSniper	SNVs	Perl	http://gmt.genome.wustl.edu/packages/somatic-sniper/	[19]
	Strelka	SNVs, indels	Perl	https://sites.google.com/site/strelkasomaticvariantcaller/	[18]
	RVD/RVD2/VI RVD	SNVs	Python	http://genomics.wpi.edu/rvd2/	[31]
	FreeBayes	SNVs, indels	Python	https://github.com/ekg/freebayes	[60]
	EBCall	SNVs, indels	R	https://github.com/friend1ws/EBCall	[20]
	DeepSNV	SNVs	R	http://www.bioconductor.org/packages/release/bioc/html/deepSNV.html	[24]
	EBM	SNVs	R	https://sites.google.com/site/zhouby98/ebm	[61]
	Snape	SNVs	C	https://code.google.com/archive/p/snape-pooled/	[62]
	SNVMix	SNVs	C	http://compbio.bccrc.ca/software/snvmix/	[63]
	SOAPsnp	SNVs	C, C++	http://soap.genomics.org.cn/soapsnp.html	[64]
	Seurat	SNVs, indels	Java	https://sites.google.com/site/seuratsomatic/	[25]
	Likelihood-based				
	glfTools	SNVs	C, C++	csg.sph.umich.edu//abecasis/glfTools/	[55]
	PolyMutt	SNVs, indels	C++	http://genome.sph.umich.edu/wiki/Polymutt	[65]
	Large Deviation Theory				
	SNPSeeker	SNVs	C	http://genetics.wustl.edu/rmlab/software/	[66]
	SPLINTER	SNVs, indels	C, C++	available on request	[67]
	Linear Classifier				
	QQ-SNV	SNVs	Perl	https://sourceforge.net/projects/qqsnv/	[68]
	Contingency Table				
	CRISP	SNVs	Ptython	https://sites.google.com/site/vibansal/software/crisp	[69]

Category and subcategory of each variant detection method is classified. SNVs, single nucleotide variants; indels, insertions/deletions. Functions for identifying SNVs and indels are distinguished. Platforms of language and source code are provided.

Strelka [18] is an algorithm for somatic variant detection through a joint analysis of matched tumour and normal samples. It is a Bayesian method that models the joint probabilistic distribution of continuous allele frequencies. Strelka is capable of maintaining high sensitivity in low purity tumour samples.

Virmid [58] has been implemented for somatic variant detection by estimating the level of sample contamination. Maximum likelihood estimation is used for sample impurity estimation and joint genotype probability estimation, and Bayesian inference is used for variant detection by Virmid. The strategy of estimating the level of contamination helps to increase computational speed and accuracy.

EM-SNP [59] can be used for allele frequency estimation, SNVs detection, and association study in pooled sequencing data. Their team developed an expectation maximization algorithm to approximate the maximum likelihood of the parameters for estimation of minor allele frequencies. It has been shown that EM-SNP outperforms SNVer [70] in rare variants detection in type 1 diabetes pooled sequencing data by comparison of dbSNP and transition/-transversion ratio metrics.

SomaticSniper [19] detects somatic variants by directly comparing the joint diploid genotype likelihoods for a tumour-normal pair. The genotype likelihood is calculated by the MAQ method [52], which incorporates the dependency of the genotypes between tumour and normal samples. Sensitivity and precision were used to evaluate the performance of variant detection on a simulated data set.

FreeBayes [60] is a haplotype-based variant detection method for short read DNA sequencing data. It is a generalization of a Bayesian statistical method [71] to detect variants in both individual and pooled samples. A gradient ascent method is employed to establish a maximum a posteriori estimate of the genotype for each sample. This framework is able to identify longer and multi-alleles by modeling multiallelic sites.

EBCall [20] is proposed to detect somatic variants by purposely incorporating sequencing errors as prior information into the model. It is developed based on an empirical Bayesian framework where a beta-binomial distribution is used to depict sequencing errors. EBCall has been shown to detect somatic variants of less than 10% allele frequencies in tumour subclones.

DeepSNV [24] is a powerful statistical method for detecting SNVs in ultra-deep sequencing data. This algorithm is built on a hierarchical beta-binomial model, and a likelihood ratio test is calculated for each base for comparison with a control or a reference sequence. DeepSNV is validated on subclonal diverse tumour samples of renal cell carcinoma and has revealed an agreement of variant allele frequencies of the variants found by the original work [24].

EBM [61] aims to detect SNVs in pooled sequencing data by accurately estimating the sequencing error distribution across multiple sequencing pools and genomic positions. It is a empirical Bayes mixture model that uses an expectation-conditional maximization (ECM) algorithm for model inference and parameter estimation. Its usability was demonstrated with lower sum of squared errors of the estimated allele frequencies compared with a naive estimator.

Snape [62] is built to detect SNVs in pooled samples. It is a Bayesian method that considers different priors to estimate the posterior frequency probability of SNVs. Snape outputs a low false discovery rate with high power on a simulated data set generated by ART [72].

SNVMix [63] is a probabilistic method to detect SNVs from tumour NGS data. It is developed based on Binomial mixture models to which an expectation maximization algorithm is used to obtain model parameters to predict allele frequencies. It demonstrates high sensitivity and specificity using a breast cancer data set of $> 40\times$ of which the ground truth of SNVs is known.

SOAPsnp [64] is a variant detection method designed for massively parallel sequencing-by-synthesis Illumina Genome Analyzer data. It uses a Bayesian statistical model to infer the likelihood of each possible genotype and outputs the genotype with highest posterior probability for each site. It achieves high accuracy in human genome deep resequencing data. Incorporating dbSNP genotypes as prior information into SOAPsn helps identify real heterozygotes in low read depth data.

Seurat [25] aims to detect somatic events, including SNVs, insertion/deletions, and structural variations, within tumours in paired tumour and normal samples. Seurat is a generalized Bayesian framework that uses a beta-binomial distribution to model the probability of a somatic event. Seurat outputs a high transition/transversion ratio, low non-synonymous/synonymous ratio, and low dbSNP rate on a lymphoma tumour data set.

PolyMutt [65] is a likelihood-based method to detect novel SNVs in samples of families. PolyMutt models the likelihood of reads in pedigrees using the Elston-Stewart algorithm [73]. In a simulation study, it was shown that the information from families helps improve the specificity of SNVs detection.

SNPSeeker [66] uses large deviation theory to detect SNVs from a large pool of multiple individuals. Negative control data is necessary for model estimation and it is capable of detecting SNVs that present with lower allele frequencies than the error rate of the sequencing platform.

SPLINTER [67], like SNPSeeker, is also based on the large deviation theory to detect rare alleles in pooled sequencing samples. This research shows that over $500\times$ coverage is guaranteed as an ideal performance of detecting low frequency variants ($25\% - 2.5\%$) on a synthetic DNA mixture of HapMap samples. It was also shown that SPLINTER has acceptable specificity and positive predictive value (PPV) in clinical sequencing regions.

QQ-SNV [68] was developed to differentiate true SNVs from errors using the quartiles from quality scores. Instead of modeling the position-specific allele frequency, a logistic regression classifier model is used to classify a position as a variant or an error by incorporating the Illumina quality scores. QQ-SNV shows a sensitivity of 100% and a specificity of 100% when tested on a paired-end HCV clinical sample where the true frequency of the lowest spiked-in is 0.5%.

CRISP [69] identifies both rare and common variants in pooled sequencing samples. It is a probabilistic method that computes a contingency table P-value and a quality-based P-value that represent the probability of the absence of a variant, which can be used to differentiate true variants from sequencing errors. CRISP is able to detect a 2% allele frequency event in a data set of two pools with 25 individuals each.

We previously developed a beta-binomial model, **RVD** [13], to characterize error rate distribution of each site of next-generation sequencing data. RVD is able to detect a 0.1% variant allele frequency event in a synthetic DNA data set. From this, we developed **RVD2** [31] that improved RVD by adding priors to tie parameters across sites and derived a Markov Chain Monte Carlo sampling algorithm for posterior inference. Based on this improvement, RVD2 can handle low read depth sequencing data and manipulate multiple replicates. Furthermore, we proposed a variational inference expectation maximization algorithm, **VI RVD** [74], for the former Bayesian statistical model to detect single nucleotide variants in heterogeneous samples. The variational inference algorithm demonstrates comparable sensitivity and specificity compared to other state-of-the-art algorithms and can track non-reference allele frequency in a real time-series sequencing data set.

2.4.2 Non-probabilistic methods

In this section, we summarize five variant detection methods that were developed based on non-probabilistic or other combination methodologies (Table 2.3).

Table 2.3: A summary of the state-of-the-art non-probabilistic methods for variant detection and the category classifications of them.

Category	Subcategory/Method	Functions	Platform	Source Code	Ref
Non-probabilistic	**Frequentist Method**				
or	SNVer	SNVs	Java	http://snver.sourceforge.net/	[70]
other cambination	**Heuristic Method**				
	VarScan2	SNVs, indels	Java	http://varscan.sourceforge.net/	[21]
	Shimmer	SNVs	Perl	https://github.com/nhansen/Shimmer	[22]
	Machine Learning				
	Atlas2	SNVs, indels	Ruby	https://sourceforge.net/projects/atlas2/	[75]
	Feature-based methods	SNVs	Python	http://compbio.bccrc.ca/software/mutationseq/	[76]
	Cake	SNVs	Perl	http://cakesomatic.sourceforge.net/	[77]

Category and subcategory of each variant detection method is classified. SNVs, single nucleotide variants; indels, insertions/deletions. Functions for identifying SNVs and indels are distinguished. Platforms of language and source code are provided.

SNVer [70] is a common and rare variant detection method for both individual and pooled sequencing data that is scalable for whole genome sequencing data. This is a frequentist method that reports overall P-values for every site without discarding bases with low read depth. SNVer uses transition/transversion ratio, genotype concordance, and dbSNP as metrics for evaluating the quality of variant calls in a real pooled sequencing data [51].

VarScan2 [21] has been demonstrated for detecting somatic variants, loss of heterozygosity, and germline variants in exome sequencing data. VarScan2 uses a heuristic method and a Fisher's exact test by comparing tumour and normal samples based on several thresholds, such as the number of allele counts and variant allele frequency. In a test of 151 ovarian tumour samples from the TCGA data set, VarScan2 identified 7,790 validated somatic variants with 93% sensitivity and 85% precision.

Shimmer [22] is a method to identify somatic changes in normal-tumour samples. It uses a Fisher's exact test with Benjamini-Hochberg [78] for multiple testing correction to control the false discovery rate (FDR). This method is sensitive enough for the detection of variants in highly heterogeneous stromal contaminated data.

Atlas2 [75] aims to detect SNVs in whole exome sequencing (WES) data from the platforms of SOLiD, Illumina, and Roche 454. Atlas2 uses a logistic regression model to detect SNVs that pass several heuristic filters. It has been integrated into the Genboree to streamline the processing of next-generation sequencing data on a web-based platform.

Feature-based classifiers [76], such as random forests, Bayesian additive regression trees, support vector machines, and logistic regression, can also be used to detect somatic variants in tumour-normal paired data sets. Supervised machine learning algorithms are trained on the ground truth data set of ~ 3400 positions from 48 breast cancer exome sequences and a cross-validation analysis is used to measure accuracy. This development shows that the feature-based machine learning algorithms outperform SAMtools and GATK in both sensitivity and specificity in this synthetic data set.

Cake [77] integrates four variant detection methods, Bambino, CaVEMan, SAMtools-mpileup, and VarScan2, together with post-processing filters to detect somatic variants. Cake outperforms any single algorithm with higher accuracy on two data sets, human hepatocellular carcinoma data and human breast cancer exome data, and also functions with WGS and WGE sequencing data using a standalone application.

2.4.3 Overall Bayesian framework for variant detection

Most current variant detection methods are developed based on Bayesian Theorem. Because of this, we describe an overall workflow for developing Bayesian-based methods for variant detection in Figure 2.2. First, control (normal sample) and case (tumour sample) sequencing data are commonly required to identify variants in the case. Then, control and case samples will be fed into a Bayesian probabilistic model independently for comparison. In addition to the sample data, a prior probabilistic distribution is needed to be defined for empirical Bayesian models. The prior distribution captures the population knowledge of the parameters, like genotype information of the data, within the Bayesian statistical model. Since the prior distribution affects the posterior distribution substantially, comparing posteriors under different plausible choices of prior distribution is important. Once a model is defined, model inference via Bayes' rule based on the prior and the likelihood of the data is implemented, and the posterior distribution is a compromise of the prior distribution and the data. As an outcome of model inference, posterior distributions for control and case are obtained to detect variants. Finally, variants are detected by hypothesis testing between the posterior distributions of control and case. Since hypothesis testing across regions of interest is multiple and independent for each site, appropriate error rate control is needed. The Benjamini-Hochberg method for false discovery rate (FDR) control and Bonferroni correction for family-wise error rate (FWER) are routinely used in the large-scale testing problems [79].

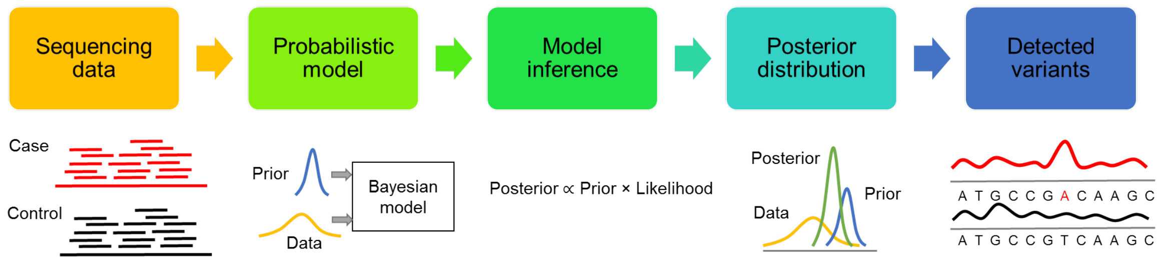


Figure 2.2: Bayesian probabilistic framework for variant detection in NGS sequencing data. The control sample is shown in black and the case sample is shown in red. The posterior distribution (green) is propotional to the prior distribution (blue) and the likelihood of the data distribution (yellow).

A major benefit of Bayesian-based methods is the flexibility of multiple levels of random variables in measuring sources of uncertainty of underlying genotypes. The Bayesian framework has become a prevailing method to identify variants in NGS data, but several difficulties still remain. First, computational calculation for Bayesian inference is normally difficult in the integration step. Also, accurate specification of prior probability in the model structure and model evaluation for fitness assessment are challenging.

2.4.4 Advantages and disadvantages of different methods

Current efforts are focusing on comparing the performance of current variant detection methods in different applications based on specific interests and requirements. We summarize eight comparative analyses on variant detection methods in tumour-normal paired NGS data in Table 2.4. We propose that this summary of method comparisons is helpful to select appropriate methods for their specific purposes. These comparisons focus mostly on the performances of VarScan, MuTect, GATK, SAMtools, JointSNVMix, SomaticSniper, EBCall, and Strelka. Synthetic or real WGS/WES data are used as benchmarking data sets for method validation. Results of this comparative analysis show that SAMtools and GATK are not able to call rare variants in heterogeneous samples. VarScan2 can detect more variants than other methods, but may yield more false positives [80, 67], while Strelka returns a low number of variants in a conservative way [81]. Although Strelka is able to call variants in heterogeneous samples, it has lower accuracy than MuTect [80]. EBCall and MuTect are robust to the changing of read depth and perform very well in a study of somatic variant detection in real exome and targeted deep sequencing data [81]. Overall, MuTect outperforms other tested methods in detecting rare variants with low allele frequency and shows high sensitivity at different dilutions [80, 29]. Since performance depends on the quality of specific data sets and the ability of methods in calling variants in heterogeneous samples, it is thought that a combination of several state-of-the-art methods could be used together to generate an intersection of sets of the candidate variants for consistency [41]. Previously, a unified framework has been provided for combining the detected variants from multiple variant detection methods with improved performance over each individual method [82].

Table 2.4: A summary of comparative analysis of variant detection methods.

Benchmarking data sets	**Compared methods**	**Advantages/Disadvantages**	**Ref**
Illumina WES CML	VarScan JointSNVMix SomaticSniper, Strelka	Unable to detect rare variants of low VAFs. Vulnerable to FPs. Gave the best performance.	[15]
Illumina WGS Melanoma Illumina WES lung tumour	VarScan2 MuTect EBCall, JointSNVMix, Strelka, VarScan2	Detected more somatic variants than others. Detected most somatic variants of low VAFs. Had lower accuracy compared with VarScan2 and MuTect.	[80]
Synthetic WGS	GATK SAMtools, glftools, Atlas2	Showed highest specificity and Ti/Tv ratio at single-sample. Showed higher sensitivity than SAMtools at multiple-sample.	[41]
Illumina WGS SSMP with rare, low, and common VAFs at $5\times$, $10\times$, $20\times$, and $30\times$	GATK, SAMtools CASAVA VarScan, glftools, SOAPsnp	Had the lowest FDR at $5\times$ read depth. Had highest accuracy at $30\times$ read depth.	[40]
Synthetic DNA mixtures with 2.5% to 25% VAFs at $> 1000\times$	SAMtools GATK, SPLINTER VarScan2	Had the lowest sensitivity. Detected more than 94% of variants with 10% VAFs. Yielded more FPs at high read depths.	[67]
WES with VAFs of 8%, 16%, 36%, and 50% Amplicon with VAFs of 8%, 16%, 36%, and 100%	MuTect, Strelka GATK, SomaticSniper, VarScan2	Showed highest sensitivity. Sensitivity changed with VAFs.	[29]
Synthetically pooled Illumina sequencing	GATK, CRISP, LoFreq VarScan, SNVer	Showed 80% accuracy with different read depths. Showed lower FPR but lower sensitivity.	[30]
Breast cancer exome sequencing Targeted deep sequencing	EBCall, MuTect, Virmid, Strelka Shimmer, Somatic Sniper, VarScan2, Seurat	Reported them to be the most reliable callers; Returned the lowest number of variants. EBCall and Mutect are robust to varying of read depth. Showed highly sensitive to increased depths; Returned high numbers of variants in low agreement with others. Seurat showed the highest sensitivity of 96%.	[81]

The order of these methods are based on the published date. CML, chronic myeloid leukemia; WGS, whole genome sequencing; FPs, false positives; VAFs, variant allele frequencies; FDR, false discovery rate; WES, whole exame sequencing; SSMP, Singapore sequencing malay project; CASAVA, consensus assessment of sequence and variation; Rare, VAF $< 1\%$; Low, $1\% \leqslant$ VAF $< 5\%$; common, VAF $\geqslant 5\%$; FPR, false positive rate; Sensitivity, true positive rate; Ti/Tv, transition/transversion ratio.

2.4.5 The potential of deep learning in variant detection

Recently, deep learning has become involved in addressing limitations in computational biology, and multiple layers of the neural networks are proposed to dissect the structure of high-throughput sequencing data [83]. Most current statistical methods are developed based on prior knowledge for sequencing data analysis. For example, a prior probability was predefined for the allelic diversity for Snape [62]. In variant detection using SOAPsnp, the prior for homozygous variant rate is set as 0.0005, and 0.001 for the heterozygous rate [64]. The prior probabilities were also set for both haploid and diploid genotypes according to the transition and transversion values in the study of dbSNPs alleles [84]. Contrarily, deep learning networks can learn directly from the data without assigning a prior. A deep learning method, called *DeepBind*, was proposed to uncover the role of DNA/RNA binding proteins and also able to discover disease-related variants in sequencing data [85]. Deep learning methods may be advantageous for enabling direct training of the model on the sequencing samples. Therefore, deep learning could be a promising direction for variant detection to yield more accurate predictions.

2.5 Conclusions

In this review, we addressed the necessity of sensitive variant detection methods and common challenges for rare SNVs detection in DNA NGS data. We classified 30 state-of-the-art methods to different categories based on the methodology. A Bayesian framework, as a leading method for current variant detection, was summarized to guide current researchers during the development of probabilistic statistical methods for variant detection. Finally, we anticipate that this review will help further the understanding of current methods for rare variant detection for biologists and bioinformaticians alike, and accelerate the interpretation of next-generation sequencing data involved with clinical studies and genetic-based research.

Single nucleotide variant detection using statistical methods is being actively developed in the characterization of genomic heterogeneity in clinical samples. A number of statistical or computational methods have been developed for variant detection in the NGS data, but high sensitive variant detection methods are still in demand. As we discussed in this chapter, the biological and statistical limitations that hinder effective variant detection are points of focus for developing sensitively accurate, scalable, and robust methods. Current methods have progressed quickly from standard cut-off threshold methods, genotypes subtraction methods, to now advanced Bayesian statistical methods. Among these advanced methods, the probabilistic statistical pipelines are built to handle uncertainty in probabilistic analogues. The un-probabilistic methods are employing and evaluating the frequentist, heuristic, and other machine learning methods. Regardless, each method type has mathematical problem formulation as well as their inherent advantages and disadvantages; hence, accurate variant detection highly relies on the characteristics of the driving statistical methods. In all, the development of more sensitive statistical methods will remain an essential factor in the fundamental research of human genetics for clinical disease monitoring and the advancement of treatments.

Chapter 3

Variational Inference for Rare Variant Detection in Heterogeneous Next-generation Sequencing Data

3.1 Background

Massively parallel sequencing data generated by next-generation sequencing technologies is routinely used to interrogate single nucleotide variants (SNVs) in research samples [86]. For example, deep sequencing confirmed the degree of genetic heterogeneity of HIV and influenza [13, 87]. Intra-tumor heterogeneity has been revealed by next-generation sequencing [88]. Whole genome sequencing has revealed that many beneficial mutations of minor allele frequencies are essential to respond to dynamic environments [89]. However, rare SNV identification in heterogeneous cell populations is challenging, because of the intrinsic error rate of next generation sequencing [48]. Thus, there is a need for accurate and scalable statistical methods to uncover SNVs in heterogeneous samples.

A number of computational methods have been developed to detect SNVs in large scale genomic data sets. These methods can be roughly categorized as probabilistic or heuristic or some combination. Among all of the current probabilistic methods, the Bayesian probabilistic framework has been increasingly used to estimate unobserved quantities such as variant allele frequency given observed genomic sequencing data.

GATK [17] and SAMTools [53] use a naive Bayesian decision rule to call variants. EBCall models sequencing errors based on a Beta-Binomial distribution, where the parameters and latent variables are estimated from a set of non-paired normal sequencing samples [20]. However, the error rate of normal sequencing samples could be unmatched with the error rate of the target samples, which may cause a problem of making false negatives calls [80]. CRISP compares aligned reads across multiple pools to obtain sequencing errors, and then distinguishes true rare variants from the sequencing errors [69]. However, the bottleneck of CRISP is its low computational efficiency due to a calculation of a large number of contingency tables. JointSNVMix introduces two Bayesian probabilistic models (JointSNVMix1 and JointSNVMix2) to jointly analyze a tumour-normal paired allelic count of NGS data [57]. JointSNVMix derives an expectation maximization algorithm to calculate maximum a-posteriori (MAP) estimate of latent variables in a particular probabilistic graphical model. Furthermore, they showed that the joint modeling method, JointSNVMix1, observes 80-fold reduction of false positives compared with its independent analogue (SNVMix1) [57]. SomaticSniper models the joint diploid genotype likelihoods for both tumour and normal samples [19]. Strelka models the joint probabilistic distribution of allele frequencies for both tumour and normal samples, which is demonstrated to be more accurate compared with the methods based on the estimated allele frequency tests between tumour and normal samples [90]. SNVer focuses on a frequentist method that is able to calculate P-values, but [70] pointed out that this approach fails to model sampling bias that will reduce the power of detecting true rare variants. VarScan compares tumour and normal samples thresholding on variant allele frequency and a number of allele counts then uses Fisher's exact test to estimate sample allele frequencies [91].

In previous work, we developed a Beta-Binomial model to estimate a null hypothesis error rate distribution at each position. Using this rare variant detection (RVD) model, we call rare variants by comparing the error rate of the sample sequence data to a null distribution obtained from sequencing a known reference sample [13]. RVD can identify mutant positions at a 0.1% fraction in mixed samples using high read depth data.

An improvement of that work, RVD2, uses hierarchical priors to tie parameters across positions to detect variants in low read depth data [31]. We derived a Markov Chain Monte Carlo sampling algorithm for posterior inference. However, the main limitation of MCMC is that it is hard to diagnose convergence and may be slow to converge [33]. An alternative inference method, that we explore here, is to use variational inference, which is based on a proposed variational distribution over latent variables. By optimizing variational parameters, we fit an approximate distribution that is close to the true posterior distribution in the sense of the Kullback-Leibler (KL) divergence. Variational inference can now handle nonconjugate distributions and tends to be more computationally efficient than MCMC sampling [34].

Here, we propose a variational EM algorithm for our Bayesian statistical model to detect rare SNVs in heterogeneous NGS data. We show that variational EM algorithm has comparable accuracy and efficiency compared with MCMC in a synthetic data set. In Section 3.2, we define the model structure, and derive our variational EM algorithm to approximate the posterior distribution over latent variables. Then, we call a variant by a posterior difference hypothesis test between the key model parameters of a pair of samples. In Section 3.3, we compare the performance of the variational EM inference algorithm to the MCMC sampling method and the state-of-the-art methods using a synthetic data set. We also show that our variational EM algorithm is able to detect rare variants and estimate non-reference allele frequency (NRAF) in a longitudinal directed evolution experimental data set.

3.2 Methods

3.2.1 Model structure

Our Bayesian statistical model is shown as a graphical model in Figure 3.1A. In the model, r_{ji} is the number of reads with a non-reference base at location j in experimental replicate i; n_{ji} is the total number of reads at location j in experimental replicate i. The model parameters are:

– μ_0 a global non-reference read rate that captures the error rate across all the positions,
– M_0 a global precision that captures the variation of the error rate across positions in a sequence, and
– M_j a local precision that captures the variation of the error rate at position j across different replicates.

The latent variables are:

– $\mu_j \sim \textbf{Beta}(\mu_0, M_0)$ a position-specific non-reference read rate for position j, and
– $\theta_{ji} \sim \textbf{Beta}(\mu_j, M_j)$ the non-reference read rate for position j in replicate i.

In Figure 3.1B, γ is the parameter for the variational distribution for latent variable μ, and δ is the parameter for the variational distribution for latent variable θ. We describe $q(\mu)$ and $q(\theta)$ in detail in section 2.2.2.

The model generative process is as follows:

1. For each location $j \in [1, \ldots, J]$:

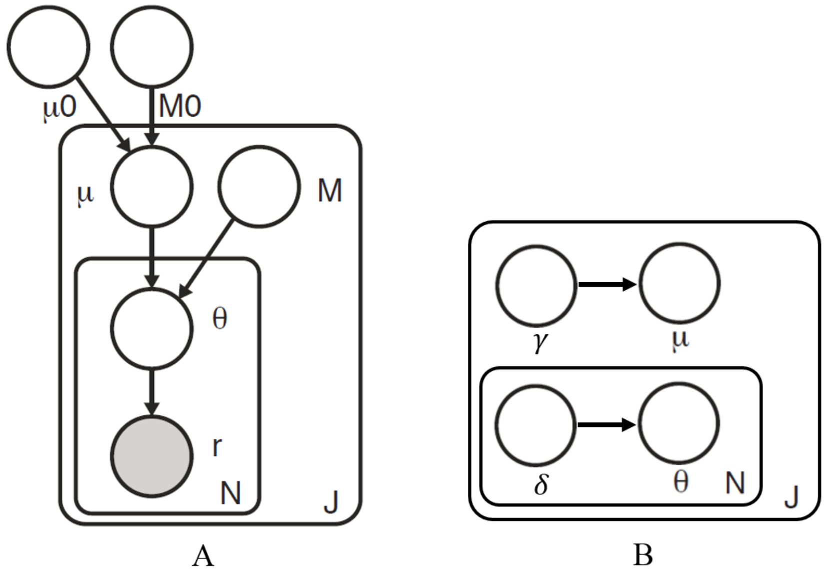


Figure 3.1: Graphical model. A. Graphical model representation of the model. B. Graphical model representation of the variational approximation to approximate the posterior distribution. Observed random variables are shown as shaded nodes and latent random variables are unshaded. The object of inference for the variational EM algorithm is the joint distribution $p(\mu, \theta | r, n)$.

(a) Draw an error rate $\mu_j \sim \text{Beta}(\mu_0, M_0)$

(b) For each replicate $i \in [1, \ldots, N]$:

  i. Draw $\theta_{ji} \sim \text{Beta}(\mu_j, M_j)$

  ii. Draw $r_{ji} | n_{ji} \sim \text{Binomial}(\theta_{ji}, n_{ji})$

The joint distribution $p(r, \mu, \theta | n; \phi)$ given the parameters can be factorized as

$$p(r, \mu, \theta | n; \phi) = p(r|\theta, n) p(\theta|\mu; M) p(\mu; \mu_0, M_0). \tag{3.1}$$

3.2.2 Variational expectation maximization (EM) inference

We developed a non-conjugate variational inference algorithm to approximate the posterior distribution,

$$p(\mu, \theta | r, n; \phi) = \frac{p(r, \mu, \theta | n; \phi)}{p(r|n; \phi)}, \tag{3.2}$$

where the parameters are $\phi \triangleq \{\mu_0, M_0, M\}$.

3.2.2.1 Factorization

We propose the following factorized variational distribution to approximate the true posterior over latent variables μ_j and θ_{ji}. Here, $q(\mu_j)$ approximates the variational posterior distribution of μ_j, which represents the local error rate distribution at position j across different replicates; and $q(\theta_{ji})$ approximates the posterior distribution of θ_{ji}, which is the error rate distribution at position j for replicate i.

$$q(\mu, \theta) = q(\mu)q(\theta) = \prod_{j=1}^{J} q(\mu_j) \prod_{i=1}^{N} q(\theta_{ji}). \tag{3.3}$$

3.2.2.2 Evidence lower bound (ELBO)

Given the variational distribution, q, the log-likelihood of the data is lower-bounded according to Jensen's inequality,

$$\begin{aligned} \log p\left(r|n;\phi\right) &= \log \int_\mu \int_\theta p\left(r,\mu,\theta|n;\phi\right) d\theta d\mu \\ &= \log \int_\mu \int_\theta p\left(r,\mu,\theta|n;\phi\right) \frac{q\left(\mu,\theta\right)}{q\left(\mu,\theta\right)} d\theta d\mu \\ &\geqslant \int_\mu \int_\theta q\left(\mu,\theta\right) \log \frac{p\left(r,\mu,\theta|n;\phi\right)}{q\left(\mu,\theta\right)} d\theta d\mu \\ &= E_q\left[\log p\left(r,\mu,\theta|n;\phi\right)\right] - E_q\left[\log q\left(\mu,\theta\right)\right] \\ &\triangleq \mathcal{L}(q,\phi). \end{aligned} \tag{3.4}$$

The function $\mathcal{L}(q, \phi)$ is the evidence of lower bound (ELBO) of the log-likelihood of the data, which is the sum of q-expected complete log-likelihood and the entropy of the variational distribution q. The goal of variational inference is to maximize the ELBO. Equivalently, q is chosen by minimizing the KL divergence between the variational distribution and the true posterior distribution.

Since θ and r are conjugate pairs, the posterior distribution of θ_{ji} is a Beta distribution,

$$p(\theta_{ji}|r_{ji}, n_{ji}, \mu_j, M_j) \sim \text{Beta}(r_{ji} + M_j\mu_j, n_{ji} - r_{ji} + M_j(1 - \mu_j)). \tag{3.5}$$

Therefore, we propose a Beta distribution with parameter vector δ_{ji} as variational distribution,

$$\theta_{ji} \sim \text{Beta}(\delta_{ji1}, \delta_{ji2}).$$

The posterior distribution of μ_j is given by its Markov blanket,

$$p(\mu_j|\theta_{ji}, M_j, \mu_0, M_0) \propto p(\mu_j|\mu_0, M_0)p(\theta_{ji}|\mu_j, M_j). \tag{3.6}$$

This is not in the form of any known distribution. But, since the support of μ_j is $[0, 1]$, we propose a Beta distribution with parameter vector γ_j as variational distribution,

$$\mu_j \sim \text{Beta}(\gamma_{j1}, \gamma_{j2}).$$

Each component of ELBO is derived in Appendix A.1.

3.2.2.3 Variational EM algorithm

Variational EM algorithm maximizes the ELBO of the likelihood by alternating between maximization over q (E-step) and maximization over $\phi = \{\mu_0, M_0, M\}$ (M-step). We update the variational parameters and the model parameters iteratively by numerically optimizing each problem using Sequential Least SQuares Programming (SLSQP) [92].

(E-step): Updating the variational distributions The terms in the ELBO that depend on $q(\theta_{ji}|\delta_{ji1}, \delta_{ji2})$ are

$$\begin{aligned}
\mathcal{L}_{[q(\theta_{ji})]} = &\sum_{j=1}^{J}\sum_{i=1}^{N} \{r_{ji}E_q[\log\theta_{ji}] + (n_{ji} - r_{ji})E_q[\log(1-\theta_{ji})]\} \\
&+ \sum_{j=1}^{J}\sum_{i=1}^{N} \{M_j E_q[\mu_j] E_q[\log\theta_{ji}] - E_q[\log\theta_{ji}]\} \\
&+ \sum_{j=1}^{J}\sum_{i=1}^{N} \{(M_j - 1 - M_j E_q[\mu_j]) E_q[\log(1-\theta_{ji})]\} \\
&- \sum_{j=1}^{J}\sum_{i=1}^{N} E_q[\log q(\theta_{ji})]
\end{aligned} \tag{3.7}$$

We update the variational parameters by numerically optimizing

$$\hat{\delta}_{ji1}, \hat{\delta}_{ji2} = \arg\max_{\delta_{ji1},\delta_{ji2}} \mathcal{L}_{[q(\theta_{ji})]} \tag{3.8}$$

subject to the constraints that $\delta_{ji1} \geqslant 0$ and $\delta_{ji2} \geqslant 0$ and conditioned on fixed values for the other model and variational parameters using Sequential Least SQuares Programming (SLSQP).

We update the variational distribution $q(\mu_j)$ using the partial ELBO depending on γ_j from each position j (3.9).

$$\begin{aligned}
\mathcal{L}_{[q(\mu_j)]} = N \sum_{j=1}^{J} E_q \left[\log \left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma(M_j(1-\mu_j))} \right) \right] \\
+ \sum_{j=1}^{J} \sum_{i=1}^{N} \left\{ M_j E_q [\mu_j] E_q [\log \theta_{ji}] - E_q [\log \theta_{ji}] \right\} \\
+ \sum_{j=1}^{J} \sum_{i=1}^{N} \left\{ (M_j - 1 - M_j E_q [\mu_j]) E_q [\log (1-\theta_{ji})] \right\} \\
+ J \log \frac{\Gamma(M_0)}{\Gamma(\mu_0 M_0)\Gamma(M_0(1-\mu_0))} \\
+ \sum_{j=1}^{J} \left\{ (M_0\mu_0 - 1) E_q [\log \mu_j] + (M_0(1-\mu_0)-1) E_q [\log(1-\mu_j)] \right\} \\
- \sum_{j=1}^{J} E_q [\log q(\mu_j)]
\end{aligned} \tag{3.9}$$

Again, we update the variational parameters by numerically optimizing

$$\hat{\gamma}_{j1}, \hat{\gamma}_{j2} = \arg\max_{\gamma_{j1},\gamma_{j2}} \mathcal{L}_{[q(\mu_j)]} \tag{3.10}$$

subject to the constraints that $\gamma_{j1} \geqslant 0$ and $\gamma_{j2} \geqslant 0$ and conditioned on fixed values for the other model and variational parameters using SLSQP. The computational cost of optimizing (3.9) is high because of the quadrature of $E_q \left[\log \left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma(M_j(1-\mu_j))} \right) \right]$ in (A.8).

(M-step): Updating the model parameters We can write out the ELBO as a function of each model parameter μ_0, M_0, and M_j as follows.

The ELBO with respect to μ_0 is

$$\begin{aligned}\mathcal{L}_{[\mu_0]} = &-J * \log\Gamma(\mu_0 M_0) - J * \log\Gamma(M_0(1-\mu_0)) \\ &+ M_0\mu_0 \sum_{j=1}^{J} \left\{E_q\left[\log\mu_j\right] - E_q\left[\log(1-\mu_j)\right]\right\}.\end{aligned} \tag{3.11}$$

The ELBO with respect to M_0 is

$$\begin{aligned}\mathcal{L}_{[M_0]} = &J * \log\frac{\Gamma(M_0)}{\Gamma(\mu_0 M_0)\Gamma(M_0(1-\mu_0))} \\ &+ M_0 \sum_{j=1}^{J} \left\{\mu_0 E_q\left[\log\mu_j\right] + (1-\mu_0)E_q\left[\log(1-\mu_j)\right]\right\}.\end{aligned} \tag{3.12}$$

The ELBO with respect to M_j is

$$\begin{aligned}\mathcal{L}_{[M_j]} = &N * \sum_{j=1}^{J} E_q\left[\log\left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma(M_j(1-\mu_j))}\right)\right] \\ &+ M_j \sum_{j=1}^{J}\sum_{i=1}^{N} \left\{E_q\left[\mu_j\right] E_q\left[\log\theta_{ji}\right] + \left(1-E_q\left[\mu_j\right]\right) E_q\left[\log\left(1-\theta_{ji}\right)\right]\right\}.\end{aligned} \tag{3.13}$$

We also use SLSQP to optimize the ELBO function with respect to each parameter, μ_0, M_0, and M_j. It is computationally easy to optimize μ_0 (3.11) and M_0 (3.12). However, it is costly for optimizing M_j (3.13) because the quadrature is needed to calculate $E_q\left[\log\left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma(M_j(1-\mu_j))}\right)\right]$ using (A.8).

There is no analytical representation for

$E_q\left[\log\left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma(M_j(1-\mu_j))}\right)\right]$, which is required to update variational distribution for μ_j and model parameter M. So, we must resort to numerical integration,

$$\begin{aligned}&E_q\left[\log\left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma((1-\mu_j)M_j)}\right)\right] = \\ &\int_0^1 q(\mu_j;\gamma_{j1},\gamma_{j2}) \log\left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma((1-\mu_j)M_j)}\right) d\mu_j.\end{aligned} \tag{3.14}$$

Unfortunately, this numerical integration step is computationally expensive.

The variational EM algorithm is summarized using pseudocode in Algorithm 1.

Algorithm 1 Variational EM Inference

```
1:  Initialize $q(\theta, \mu)$ and $\hat{\phi}$
2:  repeat
3:      // E-step
4:      repeat
5:          for j = 1 to J do
6:              for i = 1 to N do
7:                  Optimize $\mathcal{L}(q, \hat{\phi})$ over $q(\theta_{ji}; \delta_{ji}) = \text{Beta}(\delta_{ji})$
8:              end for
9:          end for
10:         for j = 1 to J do
11:             Optimize $\mathcal{L}(q, \hat{\phi})$ over $q(\mu_j; \gamma_j) = \text{Beta}(\gamma_j)$
12:         end for
13:     until change in $\mathcal{L}(q, \hat{\phi})$ is small
14:     // M-step
15:     Set $\hat{\phi} \leftarrow \arg\max_{\phi} \mathcal{L}(\hat{q}, \phi)$
16: until change in $\mathcal{L}(\hat{q}, \phi)$ is small
```

3.2.3 Hypothesis testing

The posterior distribution over $\mu_j^{\triangle} \mid r^{case}, r^{control} \triangleq \mu_j | r^{case} - \mu_j | r^{control}$ is the distribution over the change in the non-reference read rate at position j between a case and control sample. Since the variational approximate posterior distributions in the difference are Beta distributions, the distribution of the difference is not analytically known. In order to compute the statistic of interest, we approximate $\mu_j | r^{case}$ and $\mu_j | r^{control}$ with univariate Gaussian distributions by matching the first two moments of the variational Beta distributions. Then, the difference is a Gaussian distribution. As we show in section 3.2.2 the Gaussian approximation is empirically reasonable.

Under the variational approximation,

$$E_q[\mu_j | r^{case}] = \frac{\gamma_{j1}^{case}}{\gamma_{j1}^{case} + \gamma_{j2}^{case}} \tag{3.15}$$

$$\text{Var}_q[\mu_j | r^{case}] = \frac{\gamma_{j1}^{case}\gamma_{j2}^{case}}{(\gamma_{j1}^{case} + \gamma_{j2}^{case} + 1)(\gamma_{j1}^{case} + \gamma_{j2}^{case})^2} \tag{3.16}$$

for $\mu_j | r^{case}$ and likewise for $\mu_j | r^{control}$. We approximate the posterior for the case sample as

$$\mu_j | r^{case} \sim \mathcal{N}(E_q[\mu_j | r^{case}], \text{Var}_q[\mu_j | r^{case}]) \tag{3.17}$$

and likewise for the control. Then,

$$\begin{aligned}&\mu_j^{\triangle} \mid r^{case}, r^{control} \sim \\ &\quad \mathcal{N}(E_q[\mu_j|r^{case}] - E_q[\mu_j|r^{control}], \text{Var}_q[\mu_j|r^{case}] + \text{Var}_q[\mu_j|r^{control}]).\end{aligned} \tag{3.18}$$

Now, we can approximate the posterior probability of interest,

$$\Pr(\mu_j^{\triangle} \geqslant \tau \mid r^{case}, r^{control}), \tag{3.19}$$

that is, the posterior probability that the difference in the non-reference read rate is greater than a fixed effect size τ (e.g. zero) for a one sided test. For a two sided test, we compute the approximate probability

$$\Pr(|\mu_j^{\triangle}| \geqslant \tau \mid r^{case}, r^{control}). \tag{3.20}$$

A position is called a *provisional variant* if $\Pr(|\mu_j^{\triangle}| \geqslant \tau \mid r^{case}, r^{control}) \geqslant 1 - \alpha/2$, where the probability is approximated as described.

It is possible that a position is called a variant due to a differential non-reference read count, but no particular alternative base is more frequently observed than the others. In this case, the likely cause is a sequencing error that indiscriminately incorporates a non-reference base at the position. To discriminate this non-biological cause from the interesting true variants we use a χ^2 goodness-of-fit test for non-uniform base distribution [79, 31]. For each provisional variant, if we reject the null hypothesis that the distribution is uniform, we promote the position to a *called variant*.

3.3 Experiments and results

3.3.1 Data sets

We validate the performance of our method on a synthetic DNA sequence data set and furthermore apply it on a longitudinal yeast data set.

3.3.1.1 Synthetic DNA sequence data

The data set we use to assess sensitivity and specificity is described and made available elsewhere [13]. Briefly, we performed an in-vitro mixture of two DNA sequences to test the sensitivity and specificity of our approach. Two 400bp DNA sequences were chemically synthesized. One sample has 14 variant loci and is taken as the case and the other without variants is taken as the control. Case and control DNA samples were mixed in-vitro to yield defined NRAF of 0.1%, 0.3%, 1.0%, 10.0%, and 100.0%. The synthetic DNA dataset was downsampled by $10\times$, $100\times$, $1,000\times$, and $10,000\times$ using `picard` (v 1.96). The final data set contains read pairs for six replicates for the control and cases at different NRAF levels.

3.3.1.2 Longitudinal directed evolution data

The longitudinal yeast data comes from three strains of haploid S288c which were grown for 448 generations under limited-glucose (0.08%). The wild-type ancestral strain GSY1136 was sequenced as a reference. Aliquots were taken about every 70 generations and sequenced. The detail of library sequencing is described in [89], [69], and [93]. The Illumina sequencing data is available on the NCBI Sequence Read Archive (SRA054922)[89]. For this study, we received the original BAM files from one of the authors. The aligned BAM files have 266 – $1,046\times$ coverage. We used `samtools` (v 1.1) with `-mpileup -C50` flags to convert BAM files to pileup files. Then, we generated depth chart files, which are tab-delimited text tables recording in each element of the table the count of a nucleotide at a genomic position. We ran our variational inference algorithm on the depth chart files to identify SNVs.

3.3.2 Performance on synthetic DNA data

3.3.2.1 Comparison of sensitivity and specificity

The performance of variational EM algorithm is shown in receiver-operating characteristic curves (ROCs) for a broad range of median read depths and NRAFs in Figure 3.2. The results in the ROC curves are generated by varying parameter α in the posterior distribution test. It shows that the performance improved with read depth and true mutant mixtures.

Furthermore, we evaluated the performance by using both the posterior distribution test with $\alpha = 0.05$ and the χ^2 test to detect variants, and compared the performance with the MCMC sampling algorithm in terms of sensitivity and specificity (Table 3.1). The variational EM algorithm shows higher sensitivity and specificity than the MCMC algorithm in the events when NRAF is 0.1%. The variational EM algorithm has a higher specificity compared with the MCMC algorithm for a median read depth of $41,472\times$ at 0.3% NRAF and $55,489\times$ at 1.0% NRAF, but the sensitivity is slightly lower due to false negatives.

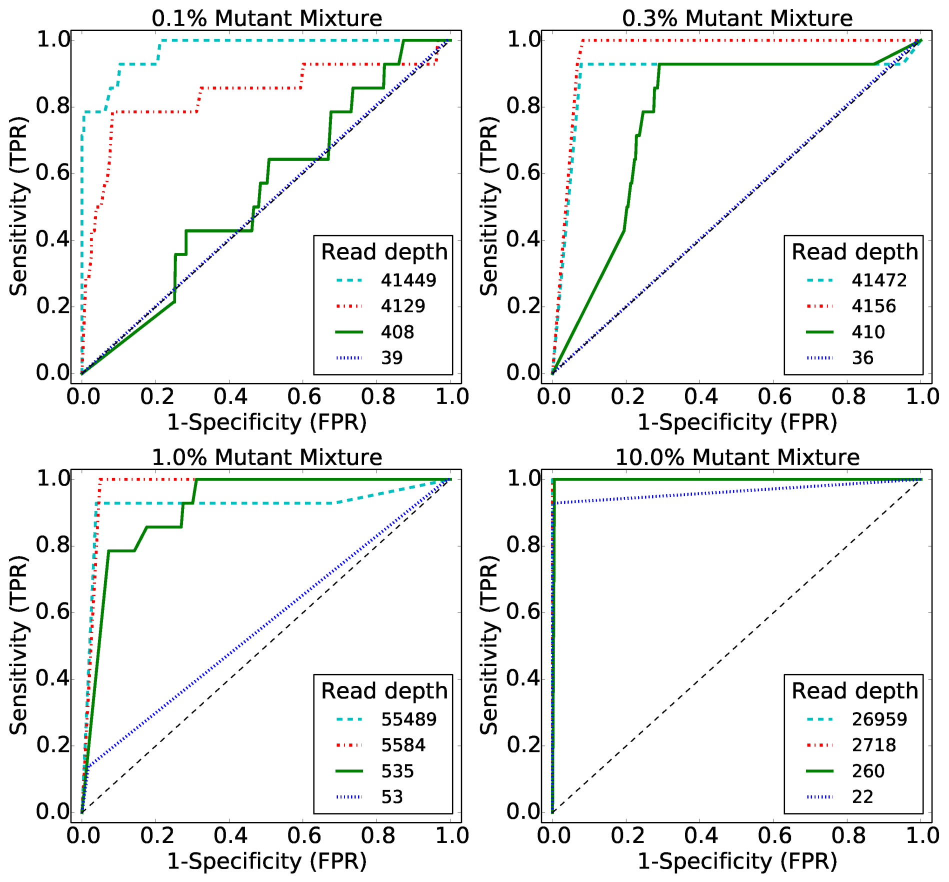


Figure 3.2: ROC curves with varying median read depths and NRAFs.

Table 3.1: Sensitivity/Specificity comparison of variational EM algorithm with MCMC algorithm.

True NRAF	Median Depth	Sensitivity		Specificity	
		MCMC	Variational	MCMC	Variational
0.1%	39	0.00	0.00	1.00	1.00
	408	0.00	0.07	1.00	1.00
	4129	0.14	0.29	1.00	1.00
	41449	0.86	1.00	0.97	1.00
0.3%	36	0.00	0.00	1.00	1.00
	410	0.00	0.00	1.00	1.00
	4156	1.00	1.00	0.99	0.98
	41472	1.00	0.93	0.85	0.91
1.0%	53	0.00	0.00	1.00	1.00
	535	0.21	0.29	1.00	1.00
	5584	1.00	1.00	0.98	0.98
	55489	1.00	0.93	0.87	0.95
10.0%	22	0.00	0.57	1.00	1.00
	260	1.00	1.00	1.00	1.00
	2718	1.00	1.00	1.00	1.00
	26959	1.00	1.00	1.00	1.00
100.0%	27	1.00	1.00	1.00	1.00
	298	1.00	1.00	1.00	1.00
	3089	1.00	1.00	1.00	1.00
	30590	1.00	1.00	1.00	1.00

3.3.2.2 Comparison of approximated posterior distribution

Figure 3.3 shows the approximate posterior distribution of the variational EM algorithm and samples of the MCMC algorithm. One variant position is taken as an example to show the comparison of the approximated posteriors. The variational EM and MCMC algorithms both identify all the variants when NRAF is 10.0% and 100.0%. The variational EM algorithm calls 90 false positive positions without a χ^2 test when NRAFs are 0.1% and 0.3% for low median read depth ($30\times$ and $400\times$). This is to be expected because it is highly unlikely to correctly identify a variant base with a population frequency of 1 in $1,000$ with less than a $1,000\times$ read depth.

A false positive, a non-mutated position that is called by the variational EM algorithm but not called by the MCMC algorithm, is shown in Figure 3.4. The variance of the MCMC posterior estimate is higher than that of the variational posterior estimate. We tested 10 random initial values variational inference algorithm and found the approximate posterior distributions from the variational EM algorithm are essentially equivalent for all random initializations. It is notable that the shape of the proposed Beta variational distribution is well approximated by a Gaussian.

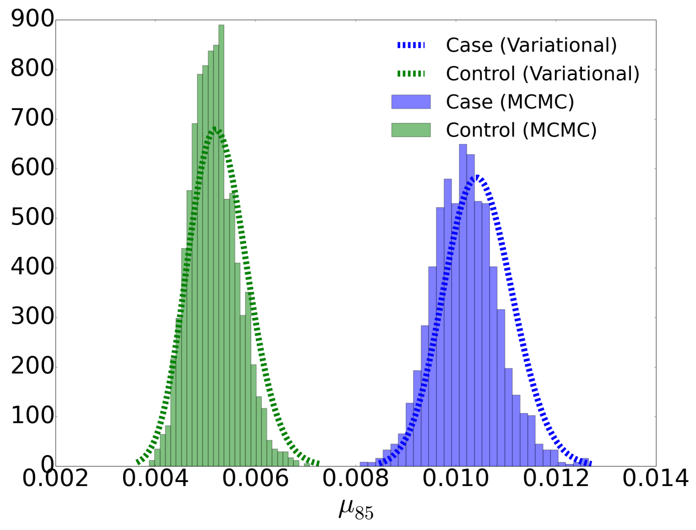


Figure 3.3: Approximated posterior distributions by the variational EM and MCMC algorithms on a true variant position (85) when the median read depth is $5,584\times$.

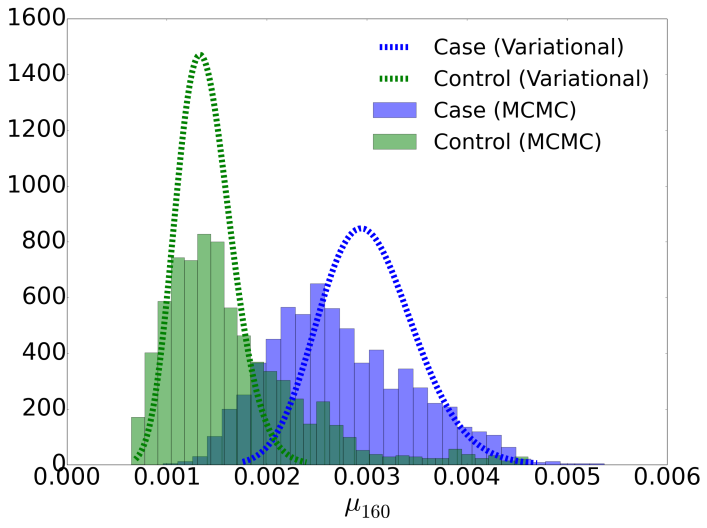


Figure 3.4: Approximated posterior distribution by the variational EM and MCMC algorithms on a non-variant position (160) that was not called by the MCMC algorithm (true negative), but was called by the variational EM algorithm (false positive) with a median read depth of $410\times$.

Table 3.2: Sensitivity/Specificity comparison with other variant detection methods.

NRAF	Median Depth	SAMtools	GATK	CRISP	VarScan2 mpileup	VarScan2 somatic	Strelka	SNVer	MuTect	RVD2 MCMC	RVD2 Variational
0.1%	39	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.00/0.99	0.00/1.00	0.00/1.00
	408	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.07/0.92	0.00/1.00	0.00/1.00	0.29/0.91	0.00/1.00	0.07/1.00
	4129	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.57/0.52	0.00/1.00	0.00/1.00	0.64/0.86	0.14/1.00	0.29/1.00
	41449	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.64/0.79	0.00/1.00	0.00/1.00	0.14/0.93	0.86/0.97	1.00/1.00
0.3%	36	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.43/1.00	0.00/1.00	0.00/1.00
	410	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.21/0.95	0.00/1.00	0.00/1.00	0.50/0.94	0.00/1.00	0.00/1.00
	4156	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.57/0.53	0.00/1.00	0.21/0.99	0.36/0.91	1.00/0.99	1.00/0.98
	41472	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.64/0.75	0.00/1.00	0.86/0.97	0.43/0.90	1.00/0.85	0.93/0.91
1.0%	53	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.00/0.99	0.00/1.00	0.00/1.00	0.29/0.98	0.00/1.00	0.00/1.00
	535	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.43/0.89	0.00/1.00	0.29/1.00	0.71/0.91	0.21/1.00	0.29/1.00
	5584	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.57/0.47	0.00/1.00	1.00/0.99	0.64/0.95	1.00/0.98	1.00/0.98
	55489	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00	0.64/0.69	0.00/1.00	1.00/0.97	0.86/0.90	1.00/0.87	0.93/0.95
10.0%	22	0.21/1.00	0.43/1.00	0.86/1.00	0.00/1.00	0.36/1.00	0.29/1.00	0.93/1.00	0.86/0.99	0.00/1.00	0.57/1.00
	260	0.00/1.00	0.57/1.00	1.00/1.00	0.00/1.00	0.86/1.00	1.00/1.00	1.00/1.00	1.00/0.99	1.00/1.00	1.00/1.00
	2718	0.00/1.00	0.79/1.00	0.07/1.00	0.00/1.00	0.57/0.78	1.00/1.00	1.00/0.99	1.00/0.98	1.00/1.00	1.00/1.00
	26959	0.00/1.00	0.57/1.00	0.00/1.00	0.00/1.00	0.64/0.53	1.00/0.99	1.00/0.98	1.00/0.98	1.00/1.00	1.00/1.00
100.0%	27	1.00/0.99	1.00/1.00	1.00/1.00	1.00/1.00	1.00/1.00	1.00/1.00	1.00/0.99	1.00/0.98	1.00/1.00	1.00/1.00
	298	1.00/0.99	1.00/1.00	1.00/1.00	1.00/1.00	1.00/0.99	1.00/0.99	1.00/0.98	1.00/0.98	1.00/1.00	1.00/1.00
	3089	0.86/1.00	1.00/1.00	1.00/0.99	1.00/1.00	1.00/0.65	1.00/0.99	1.00/0.98	1.00/0.98	1.00/1.00	1.00/1.00
	30590	0.71/1.00	1.00/1.00	1.00/1.00	1.00/1.00	1.00/0.39	1.00/1.00	1.00/0.98	1.00/0.99	1.00/1.00	1.00/1.00

3.3.2.3 Comparison to the state-of-the-art methods

We compared the performance of our variational EM algorithm with the state-of-the-art variant detection methods, SAMtools [53], GATK [17], CRISP[69], VarScan2 [91], Strelka [90], SNVer[70], MuTect [16], and RVD2 [31], using synthetic DNA data set (Table 3.2). Among all of the methods compared, our variational EM algorithm has a higher sensitivity and specificity for a broad range of read depths and NRAFs. Our variational EM algorithm shows higher specificity than all the other tested methods at a very low NRAF (0.1%) level. However, our algorithm has a slightly lower specificity than the MCMC algorithm when the median read depth is $4,156\times$ at 0.3% NRAF, and a slightly lower sensitivity than the MCMC algorithm when the median read depth is $41,472\times$ at 0.3% NRAF and a median read depth of $55,489\times$ at 1.0% NRAF. The performance of other methods is stated in detail in [31].

3.3.2.4 Comparison of timing

The computational time for approximating the variational posterior distribution is increased by expanding the length of region and the median read depth (Figure 3.5). Our variational EM algorithm is faster than the MCMC algorithm at the low median read depths of $27\times$ and $298\times$, and slower for the high median read depths of $3,089\times$ and $30,590\times$.

Table 3.3 shows the timing profile for each part of our variational EM algorithm when median read depth is $3,089\times$. Optimizing γ in the E-step and optimizing M_j in the M-step takes more than 95% of the time of one variational iteration in a test of a single processor, since the integration (3.14) is needed.

Table 3.3: Timing profile of variational EM algorithm when median depth is $3,089\times$.

		E-step			**M-step**				
Computation resource	**Region length**	Optimize γ	Optimize δ	Update ELBO	Optimize μ_0	Optimize M_0	Optimize M	Update ELBO	**Total time (s)**
single processor	100	617.7 (63%)	4.232	10.42	0.264	0.159	332.8 (34%)	10.29	976.0
	200	1124 (65%)	8.936	18.64	0.418	0.256	570.0 (33%)	18.37	1741
	300	1728 (65%)	13.27	27.81	0.445	0.400	851.5 (32%)	27.65	2649
	400	2433 (66%)	17.99	38.55	0.737	0.635	1176 (32%)	38.17	3705
60 processors	100	29.93 (41%)	0.2470	11.67	0.3070	0.1890	19.56 (26%)	11.98	73.89
	200	44.69 (40%)	0.4170	22.14	0.5230	0.3040	24.04 (21%)	22.24	114.3
	300	63.47 (40%)	0.7160	33.31	0.5620	0.5040	29.41 (18%)	33.24	161.2
	400	94.66 (43%)	0.7270	42.78	0.8200	0.7060	40.04 (18%)	44.28	219.7

Timing profile of 4 significant figures for one iteration of variational EM algorithm when median read depth is $3,089\times$. Single and multiple processors are both tested to estimate timing. Time for optimizing γ in the E-step and optimizing M in the M-step is highlighted in percentage.

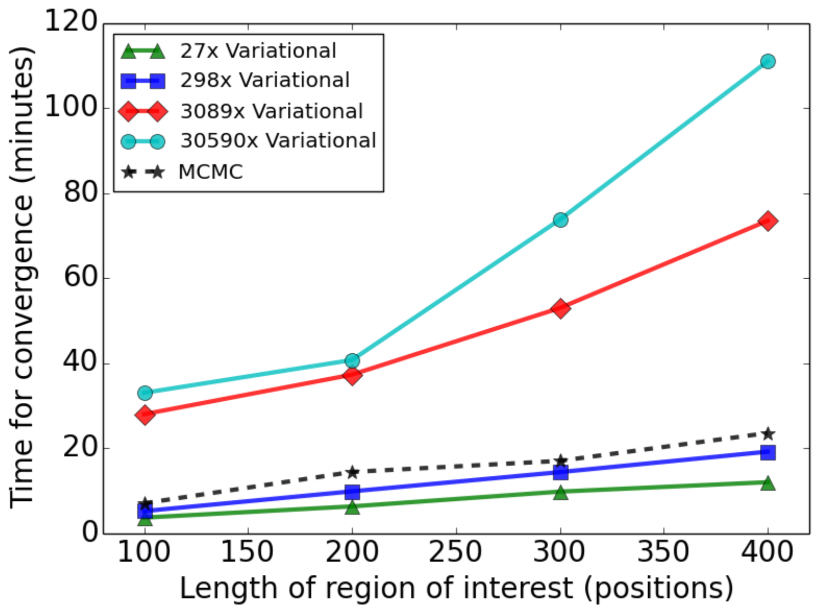


Figure 3.5: Timing comparison for the variational EM algorithm and MCMC sampling algorithm. Sixty processors are used to estimate the model on the synthetic data set.

3.3.3 Variant detection on the longitudinal directed evolution data

3.3.3.1 Detected variants

We applied our variational EM algorithm to the MTH1 gene at Chr04:1,014,401-1,015,702 (1,302bp), which is the most frequently observed mutated gene by [89]. Our algorithm detected the same variants that were found by [89] (shown as highlighted in Table 3.4 and Table 3.5). Additionally, we detected 81 novel variants in 8 timepoints that the original publication did not detect. In Additional file 2, G7 is the baseline NRAF as the control sample when comparing with G70, G133, G266, G322, G385, and G448 in the respective hypotheses testing. The corresponding NRAFs of called variants at different time points are given by the estimate of the latent variable, $\hat{\mu_j} = E_q[\mu_j|r]$.

All of these variants, except the variant at position Chr04:1,014,740, decrease in NRAF following a maximum. The allele at position Chr04:1,014,740 is a beneficial variant that arises in NRAF to 99.6% at generation 448 within a constant glucose-limited environment. Moreover, we identified the first emergence of this beneficial variant as early as 0.5% in generation 133. We detected 22 variants (NRAF $< 1.0\%$) early (at generation 70) in the evolutionary time course. Given that the median read depth is $1,649\times$, we have some confidence these are bona-fide variants.

Table 3.4: Identified variants on the longitudinal data set.

				Allele frequency of generation (%)							
Index	**Position**	**Ref**	**Alt**	**G7**	**G70**	**G133**	**G196**	**G266**	**G322**	**G385**	**G448**
1	1014407	T	C	0.127			0.620				
2	1014408	A	G	0.083	0.374						
3	1014422	C	G	0.160			1.036	7.431			
4	1014434	T	C	0.161						0.042	
5	1014436	A	G	0.161		0.127					
6	1014447	A	C	0.086			0.837	0.642			
7	1014455	C	T	0.044	0.345						
8	1014456	T	G	0.073	0.644	1.154					
9	1014457	G	T	0.031		0.777	2.675				
10	1014561	T	C	0.059	0.444						
11	1014580	A	G	0.161		0.096	0.168	0.481	0.088	0.038	0.081
12	1014582	T	C	0.161		0.149					
13	1014583	G	A	0.039	0.405	1.171	0.366				
14	1014595	A	G	0.065	0.096						
15	1014607	T	A	0.039	0.292	0.443					
16	1014615	G	T	0.079		0.865	1.327				
17	1014651	C	T	0.210			0.723				
18	1014689	T	A	0.161		0.101					
19	1014691	G	A	0.056		0.308					
20	1014698	C	T	0.160	0.13	0.629	0.565	0.767			
21	1014701	T	C	0.160		0.120					
22	1014707	A	C	0.160	0.825	1.461	7.434	6.920	0.176		
23	1014712	C	A	0.160	0.116	1.429	0.350	1.104			
24	1014740	G	C	0.158		0.522		23.849	98.286	99.715	99.603
25	1014741	C	T	0.028	0.997	1.622	0.780				
26	1014751	C	A	0.017		0.896					
27	1014765	A	C	0.159		0.431					
28	1014770	G	T	0.159	0.758	13.820	0.727	1.671			
29	1014791	C	T	0.014		0.918					
30	1014823	C	A	0.059		0.171					
31	1014856	T	C	0.160		0.119					
32	1014867	A	G	0.098	0.584	0.562	0.371	0.360			0.404
33	1014877	T	A	0.039			0.449				
34	1014920	G	C	0.049		0.324					
35	1014930	G	T	0.010		0.303					
36	1014958	C	A	0.160			1.884	1.036			
37	1014971	T	C	0.030	0.445						
38	1014968	A	T	0.037		0.316	3.446	1.462			
39	1014978	A	C	0.011		1.430					
40	1014997	G	A	0.008		0.368	2.287	2.564			
41	1015004	A	G	0.160	0.127						
42	1015036	G	A	0.026			0.640				
43	1015043	C	T	0.047		0.399					
44	1015051	G	A	0.161				0.084			
45	1015069	T	C	0.162						0.027	
46	1015074	A	T	0.002		0.400	0.510				
47	1015077	G	T	0.047		1.06					
48	1015078	A	G	0.186			0.689				
49	1015086	T	C	0.055			0.525	0.437			0.307
50	1015092	A	C	0.003			0.424				

Identified variants and corresponding NRAFs in gene MTH1 on Chromosome 4. A blank cell indicates that the position of that time point is not called significantly different than G7. The positions highlighted as blue were also identified by Kvitek, 2013. The other 81 positions are novel identified variants in 8 timepoints.

Table 3.5: Identified variants on the longitudinal data set (continued).

				Allele frequency of generation (%)							
Index	**Position**	**Ref**	**Alt**	**G7**	**G70**	**G133**	**G196**	**G266**	**G322**	**G385**	**G448**
51	1015190	A	C	0.160		0.459	0.545				
52	1015220	G	A	0.161		0.061					
53	1015222	T	C	0.161					0.040		0.067
54	1015228	T	A	0.054			0.293				
55	1015236	A	C	0.095	0.935	2.211	1.425	1.100			
56	1015247	A	G	0.060			0.142				
57	1015276	T	C	0.109			0.519				
58	1015280	T	A	0.519			1.108				
59	1015284	G	A	0.159				0.190			
60	1015317	G	C	0.160			0.168	0.306			
61	1015321	G	A	0.059		0.803					
62	1015322	A	T	0.013		0.519	0.488				
63	1015360	C	A	0.161			0.067				
64	1015368	C	T	0.064		0.147					
65	1015370	T	C	0.161		0.149					
66	1015371	G	C	0.062				0.147			
67	1015386	G	T	0.047	0.375	3.012	2.098	1.022			
68	1015411	A	G	0.042	0.214						
69	1015423	G	A	0.161			0.159				
70	1015424	T	C	0.050						0.194	
71	1015434	A	T	0.161		0.151					
72	1015477	C	A	0.043	0.440						
73	1015478	C	T	0.161			0.162				
74	1015512	G	T	0.161						0.006	
75	1015519	T	C	0.161		0.080					
76	1015521	G	A	0.018	0.371						
77	1015522	A	G	0.040			0.343				
78	1015539	G	C	0.040		0.618					
79	1015555	G	A	0.160		0.110					
80	1015556	A	G	0.161			0.117				
81	1015563	G	A	0.041			0.247				
82	1015623	T	G	0.034							0.194
83	1015627	T	C	0.161					0.060		
84	1015657	G	T	0.032		0.740	0.583	0.461			
85	1015666	G	A	0.065	0.091		0.105				
86	1015681	C	T	0.161					0.030		
87	1015691	T	C	0.037		0.358					
88	1015699	A	G	0.161			0.068				
89	1015700	C	A	0.015	0.949	1.965	1.044	0.379			
90	1015701	A	G	0.161		0.068					

Identified variants and corresponding NRAFs in gene MTH1 on Chromosome 4. A blank cell indicates that the position of that time point is not called significantly different than G7. The positions highlighted as blue were also identified by Kvitek, 2013. The other 81 positions are novel identified variants in 8 timepoints.

3.3.3.2 Sensitivity analysis

The global precision hyper-parameter M_0 could influence the estimate of μ_j due to its regularization effect. We show the influence of different $\hat{M}_0$ on variant position Chr04:1,014,740, $q(\mu_{1,014,740}|r)$ in Figure 3.6. We see that as we decrease the prior precision parameter $\hat{M}_0$, $\hat{\mu}_{1,014,740}$ increases as expected. But the effect of changing $\hat{M}_0$ over several orders of magnitude does not change $\hat{\mu}_j$ greatly. Here $\hat{M}_0 = 1.752$ in this dataset.

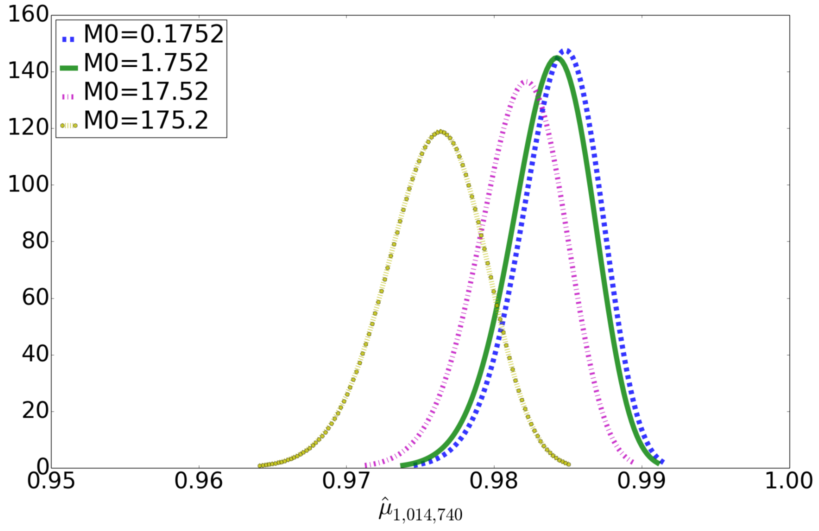


Figure 3.6: Influence of M_0 on the estimate of μ_j. Posterior distributions of the variant at position Chr04:1,014,740, $\hat{\mu}_{1,014,740}$, with different $\hat{M}_0$ are shown.

3.3.3.3 Concomitant variants detection

We identified a pair of variants, Chr04:1,014,740 in gene MTH1 and Chr12:200,286 in gene ADE16, that increase in NRAF together in time (Figure 3.7).

We hypotheses that the variants are concomitant in the same clone. In this pair of genes, gene MTH1 is a negative regulator of the glucose-sensing signal transduction pathway, and gene ADE16 is an enzyme of *de novo* purine biosynthesis. Glucose sensing induces gene expression changes to help yeast receive necessary nutrients, which could be a reason for this pair of genes to mutate together [94]. Further experimental validation of this hypothesis would be required to definitively show that the mutations are concomitant.

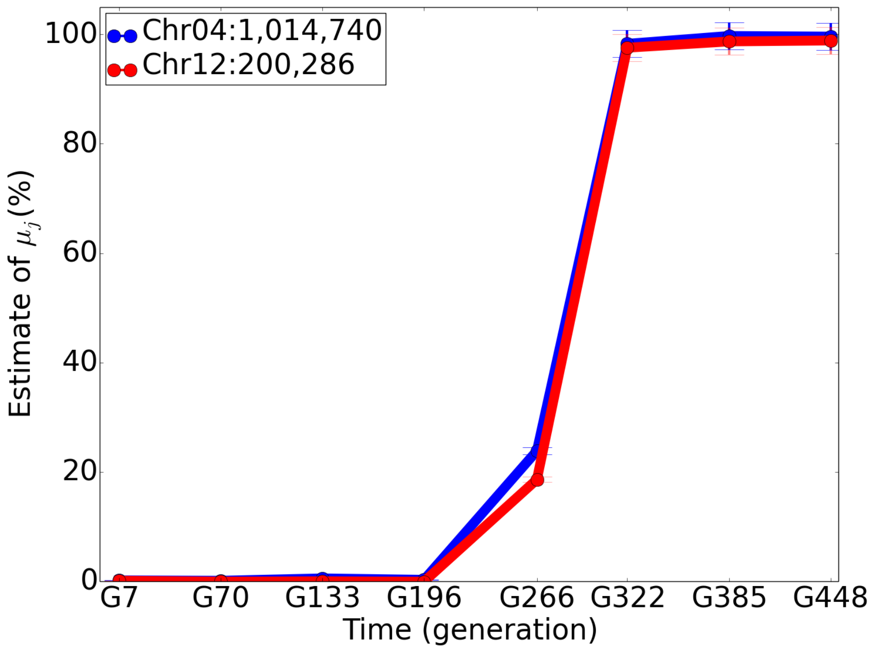


Figure 3.7: The NRAF trend of concomitant variants in gene MTH1 and ADE16. The 95% Bayesian credible intervals are shown.

3.4 Conclusions

We propose a variational EM algorithm to estimate the non-reference allele frequency in the RVD2 model to identify rare nucleotide variants in heterogeneous pools. Our results show that the variational EM algorithm (i) is able to identify rare variants at a 0.1% NRAF level with comparable sensitivity and specificity to a MCMC sampling algorithm; (ii) has a higher specificity in comparison with many state-of-the-art algorithms in a broad range of NRAFs; and (iii) detects SNVs early in the evolutionary time course, as well as tracks NRAF in a real longitudinal yeast data set.

We have chosen parametric forms for the variational distributions. This choice has left us with a complex integral in our variational optimization problem. In future work, we plan to explore other approximations of the variational distributions that render the integral easier to compute. One could use cubic splines to numerically approximate the function and then integrate that surrogate [95]. Another strategy is to consider a Laplace approximation for the variational distribution, as we and others have done previously [96, 97].

Improving the speed of the estimating algorithm enables us to interrogate whole-genome sequencing data. By doing this, we hope to reveal the dynamics of arising variants at the genome-wide scale to show the genetic basis of clonal interference. Our method could be extended to study drug resistance by characterizing tumor heterogeneity in targeted anti-cancer chemotherapy samples, or to find the causative variants that lead to drug resistance and understand the causes of resistance at the single nucleotide level.

Chapter 4

Sparse Mixed Membership Matrix Factorization using Global Optimization for Molecular Subtypes Classification

4.1 Background

Mixed membership matrix factorization has been used in document topic modeling [98], collaborative filtering [99], population genetics [100], and social network analysis [101]. The underlying assumption is that an observed feature for a given sample is a mixture of shared, underlying groups. These groups are called topics in document modeling, subpopulations in population genetics, and communities in social network analysis. In bioinformatics applications the groups are called subtypes and we adopt that terminology here. Mixed membership matrix factorization simultaneously identifies both the underlying subtypes and the distribution over those subtypes for each individual sample.

4.1.1 Mixed membership model

The mixed membership matrix factorization problem can equivalently be viewed as inference in a particular statistical model [102]. These models typically have a latent Dirichlet random variable that allows each sample to have its own distribution over subtypes and a latent variable for the feature weights that describe each subtype. The inferential goal is to estimate the joint posterior distribution over these latent variables and thus obtain the distribution over subtypes for each sample and the feature vector for each subtype. Non-negative matrix factorization techniques have been used in image analysis and collaborative filtering applications [103, 99]. Topic models for document clustering have also been cast as a matrix factorization problem [104].

The basic mixed membership model structure has been extended a variety of ways. A hierarchical Dirichlet prior allows one to obtain a posterior distribution over the number of subtypes [105]. A prior on the subtype variables allows one to impose specific sparsity constraints on the subtypes [106, 107, 108]. Correlated information may be incorporated to improve the coherence of the subtypes [109].

Sampling or variational inference methods are commonly used to estimate the posterior distribution of interest for mixed membership models, but these only provide local or approximate estimates. A mean-field variational algorithm [98] and a collapsed Gibbs sampling algorithm have been developed for Latent Dirichlet Allocation [110]. However, Gibbs sampling is approximate for finite chain lengths and variational inference is only guaranteed to converge to a local optimum.

4.1.2 Benders' decomposition and global optimization

In many applications it is important to obtain a globally optimal solution rather than a local or approximate solution. Biconvex optimization problems may have a number of local minima. However, it is possible that convex substructures of a biconvex optimization problem can be exploited to find solutions more efficiently than general nonlinear optimization methods might. In this way, biconvex optimization problems inhabit an interesting place between convex optimization problem where the local optimum is the global optimum and general nonlinear optimization problems that can be arbitrarily pathological. We expect that by exploiting the structure of a particular biconvex optimization problem, we might develop a deterministic global optimization algorithm that is scalable and efficient. Recently, there have been significant advances in deterministic optimization methods for general biconvex optimization problems [111, 112]. Here, we show that mixed membership matrix factorization can be cast as a biconvex optimization problem and an ϵ-global optimum can be obtained by these deterministic optimization methods [113].

Benders' decomposition exploits the idea that in a given optimization problem there are often "complicating variables" – variables that when held fixed yield a much simpler problem, such as a linear program, over the remaining variables [114]. Benders developed a cutting plane method for solving mixed integer optimization problems that can be so decomposed. Geoffrion later extended Benders' decomposition to situations where the primal problem (parametrized by fixed complicating variable values) no longer needs to be a linear program [115]. The Global Optimization (GOP) approach is an adaptation of the original Benders' decomposition that can handle a more general class of problems that includes mixed-integer biconvex optimization problems [116]. Here, we exploit the GOP approach for solving a particular mixed membership matrix factorization problem.

We outline the general sparse mixed membership matrix factorization problem in Section 4.2. In Section 4.3, we use GOP to obtain an ϵ-global optimum solution for the mixed membership matrix factorization problem. In Section 4.5, we show empirical accuracy and convergence time results on a synthetic data set. Finally, we discuss further computational and statistical issues in Section 4.6. The details of problem conditions, convergence properties, and a full outline of the algorithm steps for the branch-and-bound version of the algorithm are found elsewhere [116].

4.2 Problem formulation

The problem data is a matrix $y \in \mathbb{R}^{M \times N}$, where an element y_{ji} is an observation of feature j in sample i. We would like to represent each sample as a convex combination of K subtype vectors, $y_i = x\theta_i$, where $x \in \mathbb{R}^{M \times K}$ is a matrix of K subtype vectors and θ_i is the mixing proportion of each subtype. We would like x to be sparse for purposes of centroid signature interpretability. In the specific case of cancer subtyping, y_{ji} may be a normalized gene expression measurement for gene j in sample i. We write this matrix factorization problem as

$$\begin{aligned} \underset{\theta,x}{\text{minimize}}\ & \|y_i - x\theta_i\|_2^2 \\ \text{subject to}\ & \|x\|_1 \leqslant P \\ & \theta_i \in \Delta^{K-1}\ \forall i, \end{aligned} \tag{4.1}$$

where Δ^{K-1} is a K-dimensional simplex.

Optimization problem (4.1) can be recast with a biconvex objective and a convex domain as

$$
\begin{aligned}
\underset{\theta,x,z}{\text{minimize}}\ & \|y - x\theta\|_2^2 \\
\text{subject to}\ & \sum_{j=1}^{M}\sum_{k=1}^{K} z_{jk} \leqslant P \\
& -z_{jk} \leqslant x_{jk} \leqslant z_{jk}\ \forall(j,k) \\
& \theta_i \in \Delta^{K-1}\ \forall i,\, z_{jk} \geqslant 0\ \forall(j,k).
\end{aligned}
\tag{4.2}
$$

If either x or θ is fixed then (4.2) reduces to a convex optimization problem. Indeed, if x is fixed, the optimization problem is a form of constrained linear regression. If θ is fixed, we have a form of LASSO regression. We prove that (4.1) is a biconvex problem in Appendix B.2. Since the problem with x fixed and the problem with θ fixed are both computationally simple, we could take either x or θ to be the "complicating variables" in Benders' decomposition and we choose θ.

A common approach for solving an optimization problem with a nonconvex objective function is to alternate between fixing one variable and optimizing over the other. However, this approach only provides a local optimum [117]. A key to the GOP algorithm is the Benders'-based idea that feasibility and optimality information is shared between the primal problems in the form of constraints.

4.3 Algorithm

We use the global optimization approach to solve for ϵ-global optimum values of x and θ [118, 116]. First, we partition the optimization problem decision variables into "complicating" and "non-complicating" variables. Then, the GOP algorithm alternates between solving a *primal problem* over θ for fixed x, and solving a *relaxed dual problem* over x for fixed θ. The primal problem provides an upper bound on the original optimization problem because it contains more constraints than the original problem (x is fixed). The relaxed dual problem contains fewer constraints and forms a valid global lower bound. The algorithm iteratively tightens the upper and lower bounds on the global optimum by alternating between the primal and relaxed dual problem and tightening the relaxation in the relaxed dual problem at each iteration.

4.3.1 Initialization

We start by partitioning the problem into a relaxed dual problem and a primal problem (Figure 4.1). Recall our decision that the relaxed dual problem optimizes over x for fixed values of the complicating variables θ and the primal problem optimizes over θ. We also initialize an iteration counter $T = 1$.

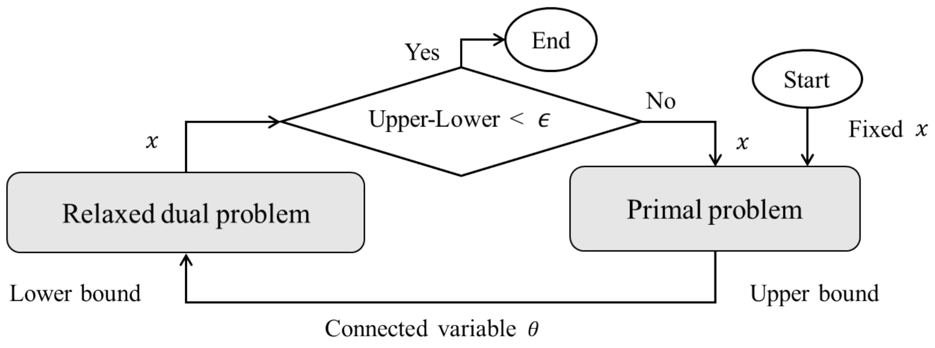


Figure 4.1: The GOP framework. The formulations for the primal problem and the relaxed dual problem are described in detail in Section 4.3.2 and Section4.3.3, respectively.

At each iteration, the relaxed dual problem is solved by forming a partition of the domain of x and solving a relaxed dual primal problem for each subset. A branch-and-bound tree data structure is used to store the solution of each of these relaxed dual primal problems and we initialize the root node $\texttt{n}(0)$ where $T = 0$. The parents of $\texttt{n}(T)$ is denoted $\texttt{par}(\texttt{n}(T))$, the set of ancestors of $\texttt{n}(T)$ is denoted $\texttt{anc}(\texttt{n}(T))$, and the set of children of $\texttt{n}(T)$ is denoted $\texttt{ch}(\texttt{n}(T))$.

Finally, we initialize x at a random feasible point, $x^{\mathrm{n}(0)}$, and store it in $\texttt{n}(0)$ since we will be starting the GOP iterations by solving the primal problem over θ for a fixed x.

4.3.2 Solve primal problem and update upper bound

The primal problem (4.2) is constrained to fixed value of x at $\texttt{n}(T)$, $x^{(\mathrm{n}(T))}$, so the primal problem is

Primal problem

(x fixed)

$$\begin{aligned} &\underset{\theta}{\text{minimize}}\ \|y - x\theta\|_2^2 \\ &\text{subject to}\ \theta_i^T 1_K = 1 \\ &\qquad\qquad \theta_{ki} \geqslant 0. \end{aligned}$$

Since the primal problem is more constrained than (4.2), the solution, $S^{(\mathrm{n}(T))}$, is a global upper bound. We store the value of the upper bound, PUBD $\leftarrow \min\{\text{PUBD}, S^{(\mathrm{n}(T))}\}$, where PUBD stores the tightest upper bound.

4.3.3 Solve the relaxed dual problem and update lower bound

The relaxed dual problem is a relaxed version of (4.2) in that it contains fewer constraints than the original problem. Initially, at the root node $\mathrm{n}(0)$ the domain of the relaxed dual problem is the entire domain of x, $\mathcal{X}$. Each node stores a set of linear constraints (cuts) such that when all of the constraints are satisfied, they define a region in $\mathcal{X}$. Sibling nodes form a partition of parent's region and a node deeper in the tree defines a smaller region than shallower nodes when incorporating the constraints of the node and all of its ancestors. These constraints are called *qualifying constraints*. Since the objective function is convex in θ for a fixed value of x, a Taylor series approximation for linearization of the Lagrangian with respect to θ provides a valid lower bound on the objective function. Finally, since the objective function is convex in θ, the Taylor approximation is linear and the optimal objective is at a bound of θ. The GOP algorithm as outlined in [111] makes these ideas rigorous.

The relaxed dual problem for the mixed membership matrix factorization problem (4.2) for a node $\mathrm{n}(T)$ is

Relaxed Dual Problem

(θ fixed)

$$\begin{aligned}
&\underset{Q,x,z}{\text{minimize}}\ Q \\
&\text{subject to}\ \sum_{j=1}^{M}\sum_{k=1}^{K} z_{jk} \leqslant P \\
&\qquad -z_{jk} \leqslant x_{jk} \leqslant z_{jk},\ z_{jk} \geqslant 0 \\
&\qquad \text{for } t \in \{\texttt{anc}(\texttt{n}(T)), \texttt{n}(T)\}: \\
&\qquad \begin{cases} Q \geqslant L(x, \theta^B(t), y, \lambda^t, \mu^t)\big|^{\text{lin}}_{x^t,\theta^t} \\ g^t_{ki}\big|^{\text{lin}}_{x^t}(x) \leqslant 0 \text{ if } \theta^B(t)_{ki} = 1 \\ g^t_{ki}\big|^{\text{lin}}_{x^t}(x) \geqslant 0 \text{ if } \theta^B(t)_{ki} = 0, \end{cases}
\end{aligned}$$

where $L(x, \theta^B(t), y, \lambda^t, \mu^t)\big|^{\text{lin}}_{x^t,\theta^t}$ is the linearized Lagrangian of (4.2), $g^t_{ki}\big|^{\text{lin}}_{x^t}(x)$ is the ki-th qualifying constraint, and $\theta^B(t)$ is the value of θ at the bound such that the linearized Lagrangian is a valid lower bound in the region defined by the qualifying constraints at node t. The corresponding Lagrangian function $L(x, \theta^B(t), y, \lambda^t, \mu^t)\big|^{\text{lin}}_{x^t,\theta^t}$ provides a lower bound on Q in the relaxed dual problem. We have taken a second Taylor approximation with respect to x to ensure the qualifying constraints are linear in x and thus valid cuts as recommended in [111].

Construct a child node in the branch-and-bound tree. A unique region in $\mathcal{X}$ for the leaf node $\texttt{ch}(\texttt{n}(T))$ is defined by the t-th row of θ^B derived from the primal problem at node $\texttt{n}(T)$. We can express this region as the qualifying constraint set,

$$\begin{aligned}
g^{\texttt{ch}(\texttt{n}(T))}_{ki}\big|^{\text{lin}}_{x^{\texttt{n}(T)}}(x) \leqslant 0 \text{ if } \theta^B(t)_{ki} = 1 \\
g^{\texttt{ch}(\texttt{n}(T))}_{ki}\big|^{\text{lin}}_{x^{\texttt{n}(T)}}(x) \geqslant 0 \text{ if } \theta^B(t)_{ki} = 0
\end{aligned}$$

First, we create the t^{th} child node of $\texttt{n}(T)$ and populate it with this constraint set and $\theta^B(t)$ which will be used in the construction of the Lagrangian function lower bound in the relaxed dual problem.

Second, we construct and solve the relaxed dual problem at $\texttt{ch}(\texttt{n}(T))$. First, we add the qualifying constraint sets contained in each node along the path in the branch-and-bound tree from $\texttt{ch}(\texttt{n}(T))$ to the root, inclusively. For example, the qualifying constraint set for a node $\texttt{n}'$ along the path is

$$g_{ki}^{\texttt{n}'}\Big|_{x^{\texttt{n}'}}^{\text{lin}}(x) \leqslant 0 \text{ if } \theta^B(\texttt{n}')_{ki} = 1$$
$$g_{ki}^{\texttt{n}'}\Big|_{x^{\texttt{n}'}}^{\text{lin}}(x) \geqslant 0 \text{ if } \theta^B(\texttt{n}')_{ki} = 0,$$

where $g_{ki}^{\texttt{n}'}$ is the node's ki^{th} qualifying constraint, $x^{\texttt{n}'}$ is the node's relaxed dual problem optimizer, and $\theta^B(\texttt{n}')$ is a 0-1 vector defining the unique region for node $\texttt{n}'$ since $\theta_{ki} \in [0, 1]$.

Third, we add the Lagrangian function lower bound constraints constructed from each node along the path in the branch-and-bound tree from $\texttt{ch}(\texttt{n}(T))$ to the root, inclusively. For example the linearized Lagrangian function for node $\texttt{n}'$,

$$L(x, \theta^B(\texttt{n}'), y, \lambda^{(\texttt{n}')}, \mu^{(\texttt{n}')})\Big|_{x^{(\texttt{n}')}, \theta^{(\texttt{n}')}}^{\text{lin}}.$$

Populate the child node with the linearized Lagrangian function and qualifying constraints. The Lagrangian function for the primal problem is

$$\begin{aligned} L(x, \theta, \lambda, \mu) &= \sum_{i=1}^{N} L(x, \theta_i, \lambda_i, \mu_i) \\ &= \sum_{i=1}^{N} (y_i - x\theta_i)^\top (y_i - x\theta_i) \\ &- \lambda_i(\theta_i^\top 1_K - 1) - \mu_i^\top \theta_i \\ &= \sum_{i=1}^{N} y_i^\top y_i - 2y_i^\top x\theta_i + \theta_i^\top x^\top x\theta_i \\ &- \lambda_i(\theta_i^\top 1_K - 1) - \mu_i^\top \theta_i \end{aligned} \tag{4.3}$$

with Lagrange multipliers $\mu \in \mathbb{R}_+^{K\times N}$ and $\lambda \in \mathbb{R}^N$.

The relaxed dual problem makes use of the Lagrangian function linearized about $\theta^{(t)}$ which we obtain through a Taylor series approximation,

$$\begin{aligned} L(x, \theta_i, \lambda_i, \mu_i)\Big|_{\theta^{(t)}}^{\text{lin}} \triangleq\ & L(x, \theta_i^{(t)}, \lambda_i^{(t)}, \mu_i^{(t)}) \\ & + \sum_{k=1}^{K} g_{ki}^{(t)}(x) \cdot \left(\theta_{ki} - \theta_{ki}^{(t)}\right), \end{aligned} \tag{4.4}$$

where the qualifying constraint function is

$$\begin{aligned} g_i^{(t)}(x) \triangleq & \nabla_{\theta_i} L\left(\theta_i, x, \lambda_i^{(t)}, \mu_i^{(t)}\right) \big|_{\theta_i^{(t)}} \\ &= -2y_i^\top x + 2\theta_i^{(t)\top} x^\top x \\ &- 1_K^\top \lambda_i^{(k)} - \mu_i^{(k)\top}. \end{aligned} \tag{4.5}$$

The qualifying constraint $g_i^{(t)}(x)$ is quadratic in x. However, we require it to be linear in x to yield a convex domain if $g_i^{(t)}(x) \geqslant 0$ or $g_i^{(t)}(x) \leqslant 0$. So, we linearize the Lagrangian first with respect to x about $x^{(t)}$ then about θ_i at $\theta_i^{(t)}$. While the linearized Lagrangian is not a lower bound everywhere in x, it is a valid lower bound in the region bound by the qualifying constraints with θ_i set at the corresponding bounds in the Lagrangian function.

The Lagrangian function linearized about $x^{(t)}$ is

$$\begin{aligned} L(y_i, \theta_i, x, \lambda_i, \mu_i)\Big|_{x^{(t)}}^{\text{lin}} \triangleq & y_i^T y_i - \theta_i^\top x^{(t)\top} x^{(t)} \theta_i \\ & - 2y_i^\top x\theta_i + 2\theta_i^\top x^{(t)\top} x\theta_i \\ & - \lambda_i(\theta_i^\top 1_K - 1) - \mu_i^\top \theta_i. \end{aligned} \tag{4.6}$$

Subsequently, the Lagrangian function linearized about $(x^{(t)}, \theta_i^{(t)})$ is

$$\begin{aligned} L(y_i, \theta_i, x, \lambda_i, \mu_i)\Big|_{x^{(t)}, \theta_i^{(t)}}^{\text{lin}} \triangleq & \; y_i^\top y_i + \theta_i^{(t)\top} x^{(t)\top} x^{(t)} \theta_i^{(t)} \\ & - 2\theta_i^{(t)\top} x^{(t)\top} x^{(t)} \theta_i \\ & - \lambda_i(1_K^\top \theta_i - 1) - \mu_i^\top \theta_i \\ & - 2\theta_i^{(t)\top} x^\top x^{(t)} \theta_i^{(t)\top} - 2y_i^\top x\theta_i \\ & + 2\theta_i^{(t)\top}(x^{(t)\top} x + x^\top x^{(t)})\theta_i \end{aligned} , \tag{4.7}$$

and the gradient used in the qualifying constraint is

$$\begin{aligned} g_i^{(t)}\big|_{x^{(t)}}^{\text{lin}}(x) \triangleq & \nabla_{\theta_i} \left[L(y_i, \theta_i, x, \lambda_i, \mu_i)\Big|_{x_0}^{\text{lin}} \right] \Big|_{\theta_i^{(t)}} \\ &= -2x^{(t)\top} x^{(t)} \theta_i^{(t)} - 2x^\top y_i \\ &+ 2(x^{(t)\top} x + x^\top x^{(t)})\theta_i^{(t)} - \lambda_i 1_K - \mu_i. \end{aligned} \tag{4.8}$$

Solve the relaxed dual problem at the child node. Once the valid qualifying constraints from the previous $t = 1, \dots, T-1$ iterations have been identified and incorporated, the constraint for the current T^{th} iteration is

$$\begin{aligned} Q \geqslant \;\; & L(x, \theta^{B_T}, y, \lambda^{(t)}, \mu^{(t)})\Big|_{x^{(t)},\theta^{(t)}}^{\text{lin}} \\ & g_{ki}^{(T)}\Big|_{x^{(t)}}^{\text{lin}}(x) \leqslant 0 \text{ if } \theta_{ki}^{B_T} = 1 \\ & g_{ki}^{(T)}\Big|_{x^{(t)}}^{\text{lin}}(x) \geqslant 0 \text{ if } \theta_{ki}^{B_T} = 0. \end{aligned}$$

The resulting relaxed dual problem is a linear program and can be solved efficiently using the off-the-shelf LP solver Gurobi [119]. We store the optimal objective function value and the optimizing decision variables in the node.

Update the lower bound. The global lower bound is provided by the lowest lower bound across all the leaf nodes in the branch-and-bound tree. We store this global lower bound in a variable, RLBD. Operationally, we maintain a dictionary where the value of a record is a pointer to a branch-and-bound tree node and the key is the optimal value of the relaxed dual problem at that leaf node. Using this dictionary, we select the smallest key and bound to the node of the tree indicated by the value. This element is eliminated from the dictionary since at the end of the next iteration, it will be an interior node and not available for consideration. We increment the iteration count $T \leftarrow T + 1$ and we update the global lower bound RLBD with the optimal value of the relaxed dual problem at the new node.

Check convergence. Since we always select the lowest lower bound provided by the relaxed dual problem, the lower bound is non-decreasing. If our convergence criteria PUBD $-$ RLBD $\leqslant \epsilon$ has been met, then we exit the algorithm and report the optimal θ from the node's primal problem and the optimal x from the node's relaxed dual problem. Finite ϵ-convergence and ϵ-global optimality proofs can be found in [116].

4.4 Computational improvements

In the relaxed dual problem branch-and-bound tree, a leaf node below the current node $\mathrm{n}(T)$ is constructed for each unique region defined by the hyperplane arrangement. In the GOP framework, there are KN hyperplanes, one of each so-called "connected variable" and all of the KN elements of θ are connected variables. So, an upper bound on the number of regions defined by KN cuts is 2^{KN} because each region may be found by selecting a side of each cut. Thus we have the computationally complex situation of needing to solve a relaxed dual problem for each of the 2^{KN} possible regions.

Let an arrangement $\mathcal{A}$ denote a set of hyperplanes and $r(\mathcal{A})$ denote the set of unique regions defined by $\mathcal{A}$. In our particular situation, all of the hyperplanes pass through the unique point $x^{(\mathrm{n}(T))}$, so all of the regions are unbounded except by the constraints provided in $\mathcal{X}$. A recursive algorithm for counting the number of regions $|r(\mathcal{A})|$ known as Zaslavsky' Theorem, is outlined in [120]. Indeed, $|r(\mathcal{A})|$ is often much less that $2^{|\mathcal{A}|}$. However, due to its recursive nature, computing the number of hyperplanes using Zaslavsky's theorem is computationally slow.

4.4.1 Cell enumeration algorithm

We have developed an A-star search algorithm for cell enumeration to simultaneously identify and count the set of unique regions defined by arrangement $\mathcal{A}$ with sign vectors. First, we preprocess the arrangement $\mathcal{A}$ to eliminate trivial and redundant hyperplanes. We eliminate a hyperplane from $\mathcal{A}$ if the coefficients are all zero and eliminate duplicate hyperplanes in $\mathcal{A}$ (see Appendix B.3). We are left with a reduced arrangement, $\mathcal{A}'$.

Here we define two concepts, *strict hyperplane* and *adjacent region*. A strict hyperplane is defined as non-redundant bounding hyperplane in a single region. If two regions exist that have sign vectors differing in only one hyperplane, then this hyperplane is a strict hyperplane. We define an adjacent region of region r as a neighbor region of r if they are separated by exactly one strict hyperplane. The general idea of the A-star algorithm uses ideas from partial order sets. We first initialize a root region using an interior point method and then determine all of its adjacent regions by identifying the set of strict hyperplanes. This process guarantees that we can enumerate all unique regions.

We define $\theta^B \in \{0,1\}^{|r(\mathcal{A}')|\times KN}$. The rows are regions and there are KN columns. Each element of this matrix is either 0 or 1. The b^{th} region in $r(\mathcal{A}')$ is uniquely identified by the zero-one vector in the b^{th} row of θ^B. If the b^{th} element of the ki^{th} row of θ^B is $+1$, then $g_{ki} \leqslant 0$. Similarly, if the b^{th} element of the ki^{th} row of θ^B is 0, then $g_{ki} \geqslant 0$. The A-star search algorithm completes the θ^B matrix for the current node $\mathrm{n}(T)$ and a leaf node is generated for each row of θ^B. Thus each unique region defined by the qualifying constraint cuts provided by the Lagrange dual of the primal problem at the current node. The details of the A-star search algorithm are covered in Section B.3.

4.5 Experiments and results

In this section, we present our experiments on synthetic data sets and show accuracy and convergence speed. Computational complexity is evaluated by both the theoretical and empirical time complexity.

4.5.1 Illustrative example

We use a simple data set to show the operation of the algorithm in detail and facilitate visualization of the cut sets. The data set, y, and true decision variable values, (x^*, θ^*), are

$$x^* = \begin{bmatrix} 0, & -1 \end{bmatrix}, \theta^* = \begin{bmatrix} 1, & 0, & 0.5 \\ 0, & 1, & 0.5 \end{bmatrix},$$
$$y = \begin{bmatrix} 0, & -1, & -0.5 \end{bmatrix}.$$

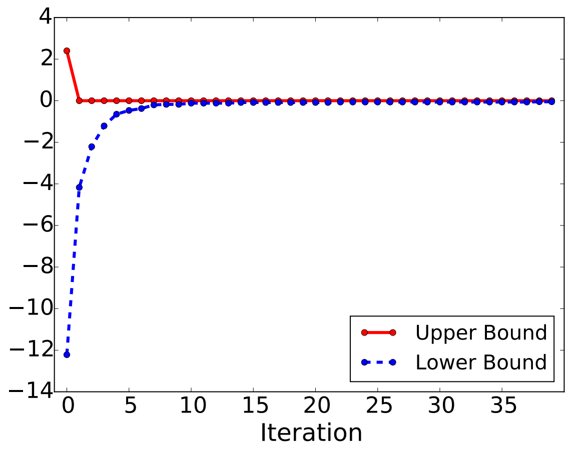
We ran the GOP algorithm with sparsity constraint variable $P = 1$ and convergence tolerance $\epsilon = 0.01$. There are $KN = 6$ connected variables, so we solve at most $2^{KN} = 64$ relaxed dual problems at each iteration. These relaxed dual problems are independent and can be distributed to different computational threads or cores. The primal problem is a single optimization problem and will not be distributed. The optimal decision variables after 72 iterations are

$$\hat{x} = x^{(72)} = \begin{bmatrix} 0.080, & -0.920 \end{bmatrix},$$
$$\hat{\theta} = \theta^{(72)} = \begin{bmatrix} 1.00, & 0.080, & 0.580 \\ 0.00, & 0.920, & 0.420 \end{bmatrix},$$

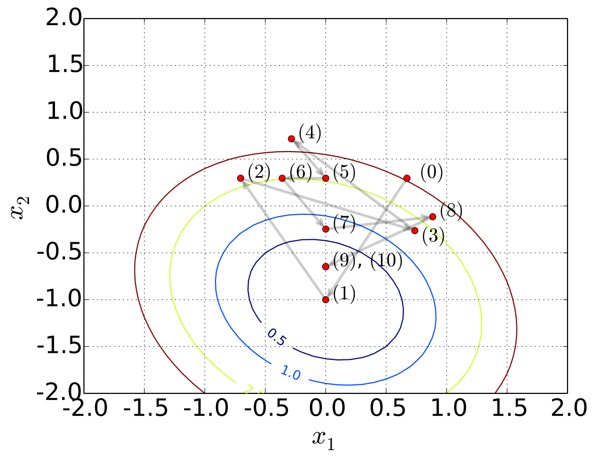
and the Lagrange multipliers are $\hat{\lambda} = [-0.147, 0, 0]$ and $\hat{\mu} = [0, 0, 0; 0.160, 0, 0]$.

Figure 4.2 (a) shows the convergence of the upper and lower bounds by iteration. The upper bound converges quickly and the majority of the time in the algorithm is spent proving optimality. With each iteration regions of the solution space are tested until the lower bound is tightened sufficiently to meet the stopping criterion. Figure 4.2 (b) shows the first ten x values considered by the algorithm with isoclines of the objective function with θ^* fixed. It is evident that the algorithm is not performing hill-climbing or any other gradient ascent algorithm during its search for the global optimum. Instead, the algorithm explores a region bound by the qualifying constraints to construct a lower bound on the objective function. We run it using 20 random initial values and the optimal objective functions for all random initializations are all 0, which shows that the GOP algorithm found the globally optimal solutions of this small instance. Furthermore, the algorithm does not search nested regions, but considers previously explored cut sets (Figure 4.2 (b)).

Figure 4.3 shows the branch-and-bound tree and corresponding x-space region with the sequence of cut sets for the first three iterations of the algorithm. One cut in Figure 4.3 (b, d, f) is obtained for each of the KN qualifying constraints. We initialize the algorithm at $x^{(0)}$.



(a) Upper and lower bounds.



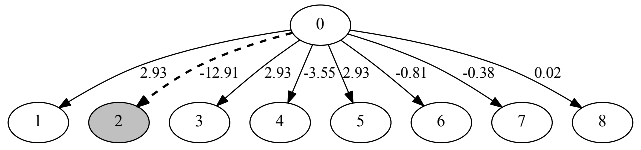
(b) Optimal relaxed dual problem decision variables.

Figure 4.2: GOP inference optimal values and optimizing x variables.

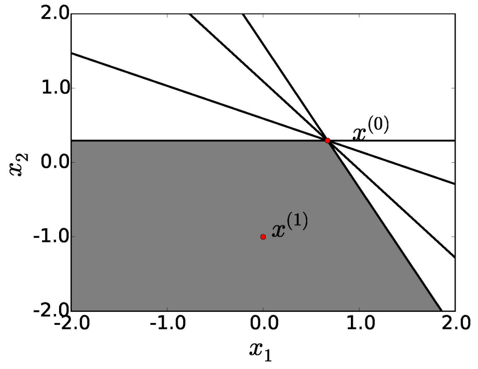
We also compare the performance of the GOP algorithm with the variational and MCMC algorithms in Figure 4.4. Here, we evaluate the summation of the estimated values for the objective function and the sparsity term per iteration for each algorithm. For accuracy, the GOP algorithm achieves the global optimal value of this summation at the last iteration. For efficiency, the variational algorithm is less efficient than the other two algorithms. In the application of this synthetic data set, the GOP algorithm is more accurate than the variational and MCMC algorithms, but it is relatively inefficient.

4.5.2 Accuracy and convergence speed

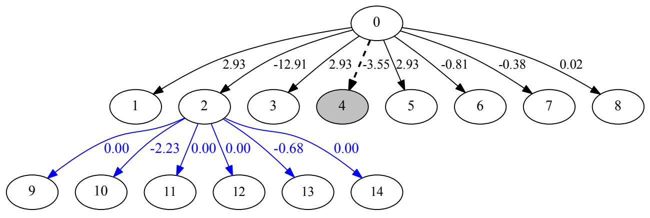
We ran our GOP algorithm using 64 processors on a synthetic data set which is randomly generated on the scale of one feature ($M = 1$), two subtypes ($K = 2$) and ten samples



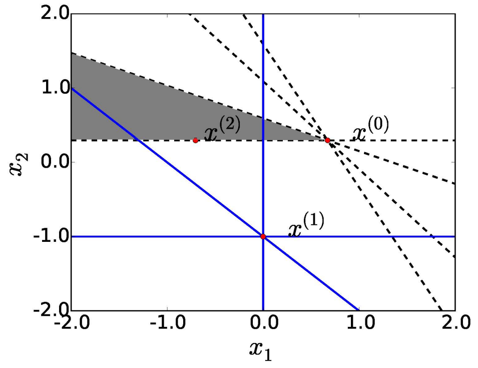
(a) Branch-and-bound tree at iteration 1



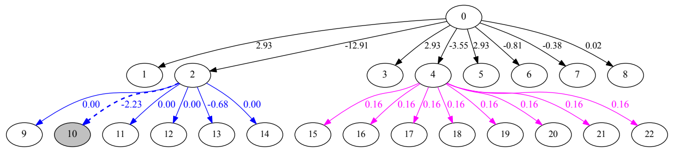
(b) x-space region at iteration 1



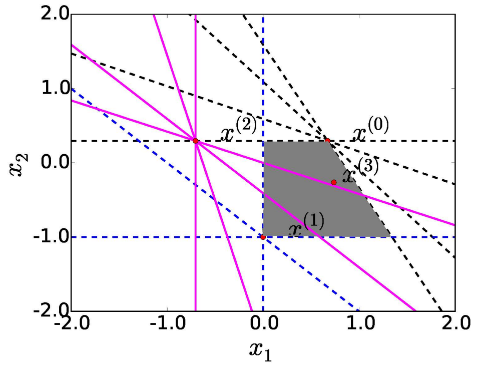
(c) Branch-and-bound tree at iteration 2



(d) x-space region at iteration 2



(e) Branch-and-bound tree at iteration 3



(f) x-space region at iteration 3

Figure 4.3: GOP branch-and-bound tree and corresponding x-space region. The gray node indicates the current node. The numbers on the edges indicate the optimal value of the relaxed dual problem.

$(N = 10)$. Figure 4.5 (a) shows that our GOP algorithm converges very quickly to 0.17 duality gap (PUBD $-$ RLBD) in the first 89 iterations in 120 seconds. The optimal x (x_1, x_2) and θ (θ_1, θ_2) of each iteration are shown with a range of colors to represent corresponding RLBD in Figure 4.5 (b, c). The dark blue represents low RLBD and the dark red represents high RLBD. The RLBD of the initial x, $x^{(0)}$, is -59.87; The RLBD of iteration 89, $x^{(89)}$, is

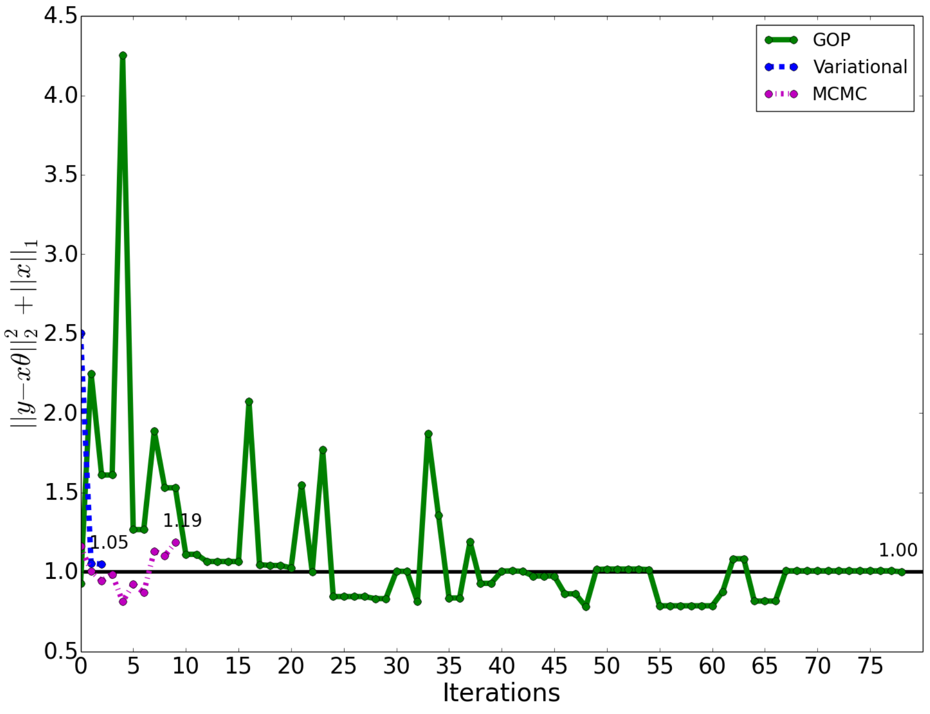


Figure 4.4: Method comparison of the GOP, variational, and MCMC algorithms.

-0.17. It demonstrates that the GOP algorithm can change modes very easily without getting stuck in local optima.

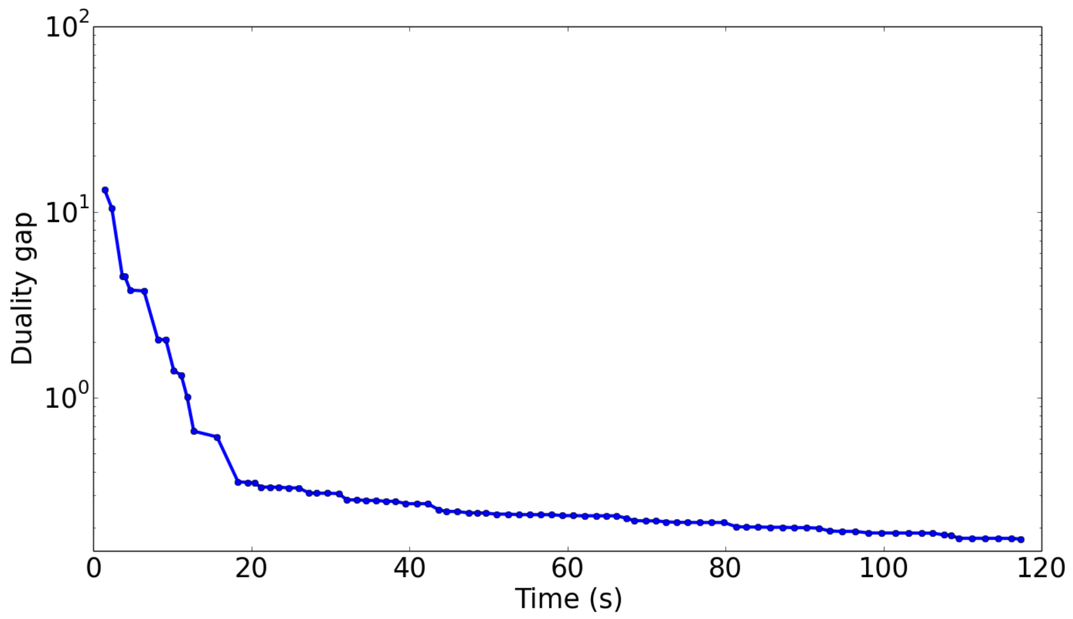
4.5.3 Computational complexity

We evaluate the GOP algorithm by theoretical analysis and empirical measurements of the time complexity on simulated data sets. The problem has four main components: primal problem, preprocessing, unique region identification, and relaxed dual problems.

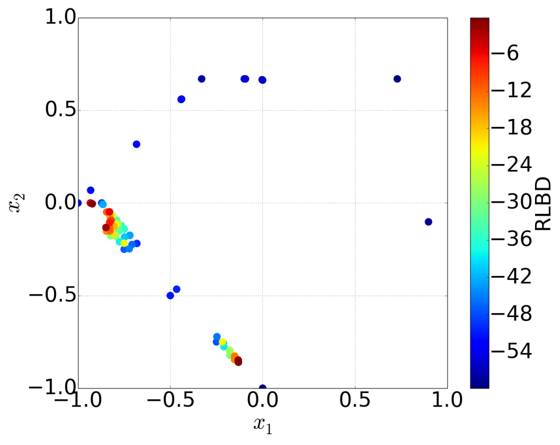
4.5.3.1 Theoretical time complexity

Primal problem The primal problem is a convex quadratic program with KN decision variables. The time complexity for the primal problem solving is then $O(K^3N^3)$ [121].

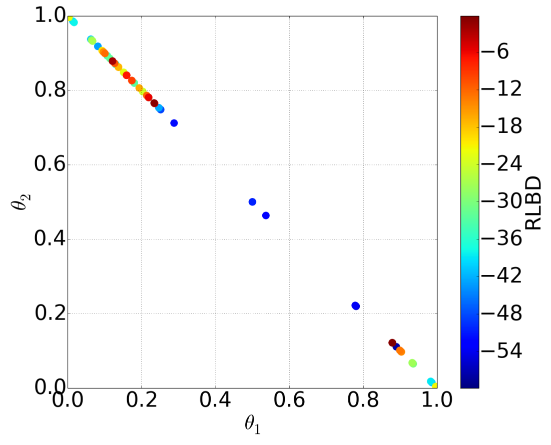
Preprocessing We address the cases of overlapping qualifying constraint cuts by sorting the rows of the $KN \cdot M$ qualifying constraint coefficient matrix and comparing the coefficients of



(a) Duality gap through the first 120 seconds.



(b) Optimal x of each iteration. The true x is (0, -1).



(c) Optimal θ of each iteration. The true θ is (0.22, 0.78).

Figure 4.5: Convergence and accuracy of our GOP algorithm on a synthetic data set on the scale of one feature, two subtypes, and ten samples.

adjacent rows. We first sort the KN rows of the qualifying constraint coefficient matrix using heapsort which takes $O(KN \cdot \log(KN))$ time on average. The algorithm subsequently passes through the rows of the matrix to identify all-zero coefficients and duplicate cuts; each pass takes $O(KN)$ time. We define $|\mathcal{A}'|$ as the number of unique qualifying constraints.

Unique region identification The interior point method that we used in the A-star search algorithm is a linear program of size $|\mathcal{A}'| \cdot MK$ with the time complexity of $O(|\mathcal{A}'| \cdot MK)$. The time complexity for enumerating the set of unique regions is $O(|\mathcal{A}'| \cdot (|\mathcal{A}'| \cdot MK))$, which exhibits polynomial behavior. The time complexity of the partial order A-star algorithm is polynomial in the best case and exponential in the worst case, depending on the heuristic. We define $|r(\mathcal{A}')|$ as the number of identified unique regions.

Relaxed dual problems There are $2MK + 1$ decision variables for each relaxed dual problem, so the time complexity for each is $O(M^3K^3)$. The total time for solving the relaxed dual problems is $O(|r(\mathcal{A}')| \cdot M^3K^3)$, which depends on the number of relaxed dual problems.

4.5.3.2 Empirical timing results

We constructed 12 synthetic data sets in a full-factorial arrangement with $M \in \{20, 40, 60, 80\}$, $K \in \{2\}$, and $N \in \{4, 5, 6\}$ and measured CPU time for each component of one iteration. For each arrangement, each element of the true x^* is:

$$x^*_{mk} = \begin{cases} 1 & \text{if } 0 \leqslant m < M/4,\ k = 0 \\ -1 & \text{if } M/4 \leqslant m < M/2,\ k = 1 \\ \mathcal{N}(0, 0.5^2) & \text{if } M/2 \leqslant m < M, \forall k \\ 0 & \text{otherwise.} \end{cases}$$

Here $\mathcal{N}(0, 0.5^2)$ is the sample from a Normal distribution by its mean 0 and standard deviation 0.5. For the true θ^*, θ^*_{kn} for $k = 0$ are n evenly spaced samples over the interval of $[0, 1]$; θ^*_{kn} for $k = 1$ are n evenly spaced samples over the interval of $[1, 0]$.

Table 4.1 shows that the time per iteration increases linearly with M when K and N are fixed. The time for solving all the relaxed dual problems increases as the number of samples increases. Note that we need to solve at most 2^{KN} relaxed dual problems per iteration, so the time per iteration increases nearly exponentially with KN when M is fixed at the worst case. Even though the step of solving all the relaxed dual problems takes more than 90% of the total time per iteration when the number of samples is 6, our algorithm is easily parallelized to solve the relaxed dual problems, allowing the algorithm to scale nearly linearly with the size of the data set.

Table 4.1: Timing profile of each component of the GOP algorithm for one iteration.

Scale		**Time (s)**					
M	**N**	**Primal**	**Pre**	**URI**	**Num**	**Dual**	**Total**
20	4	0.10	1.69	1.29	200	1.54 (33%)	4.62
40	4	0.12	1.91	1.72	202	1.69 (31%)	5.44
60	4	0.12	2.03	1.11	202	1.77 (35%)	5.03
80	4	0.13	2.39	2.05	232	3.70 (45%)	8.27
20	5	0.11	1.99	1.31	456	11.26 (77%)	14.67
40	5	0.11	2.07	1.37	485	11.45 (76%)	15.00
60	5	0.11	1.86	1.41	558	12.33 (78%)	15.71
80	5	0.12	2.23	1.26	650	17.96 (83%)	21.57
20	6	0.14	2.21	2.50	1152	65.71 (93%)	70.56
40	6	0.13	2.83	2.49	1250	67.08 (92%)	72.53
60	6	0.12	3.45	2.80	1255	69.00 (92%)	75.37
80	6	0.12	3.15	2.80	1309	77.62 (93%)	83.69

Primal: primal problem. Pre: preprocessing. URI: unique region identification. Num: number of relaxed dual problems. Dual: relaxed dual problems. Total: total time of one iteration. Here we have two subtypes. A single processor is used. Time for solving relaxed dual problems is highlighted in percentage.

4.6 Conclusions

We have presented a global optimization algorithm for a mixed membership matrix factorization problem. Our algorithm brings ideas from the global optimization community (Benders' decomposition and the GOP method) into contact with statistical inference problems for the first time. The cost of the global optimal solution is the need to solve a number of linear programs that grows exponentially in the number of so-called "connected" variables in the worst case – in this case the KN elements of θ. Many of these linear programs are redundant or yield optimal solutions that are greater than the current upper bound and thus not useful. A branch-and-bound framework [116] reduces the need to solve all possible relaxed dual problems by fathoming parts of the solution space We further mitigate this cost by developing an search algorithm for identifying and enumerating the true number of unique linear programs.

We are exploring the connections between GOP and the other alternating optimization algorithms such as the expectation maximization (EM) and variational EM algorithm. Since the complexity of GOP only depends on the connected variables, the graphical model structure connecting the complicating and non-complicating variables may be used to identify the worst-case complexity of the algorithm prior to running the algorithm. A factorized graph structure may provide an approximate, but computationally efficient algorithm based on GOP.

Additionally, because the Lagrangian function factorizes into the sum of Lagrangian functions for each sample in the data set, we may be able to update the parameters based on GOP for a selected subset of the data in an iterative or sequential algorithm. We are exploring the statistical consistency properties of such an update procedure.

Finally, we have derived an algorithm for particular loss functions for the sparsity constraint and objective function. The GOP framework can handle integer variables and thus may be used with an ℓ_0 counting “norm” rather than the ℓ_1 norm to induce sparsity. This would give us a mixed-integer biconvex program, but the conditions for the framework. Structured sparsity constraints can also be defined as is done for elastic-net extensions of LASSO regression. It may be useful to consider other loss functions for the objective function depending on the application.

Chapter 5

Conclusions and Outlook

5.1 Summary of contributions

This dissertation focuses on development of statistical and computational methods that address challenges in characterizing genomic heterogeneity in DNA next-generation sequencing and transcription data sets.

5.1.1 Rare variant detection

Next-generation sequencing enables the generation of thousands of millions of short reads in parallel fashion to reveal genomic heterogeneity in disease samples like cancer. In Chapter 3, we develop a novel hierarchical Bayesian statistical model and a variational EM algorithm to identify rare variants in heterogeneous next-generation sequencing data. Our algorithm is able to identify variants in a broad range of read depths and non-reference allele frequencies with high sensitivity and specificity. We validate our algorithm on an empirical data set and show comparable accuracy with other current variant detection methods. Furthermore, we apply our algorithm on a real longitudinal data set to detect variants in different time points in the course of yeast growth with limited glucose.

5.1.2 Intra-tumor heterogeneity

In Chapter 4, we derive a GOP algorithm that brings the global optimization algorithm into contact with the mixed membership matrix factorization problem. This includes a branch-

and-bound GOP algorithm that improves computational efficiency. As experimental results, we show that the GOP algorithm achieves the true optimal values on a simulated data set. We are able to distribute the relaxed dual problems that need to be solved per iteration to multiple processors and each relaxed dual problem can be solved in less than 1 second using any linear programming solver. Thus, the GOP algorithm can be run in parallel in key optimization steps, which allows the algorithm to scale nearly linearly with the scale of the data set. The GOP algorithmic development will generalize an exact statistical inference to a broad category of mixed membership models. It will be significant to understand the molecular mechanisms of subtype co-occurrence pattern and thus bring insights into personal-medicine treatment based on the distribution over multiple cellular subpopulations in an individual sample.

5.2 Challenges and opportunities for rare variant detection

A critical challenge for current statistical methods in rare variant detection is to reduce false positive calls. It is difficult to discern a true positive variant through statistical methods when the allele frequency of a true positive variant is close to the fraction of sequencing errors. Increasing the depth of coverage may reduce this false positive rate, but it can not be guaranteed. A possible solution is to estimate the pattern of sequencing errors as EBCall [20] has shown to distinguish false positives from true rare variants. Post-call filtering that consider parameters cut-offs may also be useful as VarScan2 [21] attempted.

Another challenge is to improve the efficiency of statistical methods on whole genome-wide sequence analysis. The size of whole-genome sequencing and whole-exome sequencing is considered large-scale and difficult to analyze, yet the value of this information and their applications are being successfully integrated to the clinical diagnostics and development of precision medicine [122, 123]. Inefficient variant detection in genome-wide sequencing data will prevent the effectiveness of translating the sequencing data into useful clinical knowledge. To approach this challenge, computational parallelization may largely help to enhance efficiency by distribution of multiple computing cores, but there remains room for improvement of scalability of statistical methods intrinsically.

Besides the challenges of accuracy and efficiency, another bottleneck for variant detection methods is reproducibility. It is difficult to test and reproduce the results of variant detection due to the insufficient information of input data, source code, and parameters settings [124]. For example, several issues, like the quality control of the data, the read depth requirements, and the confidence level in a statistical test, may influence the results of variant detection in the NGS data.

In summary, accurate detection of rare variants is important since rare variants will contribute to phenotypic divergence and complex diseases. Each detected variant can be either risk, protective, or neutral for a specific disease. Grouping of rare variants across genes has been used for rare variant association study to achieve high confidence level of disease-associated variants [125]. Recently, many studies have shown that rare variants are beneficial for clinical applications [126]. For example, rare variants identified by deep sequencing technologies were demonstrated to be associated with inflammatory bowel disease [127]. Another study revealed that multiple rare variants in *PCSK9* contribute to high-density lipoprotein cholesterol [128]. A therapeutic interference strategy targeting *PCSK9* has indicated a way of translating rare variants to the clinic [129].

5.3 Challenges and opportunities for genomic subtypes classification

Mixed membership models, such as the Gaussian Laplace Dirichlet model [96], and standard decomposition methods, such as non-negative matrix factorization [130], have been developed to discover the underlying genomic subtypes and infer the cooperativity and interference in molecular pathways. However, many of these algorithms only provide a local optimum or an approximated solution depending on random initializations. Thus, a major challenge is to provide an accurate estimation, i.e. a global optimum, of genomics subtypes for mixed samples.

Several open questions have been proposed and not yet been fully solved for molecular subtypes classification [6]. The tumor subtypes are determined by intra-tumor heterogeneity and evolutionary progression. But, the number of distinct molecular subtypes within a cancer type is still unclear. Also, intra-tumor heterogeneity is not only caused by distinct genomic subtypes, but is also influenced by stochastic factors, such as epigenetic events and protein instability. Another question is that how to demonstrate if a classification algorithm is more robust than alternative competing algorithms [131]. Cross validation can be used to evaluate the class assignment, but it is difficult to examine the performance of class discovery [6]. For example, research on classifying and clustering primary breast tumors by the TCGA has shown multiple results of subtypes for the same data set [132]. A cluster algorithm revealed 13 subtypes, while another algorithm based on semi-supervised PAM50 method revealed five subtypes [133]. Therefore, larger genomics data sets are required to quantitatively validate the performance of classification methods and interpret the answers.

In summary, heterogeneous tumor samples can be categorized using classification or clustering methods because tumor samples often consist a finite number of subclones and the molecular subtype signatures for each subclone can be computationally decomposed using their gene expression data. The clusters obtained by statistical or computational methods could present biologically meaningful subtypes. The genomic signatures of these subtypes can be used to enrich the previously discovered diseases-associated genes and establish pathways for different subtypes. In clinic, the dissected genomic subtype signatures will provide insights for clinicians to help improve prognostic and develop precision medicine. Several possible ways could help translate genomics research findings into clinical practice. First, development of robust statistical methods will pave the way for accurate predictions for genomic subtypes. Second, considerable large cohorts of longitudinal tumor samples will help robustly capture more distinct subtypes and more phenotypic heterogeneity [6]. Third, collection of multiple omics data types, such as RNA-seq, gene expression, and epigenetic level features, will assist to dissect novel molecular subtypes for clinical use. Thus, both improvement of statistical methods and generation of sufficient genomic data sets will help decipher the impact of genomic subtypes and develop effective ways for therapeutic treatment.

Appendix A

Derivation of the Variational EM Inference Algorithm

A.1 Evidence lower bound

The ELBO can be expanded as

$$\begin{aligned}\mathcal{L}(q,\phi) &= E_q\left[\log p\left(r,\mu,\theta|n;\phi\right)\right] - E_q\left[\log q\left(\mu,\theta\right)\right] \\ &= E_q\left[\log p\left(r|\theta,n\right)\right] + E_q\left[\log p\left(\theta|\mu;M\right)\right] + E_q\left[\log p\left(\mu;\mu_0,M_0\right)\right] \\ &- E_q\left[\log q\left(\mu\right)\right] - E_q\left[\log q\left(\theta\right)\right].\end{aligned} \tag{A.1}$$

We write out each component below.

$$\begin{aligned}E_q\left[\log p\left(r|\theta,n\right)\right] &= \sum_{j=1}^{J}\sum_{i=1}^{N} E_q\left[\log p\left(r_{ji}|\theta_{ji},n_{ji}\right)\right] \\ &= \sum_{j=1}^{J}\sum_{i=1}^{N} \log\left(\frac{\Gamma(n_{ji}+1)}{\Gamma(r_{ji}+1)\Gamma(n_{ji}-r_{ji}+1)}\right) \\ &\quad + \sum_{j=1}^{J}\sum_{i=1}^{N}\left\{r_{ji}E_q\left[\log\theta_{ji}\right] + (n_{ji}-r_{ji})E_q\left[\log(1-\theta_{ji})\right]\right\}\end{aligned} \tag{A.2}$$

$$\begin{aligned}E_q\left[\log p\left(\mu;\mu_0,M_0\right)\right] &= \sum_{j=1}^{J} E_q\left[\log p\left(\mu_j;\mu_0,M_0\right)\right] \\ &= J * \log\frac{\Gamma(M_0)}{\Gamma(\mu_0 M_0)\Gamma(M_0(1-\mu_0))} \\ &\quad + \sum_{j=1}^{J}\{(M_0\mu_0-1)E_q\left[\log\mu_j\right]\} \\ &\quad + \sum_{j=1}^{J}\{(M_0(1-\mu_0)-1)E_q\left[\log(1-\mu_j)\right]\}\end{aligned} \tag{A.3}$$

$$\begin{aligned}E_q\left[\log p\left(\theta|\mu;M\right)\right] &= \sum_{j=1}^{J}\sum_{i=1}^{N} E_q\left[\log p\left(\theta_{ji}|\mu_j;M_j\right)\right] \\ &= N * \sum_{j=1}^{J} E_q\left[\log\left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma(M_j(1-\mu_j))}\right)\right] \\ &\quad + \sum_{j=1}^{J}\sum_{i=1}^{N}\{M_j E_q\left[\mu_j\right] E_q\left[\log\theta_{ji}\right] - E_q\left[\log\theta_{ji}\right]\} \\ &\quad + \sum_{j=1}^{J}\sum_{i=1}^{N}\{\left(M_j - 1 - M_j E_q\left[\mu_j\right]\right) E_q\left[\log\left(1-\theta_{ji}\right)\right]\}\end{aligned} \tag{A.4}$$

Therefore, we need to compute the following expectations with respect to the variational distribution: $E_q\left[\log\theta_{ji}\right]$, $E_q\left[\log\left(1-\theta_{ji}\right)\right]$, $E_q\left[\log\mu_j\right]$, $E_q\left[\log(1-\mu_j)\right]$, $E_q\left[\mu_j\right]$, and $E_q\left[\log\left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma(M_j(1-\mu_j))}\right)\right]$.

We select the functional forms for the variational distributions $q(\theta)$ and $q(\mu)$ to facilitate these expected value computations.

A.2 Variational distributions

Since θ and r are conjugate pairs, the posterior distribution of θ_{ji} is a Beta distribution,

$$p(\theta_{ji}|r_{ji},n_{ji},\mu_j,M_j) \sim \text{Beta}(r_{ji}+M_j\mu_j, n_{ji}-r_{ji}+M_j(1-\mu_j)). \tag{A.5}$$

Therefore, we propose a Beta distribution with parameter vector δ_{ji} as variational distribution,

$$\theta_{ji} \sim \text{Beta}(\delta_{ji1}, \delta_{ji2}).$$

The posterior distribution of μ_j is given by its Markov blanket,

$$p(\mu_j|\theta_{ji}, M_j, \mu_0, M_0) \propto p(\mu_j|\mu_0, M_0)p(\theta_{ji}|\mu_j, M_j). \tag{A.6}$$

This is not in the form of any known distribution. But, since the support of μ_j is $[0, 1]$, we propose a Beta distribution with parameter vector γ_j as variational distribution,

$$\mu_j \sim \text{Beta}(\gamma_{j1}, \gamma_{j2}).$$

Given these variational distributions, we have

$$\begin{aligned} E_q\left[\log \theta_{ji}\right] &= \psi(\delta_{ji1}) - \psi(\delta_{ji1} + \delta_{ji2}) \\ E_q\left[\log\left(1 - \theta_{ji}\right)\right] &= \psi(\delta_{ji2}) - \psi(\delta_{ji1} + \delta_{ji2}) \\ E_q\left[\mu_j\right] &= \frac{\gamma_{j1}}{\gamma_{j1} + \gamma_{j2}} \\ E_q\left[\log \mu_j\right] &= \psi(\gamma_{j1}) - \psi(\gamma_{j1} + \gamma_{j2}) \\ E_q\left[\log(1 - \mu_j)\right] &= \psi(\gamma_{j2}) - \psi(\gamma_{j1} + \gamma_{j2}), \end{aligned} \tag{A.7}$$

where ψ is the digamma function.

Since there is no analytical representation for $E_q\left[\log\left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma(M_j(1-\mu_j))}\right)\right]$, we must resort to numerical integration,

$$\begin{aligned} &E_q\left[\log\left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma((1-\mu_j)M_j)}\right)\right] = \\ &\int_0^1 q(\mu_j; \gamma_{j1}, \gamma_{j2}) \log\left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma((1-\mu_j)M_j)}\right) d\mu_j. \end{aligned} \tag{A.8}$$

Here $q(\mu_j; \gamma_{j1}, \gamma_{j2})$ is the probability density function of the Beta distribution that is calculated using the Python built-in function `scipy.stats.beta.pdf`,

and $\log\left(\frac{\Gamma(M_j)}{\Gamma(\mu_j M_j)\Gamma((1-\mu_j)M_j)}\right)$ is calculated using the Python built-in function `scipy.special.betaln`. Unfortunately, this numerical integration step is computationally expensive. Finally, the entropy terms can be computed as follows,

$$
\begin{aligned}
E_q\left[\log q\left(\mu\right)\right] &= \sum_{j=1}^{J} E_q\left[\log q(\mu_j)\right] \\
&= -\sum_{j=1}^{J}\left\{\log(B(\gamma_{j1},\gamma_{j2})) - (\gamma_{j1}-1)\psi(\gamma_{j1})\right\} \\
&\quad + \sum_{j=1}^{J}\left\{-(\gamma_{j2}-1)\psi(\gamma_{j2}) + (\gamma_{j1}+\gamma_{j2}-2)\psi(\gamma_{j1}+\gamma_{j2})\right\};
\end{aligned}
\tag{A.9}
$$

and

$$
\begin{aligned}
E_q\left[\log q\left(\theta\right)\right] &= \sum_{j=1}^{J}\sum_{i=1}^{N} E_q\left[\log q(\theta_{ji})\right] \\
&= -\sum_{j=1}^{J}\sum_{i=1}^{N}\left\{\log(B(\delta_{ji1},\delta_{ji2})) - (\delta_{ji1}-1)\psi(\delta_{ji1})\right\} \\
&\quad + \sum_{j=1}^{J}\sum_{i=1}^{N}\left\{-(\delta_{ji2}-1)\psi(\delta_{ji2}) + (\delta_{ji1}+\delta_{ji2}-2)\psi(\delta_{ji1}+\delta_{ji2})\right\}.
\end{aligned}
\tag{A.10}
$$

Appendix B

GOP Inference Details

B.1 Derivation of relaxed dual problem constraints

We form the Lagrangian function for the primal problem that is presented in Section 4.3.2. The derivation of the linearized Lagrangian function is used to create the constraint set of the relaxed dual problem.

The Lagrangian function is the sum of the Lagrangian functions for each sample,

$$L(y, \theta, x, \lambda) = \sum_{i=1}^{n} L(y_i, \theta_i, x, \lambda_i, \mu_i), \tag{B.1}$$

and the Lagrangian function for a single sample is

$$L(y_i, \theta_i, x, \lambda_i, \mu_i) = y_i^T y_i - 2y_i^T x\theta_i + \theta_i^T x^T x\theta_i - \lambda_i(\theta_i^T 1_K - 1) - \mu_i^T \theta_i. \tag{B.2}$$

Here, the Lagrange multipliers are $\mu \in \mathbb{R}_+^{K\times N}$ and $\lambda \in \mathbb{R}^N$. We see that the Lagrangian function is biconvex in x and θ_i. We develop the constraints for a single sample for the remainder.

B.1.1 Linearized Lagrangian function with respect to x

Casting x as a vector and rewriting the Lagrangian function gives

$$L(y_i, \theta_i, \bar{x}, \lambda_i, \mu_i) = a_i - 2b_i^T \bar{x} + \bar{x}^T C_i \bar{x} - \lambda_i(\theta_i^T 1_K - 1) - \mu_i^T \theta_i, \tag{B.3}$$

where $\bar{x}$ is formed by stacking the columns of x in order. The coefficients are formed such that

$$\begin{aligned} a &= y_i^T y_i, \\ b_i^T \bar{x} &= y_i^T x \theta_i, \\ \bar{x}^T C_i \bar{x} &= \theta_i^T x^T x \theta_i. \end{aligned}$$

The linear coefficient matrix is the $KM \times 1$ vector,

$$b_i = \left[y_i \theta_{1i}, \cdots, y_i \theta_{Ki} \right].$$

The quadratic coefficient is the $KM \times KM$ and block matrix

$$C_i = \begin{bmatrix} \theta_{1i}^2 I_M & \cdots & \theta_{1i}\theta_{Ki} I_M \\ \vdots & \ddots & \vdots \\ \theta_{Ki}\theta_{1i} I_M & \cdots & \theta_{Ki}^2 I_M \end{bmatrix}.$$

The Taylor series approximation about x_0 is

$$L(y_i, \theta_i, \bar{x}, \lambda_i, \mu_i)\Big|_{\bar{x}_0}^{\text{lin}} = L(y_i, x_0, \theta_i, \lambda_i, \mu_i) + (\nabla_x L\big|_{x_0})^T (x - x_0). \tag{B.4}$$

The gradient with respect to x is

$$\nabla_x L(y_i, \theta_i, \bar{x}, \lambda_i, \mu_i) = -2b_i + 2C_i \bar{x}. \tag{B.5}$$

Plugging the gradient into the Taylor series approximation gives

$$L(y_i, \theta_i, \bar{x}, \lambda_i)\Big|_{\bar{x}_0}^{\text{lin}} = a_i - 2b_i^T \bar{x}_0 + \bar{x}_0^T C_i \bar{x}_0 - \lambda_i(\theta_i^T 1_K - 1) - \mu_i^T \theta_i + (-2b_i + 2C_i \bar{x}_0)^T (\bar{x} - \bar{x}_0). \tag{B.6}$$

Simplifying the linearized Lagrangian function gives

$$L(y_i, \theta_i, \bar{x}, \lambda_i, \mu_i)\Big|_{\bar{x}_0}^{\text{lin}} = (y_i^T y_i - \bar{x}_0^T C_i \bar{x}_0 - \lambda_i(\theta_i^T 1_K - 1) - \mu_i^T \theta_i) - 2b_i^T \bar{x} + 2\bar{x}_0^T C_i \bar{x}. \tag{B.7}$$

Finally, we write the linearized Lagrangian using the matrix form of x_0,

$$L(y_i,\theta_i,x,\lambda_i,\mu_i)\Big|_{x_0}^{\text{lin}} = y_i^T y_i^T - \theta_i^T x_0^T x_0 \theta_i - 2y_i^T x\theta_i + 2\theta_i^T x_0^T x\theta_i - \lambda_i(\theta_i^T 1_K - 1) - \mu_i^T\theta_i. \quad \text{(B.8)}$$

While the original Lagrangian function is convex in θ_i for a fixed x, the linearized Lagrangian function is not necessarily convex in θ_i. This can be seen by collecting the quadratic, linear and constant terms with respect to θ_i,

$$L(y_i,\theta_i,x,\lambda_i,\mu_i)\Big|_{x_0}^{\text{lin}} = (y_i^T y_i^T + \lambda_i) + (-2y_i^T x - \lambda_i 1_K^T - \mu_i^T)\theta_i + \theta_i^T(2x_0^T x - x_0^T x_0)\theta_i. \quad \text{(B.9)}$$

Now, if and only if $2x_0^T x - x_0^T x_0 \succeq 0$ is positive semidefinite, then $L(y_i,\theta_i,x,\lambda_i,\mu_i)\Big|_{x_0}^{\text{lin}}$ is convex. The condition is satisfied at $x = x_0$ but may be violated at some other value of x.

B.1.2 Linearized Lagrangian function with respect to θ_i

Now, we linearize (B.7) with respect to θ_i. Using the Taylor series approximation with respect to θ_{0i} gives

$$L(y_i,\theta_i,x,\lambda_i,\mu_i)\Big|_{x_0,\theta_{0i}}^{\text{lin}} = L(y_i,\theta_{0i},x,\lambda_i,\mu_i)\Big|_{x_0}^{\text{lin}} + \left(\nabla_{\theta_i} L(y_i,\theta_i,x,\lambda_i,\mu_i)\Big|_{x_0}^{\text{lin}}\Big|_{\theta_{0i}}\right)^T (\theta_i - \theta_{0i}). \quad \text{(B.10)}$$

The gradient for this Taylor series approximation is

$$g_i(x) \triangleq \nabla_{\theta_i} L(y_i,\theta_i,x,\lambda_i,\mu_i)\Big|_{x_0}^{\text{lin}}\Big|_{\theta_{0i}} = -2x_0^T x_0\theta_{0i} - 2x^T y_i + 2(x_0^T x + x^T x_0)\theta_{0i} - \lambda_i 1_K - \mu_i, \quad \text{(B.11)}$$

where $g_i(x)$ is the vector of K qualifying constraints associated with the Lagrangian function. The qualifying constraint is linear in x.

Plugging the gradient into the approximation gives

$$\begin{aligned} L(y_i, \theta_i, x, \lambda_i, \mu_i)\Bigg|_{x_0,\theta_{0i}}^{\text{lin}} &= y_i^T y_i^T - \theta_{0i}^T x_0^T x_0 \theta_{0i} - 2y_i^T x \theta_{0i} + 2\theta_{0i}^T x_0^T x \theta_{0i} - \lambda_i(\theta_{0i}^T 1_K - 1) - \mu_i^T \theta_{0i} \\ &+ (-2x_0^T x_0 \theta_{0i} - 2x^T y_i + 2(x_0^T x + x^T x_0)\theta_{0i} - \lambda_i 1_K - \mu_i)^T (\theta_i - \theta_{0i}) \end{aligned} \tag{B.12}$$

The linearized Lagrangian function is bi-linear in x and θ_i.

Finally, simplifying the linearized Lagrangian function gives

$$\begin{aligned} L(y_i, \theta_i, x, \lambda_i, \mu_i)\Bigg|_{x_0,\theta_{0i}}^{\text{lin}} &= y_i^T y_i^T + \theta_{0i}^T x_0^T x_0 \theta_{0i} - 2\theta_{0i}^T x_0^T x_0 \theta_i - \lambda_i(1_K^T \theta_i - 1) - \mu_i^T \theta_i \\ &- 2\theta_{0i}^T x^T x_0 \theta_{0i} - 2y_i^T x \theta_i + 2\theta_{0i}^T (x_0^T x + x^T x_0)\theta_i. \end{aligned} \tag{B.13}$$

B.2 Proof of biconvexity

To prove the optimization problem is biconvex, first we show the feasible region over which we are optimizing is biconvex. Then, we show the objective function is biconvex by fixing θ and showing convexity with respect to x, and then vice versa.

B.2.1 The constraints form a convex feasible region

Our constraints can be written as

$$||x||_1 \leqslant P \tag{B.14}$$

$$\sum_{k=1}^{K} \theta_{ki} = 1 \; \forall i \tag{B.15}$$

$$0 \leqslant \theta_{ki} \leqslant 1 \; \forall (k, i). \tag{B.16}$$

The inequality constraint (B.14) is convex if either x or θ is fixed, because any norm is convex. The equality constraints (B.15) is an affine combination that is still affine if either x or θ is fixed. Every affine set is convex. The inequality constraint (B.16) is convex if either x or θ is fixed, because θ is a linear function.

B.2.2 The objective is convex with respect to θ

We prove the objective is a biconvex function using the following two theorems.

Theorem B.2.1 *Let $A \subseteq \mathbb{R}^n$ be a convex open set and let $f : A \to \mathbb{R}$ be twice differentiable. Write $H(x)$ for the Hessian matrix of f at $x \in A$. If $H(x)$ is positive semidefinite for all $x \in A$, then f is convex ([121]).*

Theorem B.2.2 *A symmetric matrix A is positive semidefinite (PSD) if and only if there exists B such that $A = B^T B$ ([134]).*

The objective of our problem is,

$$\begin{aligned} f(y, x, \theta) = ||y - x\theta||_2^2 &= (y - x\theta)^T (y - x\theta) && \text{(B.17)} \\ &= (y^T - \theta^T x^T)(y - x\theta) && \text{(B.18)} \\ &= y^T y - y^T x\theta - \theta^T x^T y + \theta^T x^T x\theta. && \text{(B.19)} \end{aligned}$$

The objective function is the sum of the objective functions for each sample.

$$\begin{aligned} f(y, x, \theta) &= \sum_{i=1}^{N} f(y_i, x, \theta_i) && \text{(B.20)} \\ &= \sum_{i=1}^{N} y_i^T y_i - 2y_i^T x\theta_i + \theta_i^T x^T x\theta_i. && \text{(B.21)} \end{aligned}$$

The gradient with respect to θ_i is

$$\begin{aligned} \nabla_{\theta_i} f(y_i, x, \theta_i) &= -2y_i^T x + (x^T x + (x^T x)^T)\theta_i && \text{(B.22)} \\ &= -2x^T y_i + 2x^T x\theta_i. && \text{(B.23)} \end{aligned}$$

Taking the second derivative with respect to θ_i to get Hessian matrix, we obtain

$$\begin{aligned} \nabla_{\theta_i}^2 f(y_i, x, \theta_i) &= \triangledown_{\theta_i}(-2x^T y_i + 2x^T x\theta_i) && \text{(B.24)} \\ &= 2\triangledown_{\theta_i}(x^T x\theta_i) && \text{(B.25)} \\ &= 2(x^T x)^T && \text{(B.26)} \\ &= 2x^T x. && \text{(B.27)} \end{aligned}$$

The Hessian matrix $\nabla^2_{\theta_i} f(y_i, x, \theta_i)$ is positive semidefinite based on Theorem B.2.2. Then, we have $f(y_i, x, \theta_i)$ is convex in θ_i based on Theorem B.2.1. The objective $f(y, x, \theta)$ is convex with respect to θ, because the sum of convex functions, $\sum_{i=1}^{N} f(y_i, x, \theta_i)$, is still a convex function.

B.2.3 The objective is convex with respect to x

The objective function for sample i is

$$f(y_i, x, \theta_i) = y_i^T y_i - 2y_i^T x\theta_i + \theta_i^T x^T x\theta_i. \tag{B.28}$$

We cast x as a vector $\bar{x}$, which is formed by stacking the columns of x in order. We rewrite the objective function as

$$f(y_i, \bar{x}, \theta_i) = a_i - 2b_i^T \bar{x} + \bar{x}^T C_i \bar{x}. \tag{B.29}$$

The coefficients are formed such that

$$a = y_i^T y_i, \tag{B.30}$$
$$b_i^T \bar{x} = y_i^T x\theta_i, \tag{B.31}$$
$$\bar{x}^T C_i \bar{x} = \theta_i^T x^T x\theta_i. \tag{B.32}$$

The linear coefficient matrix is the $KM \times 1$ vector

$$b_i = [y_i\theta_{1i}, ..., y_i\theta_{Ki}]. \tag{B.33}$$

The quadratic coefficient is the $KM \times KM$ and block matrix

$$C_i = \begin{bmatrix} \theta_{1i}^2 I_M & \cdots & \theta_{1i}\theta_{Ki} I_M \\ \vdots & \ddots & \vdots \\ \theta_{Ki}\theta_{1i} I_M & \cdots & \theta_{Ki}^2 I_M \end{bmatrix}. \tag{B.34}$$

The gradient with respect to $\bar{x}$

$$\nabla_{\bar{x}} f(y_i, \bar{x}, \theta_i) = -2b_i + 2C_i\bar{x}. \tag{B.35}$$

Take second derivative to get Hessian matrix,

$$\nabla_{\bar{x}^2} f(y_i, \bar{x}, \theta_i) = 2C_i^T \quad \text{(B.36)}$$

$$= 2(\theta_i \theta_i^T)^T \quad \text{(B.37)}$$

$$= 2(\theta_i^T)^T (\theta_i^T). \quad \text{(B.38)}$$

The Hessian matrix $\nabla_{\bar{x}}^2 f(y_i, \bar{x}, \theta_i)$ is positive semidefinite based on Theorem B.2.2. Then, we have $f(y_i, \bar{x}, \theta_i)$ is convex in $\bar{x}$ based on Theorem B.2.1. The objective $f(y, x, \theta)$ is convex with respect to x, because the sum of convex functions, $\sum_{i=1}^{N} f(y_i, x, \theta_i)$, is still a convex function.

The objective is biconvex with respect to both x and θ. Thus, we have a biconvex optimization problem based on the proof of convexity of the constraints, and biconvexity of the objective.

B.3 A-star search algorithm

In this procedure, first we remove all the duplicate and all-zero coefficients hyperplanes to get unique hyperplanes. Then we start from a specific region r and put it into a open set. Open set is used to maintain a region list which need to be explored. Each time we pick one region from the open set to find adjacent regions. Once finishing the step of finding adjacent regions, region r will be moved into a closed set. Closed set is used to maintain a region list which already be explored. Also, if the adjacent region is a newly found one, it also need to be put into the open set for exploring. Finally, once the open set is empty, regions in the closed set are all the unique regions, and the number of the unique regions is the length of the closed set. This procedure begins from one region and expands to all the neighbors until no new neighbor existed.

The overview of the A-star search algorithm to identify unique regions is shown in Algorithm 2.

Hyperplane filtering Assuming there are two different hyperplanes H_i and H_j represented by $A_i = \{a_{i,0}, ..., a_{i,MK}\}$ and $A_j = \{a_{j,0}, ..., a_{j,MK}\}$. We take these two hyperplanes duplicated when

$$\frac{a_{i,0}}{a_{j,0}} = \frac{a_{i,1}}{a_{j,1}} = ... = \frac{a_{i,MK}}{a_{j,MK}} = \frac{\sum_{l=0}^{MK} a_{i,l}}{\sum_{l=0}^{MK} a_{j,l}}, a_{j,l}! = 0 \quad \text{(B.39)}$$

Algorithm 2 A-star Search Algorithm

```
1: Sort the rows of the $KN$ x $M$ qualifying constraint coefficient matrix.
2: Compare adjacent rows of the qualifying constraint coefficient matrix and eliminate duplicate rows.
3: Eliminate rows of the qualifying constraint coefficient matrix with all-zero coefficients.
4: Determine the list of unique qualifying constraints by pairwise test.
5: Set $S$ and $|\mathcal{A}'|$ to the set of unique, non-trivial qualifying constraints and the number of them.
6: Initialize a region $root$ using an interior point method (Algorithm 3).
7: Put region $root$ into the open set.
8: if open set is not empty then
9:     Get a region $R$ from the open set.
10:    Calculate the adjacent regions set $R\_adj$ (Algorithm 4).
11:    Put region $R$ into the closed set.
12:    for each region $r$ in $R\_adj$ do
13:        if $r$ is not in the open set $and$ not in the closed set then
14:            Put region $r$ into the open set.
15:        end if
16:    end for
17: end if
18: Reflect the sign of the regions in the close set.
19: Get all the regions represented by string of 0 and 1.
```

This can be converted to

$$|\sum_{l=0}^{MK} a_{i,l} \cdot a_{j,n} - \sum_{l=0}^{MK} a_{j,l} \cdot a_{i,n}| \leqslant \tau, \forall\, n\epsilon[0, MK], \tag{B.40}$$

where threshold τ is a very small positive value.

We eliminate a hyperplane H_i represented by $A_i = \{a_{i,0}, ..., a_{i,MK}\}$ from hyperplane arrangement $\mathcal{A}$ if the coefficients of A_i are all zero,

$$|a_{i,j}| \leqslant \tau, \forall\, a_{i,j}\epsilon A_i, j\epsilon[0, MK] \tag{B.41}$$

$\mathcal{A}'$ is the reduced arrangement and $A'x = b$ are the equations of unique hyperplanes.

Interior point method An interior point is found by solving the following optimization problem:

$$
\begin{aligned}
&\text{maximize } z \\
&\text{subject to } -A_i'x + z \leqslant b_i, \text{if } \theta_i^B = 0 && \text{(B.42)} \\
&\qquad\qquad A_i'x + z \leqslant -b_i, \text{if } \theta_i^B = 1 && \text{(B.43)} \\
&\qquad\qquad z > 0. && \text{(B.44)}
\end{aligned}
$$

Algorithm 3 Interior Point Method

```
1: Generate $2^{|\mathcal{A}'|}$ different strings using $0$ and $1$.
2: for each $s$ in the strings do
3:     Solve an optimization problem to get an interior point.
4:     if Get a interior point then
5:         Get the $root$ region represented by $0$ and $1$.
6:     end if
7: end for
```

Algorithm 4 Get Adjacent Regions

```
1: Initialize an empty set $SH$ for strict hyperplanes.
2: Initialize an adjacent region set $ADJ$.
3: # Find out all the strict hyperplanes for region $R$.
4: for each hyperplane $H$ of $|\mathcal{A}'|$ hyperplanes do
5:     Pick one hyperplane $H$ from all the hyperplanes defining region R.
6:     Flip the sign of $H$ to get $\neg H$.
7:     Form a new hyperplane arrangement $\neg\mathcal{A}'$ with $\neg H$.
8:     Solve the problem to get an interior point constrained by $\neg\mathcal{A}'$.
9:     if the interior point is not Non then
10:        $H$ is a strict hyperplane and put into set $SH$.
11:    else
12:        $H$ is a redundant hyperplane.
13:    end if
14: end for
15: # Find out all the adjacent regions for region $R$.
16: for each strict hyperplane $sh$ in set $SH$ do
17:     Take the opposite sign $\neg sh$ of $sh$.
18:     Form a adjacent region $adj$ based on $\neg sh$ and all the else hyperplanes.
19:     Put $adj$ into set $ADJ$.
20: end for
```

Appendix C

Source Code

C.1 Variational inference RVD code

```python
# -*- coding: utf-8 -*-
from __future__ import print_function
from __future__ import division

import numpy as np

import scipy.stats as ss
import scipy.optimize as so
from scipy.special import gammaln, psi, betaln
from scipy import linalg, integrate
from itertools import repeat

#import pandas as pd
import multiprocessing as mp
import h5py
import tempfile
import logging
import time
from datetime import datetime
from datetime import date
import warnings
import pdb
import re
from timeit import default_timer as timer

def main():
    log_level = logging.DEBUG # default logging level
    logging.basicConfig(level=log_level, \
       format='%(levelname)s:%(module)s:%(message)s')

    ## Generate simulation data
    J = 10 # number of positions
    N = 3 # number of replicates

    n = np.empty([N,J], dtype=np.int64)
    n.fill(1000)

    # Set model parameters
    # can be estimated using function of "estimate_mom".
    phi = {'M0':100, 'mu0':0.1, 'M':[1000]*J}

    ''' Case:
    Variational EM algorithm for maximizing ELBO '''
    (r, theta, mu)=generate_sample(phi, N, J, n, seedint=20150928)

    (phiHat, qHat)=ELBO_opt(r, n, seed = 20150928, pool = 60)
    save_model('case_model.hdf5', r, n, phiHat, qHat)

    ''' Control:
    Variational EM algorithm for maximizing ELBO '''
    (r, theta, mu)=generate_sample(phi, N, J, n, seedint=20150928)

    (phiHat, qHat) = ELBO_opt(r, n, seed = 20150928, pool = 60)
    save_model('control_model.hdf5', r, n, phiHat, qHat)

    ''' Hypotheses testing for variant detection'''
    test('case_model.hdf5','control_model.hdf5')


def test(caseHDF5Name, controlHDF5Name, alpha=0.05, tau=0, \
        chi2=False, outputFile=None):
    # pdb.set_trace()
    caseR,caseN,casephi,caseq,loc,refb = load_model(caseHDF5Name)
    casegam = caseq['gam']
    controlR, controlN, controlphi, controlq, _, _ = \
         load_model(controlHDF5Name)
    controlgam = controlq['gam']

    (N,J) = np.shape(caseR)[0:2]

    def beta_mean(p):
        return p[0]*1.0/np.sum(p)

    def beta_var(p):
        s = np.sum(p)
        return p[0]*p[1]/(s**2*(s+1))

    # pdb.set_trace()
    bayescall = []
    for j in xrange(J):
        mu = (beta_mean(casegam[j,:]) - casephi['mu0'])- \
             (beta_mean(controlgam[j,:])-controlphi['mu0'])
        sigma = np.sqrt(beta_var(casegam[j,:]) \
                +beta_var(controlgam[j,:]))
        z = (tau - mu)/sigma
        p = ss.norm.cdf(z)
        # pdb.set_trace()
        bayescall.append(p[0]<alpha)

    ## combine the chi2 goodness of fit test
    if chi2:
        chi2call, chi2P = chi2combinetest(caseR, caseN, bayescall)
        call = np.logical_and(bayescall,chi2call)
    else:
        call = bayescall

    if outputFile is not None:

        vcfFilename = outputFile+'.vcf'

        write_dualvcf(vcfFilename, loc, call, refb, controlR, \
                      controlN, caseR, caseN)
        # output hdf5 file
        h5Filename = outputFile +'.hdf5'
        h5file = h5py.File(h5Filename, 'w')

        h5file.create_dataset('call', data=call)
        h5file.create_dataset('refb', data=refb)
        h5file.create_dataset('loc', data=loc,
                chunks=True, fletcher32=True, compression='gzip')
        h5file.create_dataset('controlN', data=controlN,
                chunks=True, fletcher32=True, compression='gzip')
        h5file.create_dataset('caseN', data=caseN,
                chunks=True, fletcher32=True, compression='gzip')
        h5file.create_dataset('controlR', data=controlR,
                chunks=True, fletcher32=True, compression='gzip')
        h5file.create_dataset('caseR', data=caseR,
                    chunks=True, fletcher32=True, compression='gzip')
        if chi2:
            h5file.create_dataset('chi2call', data=chi2call,
                    chunks=True, fletcher32=True, compression='gzip')
        h5file.create_dataset('bayescall', data=bayescall)
        h5file.close()


    ## output the results
def write_dualvcf(outputFile, loc, call, refb, controlR=None, \
                    controlN=None, caseR=None, caseN=None):

    controlR = np.median(controlR,0)
    caseR = np.median(caseR,0)
    '''
    Write high confidence variant calls from somatic test
    when there are both control and case sample to VCF 4.2 file.
    '''
    J = len(loc)

    today=date.today()

    chrom = [x.split(':')[0][3:] for x in loc]
    pos = [int(x.split(':')[1]) for x in loc]

    vcfF = open(outputFile,'w')

    print("##fileformat=VCFv4.1", file=vcfF)
    print("##fileDate=%0.4d%0.2d%0.2d" \
         % (today.year, today.month, today.day), file=vcfF)

    print("##source=rvd2", file=vcfF)

    print('##PosteriorTestSample= control-case-paired_sample.', \
          file=vcfF)

    uniquechrom = set(chrom)
    uniquechrom = list(uniquechrom)

    for i in xrange(len(uniquechrom)):
        seq = [idx for idx, name in enumerate(chrom) \
                 if name==uniquechrom[i]]
        seqlen = len(seq)
        print("##contig=<ID=%(chr)s,length=%(seqlen)d>" \
             %{'chr': uniquechrom[i],'seqlen': seqlen}, file=vcfF)


    print("##INFO=<ID=COAF,Number=1,Type=Float, \
          Description=\"Control Allele Frequency\">", file=vcfF)
    print("##INFO=<ID=CAAF,Number=1,Type=Float, \
          Description=\"Case Allele Frequency\">", file=vcfF)

    print("##FORMAT=<ID=AU,Number=1,Type=Integer, \
          Description=\"Number of 'A' alleles\">", file=vcfF)
    print("##FORMAT=<ID=CU,Number=1,Type=Integer, \
          Description=\"Number of 'C' alleles\">", file=vcfF)
    print("##FORMAT=<ID=GU,Number=1,Type=Integer, \
          Description=\"Number of 'G' alleles\">", file=vcfF)
    print("##FORMAT=<ID=TU,Number=1,Type=Integer, \
          Description=\"Number of 'T' alleles\">", file=vcfF)

    print("#CHROM\tPOS\tID\tREF\tALT\tQUAL\tFILTER \
            \tINFO\tFORMAT\tNormal\tCase", file=vcfF)

    for i in xrange(J):
        # pdb.set_trace()
        if call[i]:
            # restore R
            actg = ['A','C','G','T']

            idx = actg.index(refb[i])
            caseR4 = np.zeros(4)
            controlR4 = np.zeros(4)
            caseR4[idx] = np.median(caseN[:,i])-np.sum(caseR[i,:])
            controlR4[idx] = np.median(controlN[:,i]) \
                             -np.sum(controlR[i,:])
            for d in xrange(idx):
               caseR4[d] = caseR[i,d]
               controlR4[d] = controlR[i,d]
            for d in xrange(3-idx):
               caseR4[d+idx+1] = caseR[i,d+idx]
               controlR4[d+idx+1] = controlR[i,d+idx]

            print ("chr%s\t%d\t.\t%s\t.\t.\tPASS\t.\tAU:CU:GU:TU\t\
                  %d:%d:%d:%d\t%d:%d:%d:%d" % (chrom[i], pos[i], \
             refb[i], controlR4[0], controlR4[1], \
                  controlR4[2], controlR4[3], caseR4[0], \
             caseR4[1], caseR4[2], caseR4[3]), file=vcfF)

    vcfF.close()
```

```python


def chi2combinetest(R, N, bayescall = 1, pvalue = 0.05):

    nRep = R.shape[0]
    J = R.shape[1]
    chi2Prep = np.zeros((J,nRep))
    chi2P = np.zeros((J,1))
    for j in xrange(J):
            chi2Prep[j,:] = np.array([chi2test(R[i,j,:] ) \
            for i in xrange(nRep)] )
            if np.any(np.isnan(chi2Prep[j,:])):
                chi2P[j] = np.nan
            else:
                # combine p-values using Fisher's Method
                chi2P[j]=1-ss.chi2.cdf(-2*np.sum(np.log \
                (chi2Prep[j,:] + np.finfo(float).eps)), 2*nRep)

    nbayescall = sum(bayescall)
    if nbayescall < 1:
        nbayescall = 1

    #Benjamini-Hochberg method FWER control
    if np.median(N) > 500:
        chi2call = chi2P < pvalue/nbayescall
    else:
        chi2call = chi2P < pvalue

    chi2call = chi2call.flatten()
    chi2P = chi2P.flatten()

    return  chi2call, chi2P


def chi2test(X, lamda=2.0/3, pvector=np.array([1.0/3]*3)):
    """ Do chi2 test to decide how well the error reads fits
        uniform multinomial distribution. P-value returned.
        lamda=1 Pearson's chi-square
        lamda=0 the log likelihood ratio statistic/ G^2
        lamda=-1/2 Freeman-Tukey's F^2
        lamda=-1  Neyman modified chi-square
        lamda=-2  modified G^2
    """
    X=np.array(X)

    nsum=np.sum(X)
    # return NaN if there are no counts
    if nsum == 0: return np.nan
    E=nsum*pvector


    if lamda==0 or lamda==-1:
        C=2.0*np.sum(X*np.log(X*1.0/E))
    else:
        C=2.0/(lamda*(lamda+1))*np.sum(X*((X*1.0/E)**lamda-1))

    df=len(pvector)-1
    #p=scipy.special.gammainc(C,df)
    # p=1-gammainc(df/2,C/2)
    p = 1 - ss.chi2.cdf(C, df)
    return(p)



def generate_sample(phi, N=3, J=100, n=100, seedint=None):
    """Returns a sample with n reads, N replicates, and
    J locations. The parameters of the model are in the
    structure phi.
    """

    if seedint is not None:
        np.random.seed(seedint)

    #TODO: test for size of n and make an array if a scalar

    # Draw J location-specific error rates from a Beta
    alpha0 = phi['M0']*phi['mu0']
    beta0 = phi['M0']*(1-phi['mu0'])
    mu = ss.beta.rvs(alpha0, beta0, size=J)

    # Draw sample error rate and error count
    theta=np.zeros((N,J))
    r = np.zeros((N,J))
    for j in xrange(0, J):
        alpha = mu[j]*phi['M'][j]
        beta = (1-mu[j])*phi['M'][j]
        theta[:,j] = ss.beta.rvs(alpha, beta, size=N)
        r[:,j] = ss.binom.rvs(n[:,j], theta[:,j])
    return r, theta, mu

```

```

## compute sufficient statistics
def EqlogTheta(delta):
    if delta[0] < np.finfo(float).eps:
        delta[0] += np.finfo(float).eps
    return psi(delta[0]) - psi(np.sum(delta))

def Eqlog1_Theta(delta):
    if delta[1] < np.finfo(float).eps:
        delta[1]+=np.finfo(float).eps
    return psi(delta[1]) - psi(np.sum(delta))

def EqMu(gam):
    return gam[0] / (np.sum(gam)) # eps?


def EqlogMu(gam):
    if gam[0] < np.finfo(float).eps:
        gam[0] += np.finfo(float).eps
    return psi(gam[0]) - psi(np.sum(gam))

def Eqlog1_Mu(gam):
    return psi(gam[1]) - psi(np.sum(gam))

def EqlogGamma(gam, M):
    # Expectation of Beta function coefficient
    logGamma = integrate.quad(kernel, 1e-3, 1-1e-3, \
                    args=(gam, M), full_output=1)
    return logGamma[0]

def kernel(mu, gam, M):
    return -ss.beta.pdf(mu, gam[0], gam[1])*betaln(mu*M, (1-mu)*M)


## compute entropy
def BetaEntropy(x):
    # To compute EqlogQmu and EqlogQtheta
    return betaln(x[0], x[1]) - (x[0]-1) * psi(x[0]) - (x[1] - 1)\
           * psi(x[1]) + (x[0] + x[1] -2) * psi(x[0] + x[1])

## compute ELBO
def ELBO(r, n, M, mu0, M0, delta, gam):
    if np.ndim(r) == 1:
        N, J = (1, np.shape(r)[0])
    elif np.ndim(r) == 2:
        N, J = np.shape(r)

    # Compute the expectations
    try:
        Mu = np.array([EqMu(gam[j,:]) for j in xrange(J)])
    except TypeError:
        pdb.set_trace()
    # Mu = np.array([EqMu(gam[j,:]) for j in xrange(J)])
    logMu = np.array([EqlogMu(gam[j,:]) for j in xrange(J)])
    log1_Mu = np.array([Eqlog1_Mu(gam[j,:]) for j in xrange(J)])

    logTheta = np.zeros((N,J))
    log1_Theta = np.zeros((N,J))

    for j in xrange(J):
        for i in xrange(N):
            logTheta[i,j] = EqlogTheta(delta[i,j,:])
            log1_Theta[i,j] = Eqlog1_Theta(delta[i,j,:])

    # Eq[log p(r|theta, n)]
    EqlogPr = 0.0
    for j in xrange(J):
        for i in xrange(N):
            EqlogPr += -betaln(r[i,j] + 1, n[i,j] - r[i,j] +1) \
                       -np.log(n[i,j]+1)
            EqlogPr += r[i,j]*logTheta[i,j] + (n[i,j] - r[i,j]) \
                       * log1_Theta[i,j]

    # Eq[log p(theta|mu, M)]
    EqlogPtheta = 0.0
    for j in xrange(J):

        EqlogPtheta += N*EqlogGamma(gam[j,:], M[j])
        for i in xrange(N):
            EqlogPtheta += (M[j]* Mu[j]- 1)*logTheta[i,j] +\
            (M[j]*(1 - Mu[j]) - 1)*log1_Theta[i,j]
    # Eq[log p(mu; mu0, M0)]
    EqlogPmu = -J * betaln(mu0*M0, (1-mu0)*M0)
    for j in xrange(J):
        EqlogPmu += (M0*mu0-1)*logMu[j] + (M0*(1-mu0)-1)*log1_Mu[j]

    EqlogQtheta = 0.0
    for j in xrange(J):
        for i in xrange(N):
            EqlogQtheta -= BetaEntropy(delta[i,j,:])
```

```python

    EqlogQmu = 0.0
    for j in xrange(J):
        EqlogQmu -= BetaEntropy(gam[j,:])

    return EqlogPr + EqlogPtheta + EqlogPmu \
           - EqlogQtheta - EqlogQmu


def ELBO_delta_ij(r, n, M, delta, gam):
    ## partial ELBO from replicate i position j
    ## ELBO used to optimize delta
    ## Commented out all items that don't depend on delta

    Mu = EqMu(gam)
    logTheta = EqlogTheta(delta)
    log1_Theta = Eqlog1_Theta(delta)

    EqlogPr = r*logTheta + (n - r)*log1_Theta

    EqlogPtheta = (M*Mu - 1)*logTheta + (M*(1-Mu)-1)*log1_Theta

    EqlogQtheta = -BetaEntropy(delta)

    return EqlogPr + EqlogPtheta - EqlogQtheta
    # return EqlogPr + EqlogPtheta

def neg_ELBO_delta_ij(logdelta, gam, r, n, M):
    return -ELBO_delta_ij(r, n, M, np.exp(logdelta), gam)

def opt_delta_ij(args):
    # pdb.set_trace()
    r, n, M, delta, gam = args
    # pdb.set_trace()
    # limit delta to [0.001, 1000], np.log(delta) is [-6.9, 6.9]
    #bnds = [[-7, 7]]*2
    # limit delta to [0.0001, 10000], np.log(delta) is [-10, 10]
    bnds = [[-10, 10]]*2
    args=(gam, r, n, M)

    # logging.debug(bnds)
    # logging.debug(np.log(delta))
    logdelta = opt_par(neg_ELBO_delta_ij, np.log(delta), \
                       args, bnds, 'delta')
    delta = np.exp(logdelta)

    return delta

def opt_delta(r, n, M, delta, gam, pool = None):
    logging.debug("Optimizing delta")

    if np.ndim(r) == 1: N, J = (1, np.shape(r)[0])
    elif np.ndim(r) == 2: N, J = np.shape(r)

    st = time.time()
    if pool is not None:
        for i in xrange(N):
            args = zip (r[i,:], n[i,:], M, delta[i,:], gam)
            temp  = pool.map(opt_delta_ij, args)
            delta[i,:] = np.array(temp)
    else:
        logging.debug('Optimizing delta in single thread')
        for i in xrange(N):
            for j in xrange(J):
                logging.debug('Optimizing position %d of %d \
                 and replicate %d of %d' % (j,J,i,N))
                args = (r[i,j],n[i,j],M[j],delta[i,j,:],gam[j,:])
                delta[i,j,:] = opt_delta_ij(args)

    logging.debug('Delta update elapsed time is %0.3f sec for %d\
            samples %d replicates.' % (time.time() - st, J, N))
    return delta

def ELBO_gam_j( M, mu0, M0, delta, gam):
    ## partial ELBO depending on gam from each position j
    ## ELBO used to gam

    if np.ndim(delta) == 1: N = 1
    elif np.ndim(delta) == 2: N= np.shape(delta)[0]

    Mu = EqMu(gam)
    logMu = EqlogMu(gam)
    log1_Mu = Eqlog1_Mu(gam)

    logTheta = np.zeros((N,1))
    log1_Theta = np.zeros((N,1))

    for i in xrange(N):
        logTheta[i] = EqlogTheta(delta[i,:])
        log1_Theta[i] = Eqlog1_Theta(delta[i,:])

    EqlogPtheta = N*EqlogGamma(gam,M)
    for i in xrange(N):
        EqlogPtheta += (M*Mu-1) * logTheta[i] + (M*(1-Mu)-1) \
                       *log1_Theta[i] ## I had a typo here (M->Mu)

    EqlogPmu= -betaln(mu0*M0, (1-mu0)*M0)+ (M0*mu0-1)*logMu \
              + (M0*(1-mu0)-1)*log1_Mu

    EqlogQmu = -BetaEntropy(gam)

    return  EqlogPtheta + EqlogPmu - EqlogQmu
    # return  EqlogPtheta + EqlogPmu

def neg_ELBO_gam_j(loggam, delta, M, mu0, M0):
    return -ELBO_gam_j(M, mu0, M0, delta, np.exp(loggam))

def opt_gam_j(args):
    M, mu0, M0, delta, gam = args
    # pdb.set_trace()
    # limit gam to [0.001, 1000], np.log(gam) is [-6.9, 6.9]
    #bnds = [[-7, 7]]*2
    # limit gam to [0.0001, 10000], np.log(gam) is [-10, 10]
    bnds = [[-10, 10]]*2
    args = (delta, M, mu0, M0)

    # def opt_par(func, x, args, bnds, parlabel):
    loggam = opt_par(neg_ELBO_gam_j,np.log(gam),args,bnds,'gamma')
    gam = np.exp(loggam)
    # logging.debug(bnds)
    # logging.debug(loggam)
    return gam

def opt_gam(M, mu0, M0, delta, gam, pool = None):
    logging.debug("Optimizing gam")

    if np.ndim(gam) == 1: J=1
    elif np.ndim(gam) == 2: J=np.shape(gam)[0]

    st = time.time()

    if pool is not None:
        args = zip( M, repeat(mu0,J), repeat(M0,J), \
                 np.transpose(delta,axes=(1,0,2)),gam)
        gam = pool.map(opt_gam_j, args)
        gam = np.array(gam)

    else:
        for j in xrange(J):
            # pdb.set_trace()
            logging.debug("Optimizing gamma %d of %d" % (j, J))
            args = ( M[j], mu0, M0, delta[:,j,:], gam[j] )
            gam[j] = opt_gam_j(args)

    logging.debug('Gamma update elapsed time is %0.3f sec \
                  for %d samples.' % (time.time() - st, J))
    return gam

def ELBO_0(mu0, M0, gam):
    ## Items in ELBO depends on mu0 and M0
    ## FOr optimization of mu0 and M0

    J = gam.shape[0]

    logMu = np.array([EqlogMu(gam[j,:]) for j in xrange(J)])

    log1_Mu = np.array([Eqlog1_Mu(gam[j,:]) for j in xrange(J)])

    EqlogPmu = -J * betaln(mu0*M0, (1-mu0)*M0)
    for j in xrange(J):
        EqlogPmu += (M0*mu0-1)*logMu[j] + (M0*(1-mu0)-1)*log1_Mu[j]

    return EqlogPmu

def neg_ELBO_mu0(mu0, M0, gam):
    return -ELBO_0(mu0, M0, gam)

def opt_mu0(mu0, M0, gam):
    logging.debug("Optimizing mu0")
    #bnds = np.array([[0.01,0.99]])
    bnds = np.array([[0.0,1.0]])
    args=(M0, gam)
    mu0 = opt_par(neg_ELBO_mu0, mu0, args, bnds, 'mu0' )

    return mu0

def neg_ELBO_M0(logM0, mu0, gam):
    return -ELBO_0(mu0, np.exp(logM0), gam)

def opt_M0(mu0, M0, gam):
    logging.debug("Optimizing M0")

    #bnds = np.array([[-7,7]])
    bnds = np.array([[-10,10]])

    args = (mu0, gam)
    logM0 = opt_par(neg_ELBO_M0, np.log(M0), args, bnds, 'M0' )
    M0 = np.exp(logM0)
    return M0

def ELBO_M_j(M, delta, gam):
    ## partial ELBO depending on M from each position j
    ## ELBO used to optimize M

    if np.ndim(delta) == 1: N = 1
    elif np.ndim(delta) == 2: N= np.shape(delta)[0]

    Mu = EqMu(gam)

    logTheta = np.zeros((N,1))
    log1_Theta = np.zeros((N,1))

    for i in xrange(N):
        logTheta[i] = EqlogTheta(delta[i,:])
        log1_Theta[i] = Eqlog1_Theta(delta[i,:])

    EqlogPtheta = N*EqlogGamma(gam,M)
    for i in xrange(N):
        EqlogPtheta += (M*Mu-1) * logTheta[i] \
                       + (M*(1-Mu)-1)*log1_Theta[i]

    return  EqlogPtheta

def neg_ELBO_M_j(logM, delta, gam):
    return -ELBO_M_j(np.exp(logM), delta, gam)

def opt_M_j(args):

    (M, delta, gam) = args
    #bnds = np.array([[-1, 11]]) # limit delta to [0.0001, 10000]
    #limit delta to [0.0001, 10000], np.log(delta) is [-9.21, 9.21]
    bnds = np.array([[-10, 10]])
    M = np.array(M)
    args = (delta, gam)

    logM = opt_par(neg_ELBO_M_j, np.log(M), args, bnds, 'M')
    M = np.exp(logM)

    return M

def opt_M(M, delta, gam, pool = None):
    # M = opt_M(M, delta, gam, pool = pool)
    logging.debug("Optimizing M")

    J= np.shape(M)[0]
    # pdb.set_trace()
    # M = np.array(M)

    if pool is not None:
        args = zip(M, np.transpose(delta, axes =(1,0,2)), gam)
        M = pool.map(opt_M_j, args)

    else:
        for j in xrange(J):
            args = (M[j],delta[:,j,:],gam[j,:])
            M[j] = opt_M_j(args)

    return M

def opt_par(func, x, args, bnds, parlabel):

    # often the fastest method to minimize functions of many
    # variables uses the Newton-Conjugate Gradient algorithm.
    # A function which computes the Hessian must be provided.

    # res = so.minimize(func, x,
    #     args=args, bounds=bnds,
    #     method='Newton-CG')

    # logging.debug("Inside of optimize function. got res")
    # Nelder-Mead is the simplest way to minimize a
    # well-behaved function. Good for simple minimization
    # problems. Does not use any gradient evaluations,
    # might take longer.
    # There is no bounds for Nelder-Mead method
    # if res.success == False:
    #     logging.debug(2)
    #     res = so.minimize(func, x,
    #         args=args, bounds=bnds, method='Nelder-Mead')
    # pdb.set_trace()

    '''res = so.minimize(func, x,
        args=args, bounds=bnds,
        method='L-BFGS-B' ) # limited memory BFGS method'''

    #if res.success == False:
    #    logging.debug(3)
    res = so.minimize(func, x,
        args=args, bounds=bnds, method='SLSQP') \
        # Sequential Least SQuares Programming to minimize
        # a function of several variables with any combination of
        # bounds, equality and inequality constraints

    if res.success == False and parlabel != 'M':
        logging.debug(1)
        res = so.minimize(func, x,
        args=args, bounds=bnds, method='TNC')
        # truncated Newton algorithm to minimize a function
        # with variables subject to bounds.

    if res.success == False:
        logging.debug(2)
        res = so.minimize(func, x, args=args, bounds=bnds, \
              method='L-BFGS-B' ) # limited memory BFGS method

    # use the gradient of the objective function,
    # which can be given by the user.
    # quasi-Newton method of Broyden, et al.
    if res.success == False:
        logging.debug(3)
        pdb.set_trace()
        res = so.minimize(func, x, bounds=bnds,\
            args=args, method='BFGS')

    if res.success == False or np.any ( np.isnan(res.x) ) \
                     or np.any(np.isinf(res.x)):
        logging.warning("Could not optimize %s or %s is NaN."\
                        %(parlabel, parlabel))
        x = np.random.uniform(low=np.amin(bnds), \
            high=np.amax(bnds), size = np.shape(x))
        return x

    return res.x

def beta_mean(p):
    return p[0]*1.0/np.sum(p)


def ELBO_opt(r,n,phi=None,q=None,seed=None,pool=None,vaf=None):

    if pool is not None:
        pool = mp.Pool(processes=pool)
    # t = str(datetime.now)
    f = open('ELBO%s.txt' % str(vaf).replace(".", "_", 1),'w')
    t = time.time()

    if np.ndim(r) == 1: N, J = (1, np.shape(r)[0])
    elif np.ndim(r) == 2: N, J = np.shape(r)
    elif np.ndim(r) == 3:
        r = np.sum(r, 2)
        (N, J) = r.shape# sum over non-reference bases
    # r = r.T
    # n = n.T

    if seed is not None: np.random.seed(seed = seed)

    h5file = tempfile.NamedTemporaryFile(suffix='.hdf5')
    logging.info('Storing model updates in %s' % h5file.name)
    #temp = "tmp.hdf5"
    #logging.info('Storing model updates in %s' % temp)

    ## Define optimization stopping criterion
    MAXITER = 80
    ELBOTOLPCT = 0.001 *100
    MAXVARITER = 80
    NORMTOL = 0.1

    ## Initialize model parameters
    if phi is None:
        phi, mu, theta = estimate_mom(r, n)
    else:
        _, mu, theta = estimate_mom(r, n)
    mu0 = phi['mu0']
    M0 = phi['M0']
    M = phi['M']

    ## Initialize the variational parameters
    if q is None:
        #delta=np.random.uniform(low=0.1,high=100,size=(N,J,2))
        #gam=np.random.uniform(low=0.1,high=100,size=(J,2))
        delta=np.random.uniform(low=0.0001,high=10000,size=(N,J,2))
        gam=np.random.uniform(low=0.0001,high=10000,size=(J,2))

    else:
        delta = q['delta']
        gam = q['gam']

    phi = {'mu0':mu0, 'M0':M0, 'M':M}
    q = {'delta':delta, 'gam':gam}
    #save_model('initial_value.hdf5', r, n, phi, q)

    '''# Look at the initial random value of \mu_j
    logging.info("Initial gam: %s" % gam[344,:])
    logging.info("Initial $\mu$: %s" % beta_mean(gam[344,:]))'''
    ## Initialize ELBO

    elbo = [ELBO(r, n, M, mu0, M0, delta, gam)]
    logging.info("Initial ELBO: %0.2f" % elbo[-1])

    print("M-iter\tE-iter\tELBO\tInc_Per\tdelta-deltaprev \
         \tgam-gamprev\tt-gam\tt-delta\tt-mu0\tt-M0\tt-M",file=f)

    print("%d\t%d\t%0.2f\t%0.3f%%\t\t\t\t\t\t\t" \
         %(0,0,elbo[-1],0),file=f)

    # print("Initial \tELBO: \t%0.2f" % elbo[-1], file = f)


    ## Optimization
    moditer = 0
    delta_elbo_pct = np.inf

    while moditer<MAXITER and np.abs(delta_elbo_pct)>ELBOTOLPCT:
        # E-step: Update the variational distribution
        variter = 0
        var_elbo = [ elbo[-1] ]
        (norm_delta_delta, norm_delta_gam) = (np.inf, np.inf)
        delta_varelbo_pct = np.inf
        logging.info("E-step")
        while variter < MAXVARITER \
            and delta_varelbo_pct > ELBOTOLPCT \
            and (norm_delta_delta > NORMTOL \
            or norm_delta_gam > NORMTOL):

            #Store the previous parameter values
            (delta_prev, gam_prev) = (np.copy(delta), np.copy(gam))

            #Update the variational distribution
            # pdb.set_trace()
            t0=time.time()
            # mu~Beta(gam)
            gam = opt_gam( M, mu0, M0, delta, gam, pool = pool)
            t1=time.time()
            # theta~Beta(delta)
            delta = opt_delta(r, n, M, delta, gam, pool = pool)
            t2=time.time()

            #Test for convergence
            var_elbo.append(ELBO(r, n, M, mu0, M0, delta, gam))
            delta_varelbo_pct = 100.0*(var_elbo[-1] - \
                                var_elbo[-2])/abs(var_elbo[-2])
            logging.info("****Variational Iteration %d of %d****"\
                         % (variter+1, MAXVARITER))
            logging.info("ELBO: %0.2f; Percent Change: %0.3f%%"\
                         % (var_elbo[-1], delta_varelbo_pct))


            norm_delta_delta = linalg.norm(delta - delta_prev)
            norm_delta_gam = linalg.norm(gam - gam_prev)
            logging.debug("||delta - delta_prev|| = %0.2f;\
                            ||gam - gam_prev|| = %0.2f"
                    % (norm_delta_delta, norm_delta_gam))

            print("%d\t%d\t%0.2f\t%0.3f%%\t%0.2f\t%0.2f\t%0.2f\t\
                  %0.2f\t\t\t" %(moditer, variter+1, var_elbo[-1],\
                  delta_varelbo_pct, norm_delta_delta,\
                  norm_delta_gam, t1-t0,t2-t1), file=f)
            variter += 1

        logging.info("M-step")
        # M-step: Update model parameters
        t0=time.time()
        mu0 = opt_mu0(mu0, M0, gam)
        t1=time.time()
        M0 = opt_M0(mu0, M0, gam)
        t2=time.time()
        M = opt_M(M, delta, gam, pool = pool)
        t3=time.time()

        elbo.append(ELBO(r, n, M, mu0, M0, delta, gam))
        delta_elbo_pct = 100*(elbo[-1] - elbo[-2])/abs(elbo[-2])
        moditer += 1


        # ibic

        # Display results for debugging
        logging.info("----------Iteration %d of %d.----------" \
                     % (moditer, MAXITER))
        logging.info("ELBO: %0.2f; Percent Change: %0.3f%%" \
                    % (elbo[-1], delta_elbo_pct))

        print("%d\t%d\t%0.2f\t%0.3f%%\t\t\t\t\t%0.2f\t \
        %0.2f\t%0.2f" %(moditer,0, elbo[-1],delta_elbo_pct, \
        t1-t0,t2-t1,t3-t2), file=f)

        logging.info("M0 = %0.2e" % M0)
        logging.info("mu0 = %0.2f" % mu0)

        '''# Store the model for viewing
        phi = {'mu0':mu0, 'M0':M0, 'M':M}
        q = {'delta':delta, 'gam':gam}
        save_model(h5file.name, r, n, phi, q)'''

    print("Total time is %0.3f seconds." %(time.time()-t), file=f)

    f.close()
    return(phi, q)

def estimate_mom(r, n):
    """ Return model parameter estimates using method-of-moments.
    """
    # make sure this is non-truncating division
    theta = r/(n + np.finfo(np.float).eps)
    if np.ndim(r) == 1: mu = theta
    elif np.ndim(r) > 1: mu = np.mean(theta, 0)

    mu0 = np.mean(mu)
    M0 = (mu0*(1-mu0))/(np.var(mu) + np.finfo(np.float).eps) \
         + np.finfo(np.float).eps

    # Estimate M.
    # If there is only one replicate, set M as 10 times of M0.
    # If there is multiple replicates, set M according to
    # the moments of beta distribution
```

```python

    if np.shape(theta)[0] is 1:
        M = 10*M0*np.ones_like(mu)
    else:
        M = (mu*(1-mu))/(np.var(theta, 0) + np.finfo(np.float).eps)

    J = len(M)
    for i in xrange(J):
        if M[i] < 1:
            M[i] = 1

    phi = {'mu0':mu0, 'M0':M0, 'M':M}
    return phi, mu, theta

def save_model(h5Filename, r, n, phi, q, loc=None, refb=None):

    f = h5py.File(h5Filename, 'w')

    f.create_dataset('r', data=r)
    f.create_dataset('n', data=n)

    f.create_group('phi')
    f['phi'].create_dataset('mu0', data=phi['mu0'])
    f['phi'].create_dataset('M0', data=phi['M0'])
    f['phi'].create_dataset('M', data=phi['M'])

    f.create_group('q')
    f['q'].create_dataset('delta', data=q['delta'])
    f['q'].create_dataset('gam', data=q['gam'])

    # Save the reference data
    if loc is not None:
        f.create_dataset('loc', data=loc,
        chunks=True, fletcher32=True, compression='gzip')
    if refb is not None:
        f.create_dataset('refb', data=refb)

    f.close()

def load_model(h5Filename):

    f = h5py.File(h5Filename, 'r')

    out = []

    # pdb.set_trace()
    r = f['r'][...]
    out.append(r)

    n = f['n'][...]
    out.append(n)

    phi = {}
    phi['mu0'] = f['phi/mu0'][...]
    phi['M0'] = f['phi/M0'][...]
    phi['M'] = f['phi/M'][...]
    out.append(phi)

    q = {}
    q['delta'] = f['q/delta'][...]
    q['gam'] = f['q/gam'][...]
    out.append(q)

    if u"loc" in f.keys():
        loc = f['loc'][...]
        out.append(loc)

    if u"refb" in f.keys():
        refb = f['refb'][...]
        out.append(refb)

    f.close()
    # pdb.set_trace()

    return tuple(out)

def load_depth(dcFileNameList):
    """ Return (r, n, location, reference base) for a list of
        depth charts. The variable r is the error read depth
        and n is the total read depth.
    """
    r=[]; n=[]
    acgt = {'A':0, 'C':1, 'G':2, 'T':3}

    loc = []
    refb = {}
    cd = []

    # pdb.set_trace()

    for dcFileName in dcFileNameList:
        with open(dcFileName, 'r') as dcFile:
            header = dcFile.readline().strip()
            dc = dcFile.readlines()
            dc = [x.strip().split("\t") for x in dc]
            loc1 = [x[1]+':'+str(x[2]).strip('\000') for x in dc \
                   if x[4] in acgt.keys()]
            loc.append( loc1 )
            refb1 = dict(zip(loc1, [x[4] for x in dc \
                    if x[4] in acgt.keys()]))
            refb.update(refb1)
            cd.append(dict(zip(loc1, [map(int, x[5:9]) for x in dc\
                     if x[4] in acgt.keys()])) )

    loc = list(reduce(set.intersection, map(set, loc)))

    def stringSplitByNumbers(x):
        r = re.compile('(\d+)')
        l = r.split(x)
        return [int(y) if y.isdigit() else y for y in l]

    loc = sorted(loc,key = stringSplitByNumbers)
    logging.debug(loc)
    refb = [refb[k] for k in loc]

    J = len(loc)
    N = len(dcFileNameList)
    for i in xrange(0, N):
        logging.debug("Processing %s" % dcFileNameList[i])
        c = np.array( [cd[i][k] for k in loc] )
        n1 = np.sum(c, 1)
        #r1 = np.zeros(J)
        refIdx=np.zeros(J)

        for j in xrange(0,J):
            #r1[j] = n1[j] - c[j, acgt[refb[j]]]
            refIdx[j] = 4*j+acgt[refb[j]]
        c = np.delete(c, refIdx, None)
        c = np.reshape(c, (J, 3) )
        #r.append(r1)
        n.append(n1)
        r.append(c)
    r = np.array(r)
    n = np.array(n)

```

```python
    return (r, n, loc, refb)

if __name__ == "__main__":
    main()
```

C.2 GOP code

C.2.1 Main function

```python
__author__ = 'Fan Zhang'
# GOP_main.py

import numpy as np
from multiprocessing import Pool
import h5py as h5
from tree import Tree

from time import time
import time
import parallel_cell_enumeration as p_cell
import parallel_masterprob as p_ma
import pre_process as pre
import subprob as sub


if __name__ == "__main__":

    # Set a random seed
    np.random.seed(19860522)

    # Define global optimization parameters
    #gb.setParam(gb.GRB.Param.Threads, NUM_THREADS)

    NUM_CORES = 10

    #Parallel
    pool = Pool(processes=NUM_CORES)
    pool = None

    ################# Define the problem data. #############
    '''M = 20
    K = 2
    N = 10
    # make theta_star
    x_star = np.random.normal(size=(M, K))
    alpha = np.random.uniform(0,1,K)
    theta = np.random.dirichlet(alpha, N)
    theta_star = np.transpose(theta)
    y = np.dot(x_star, theta_star)'''

    # illustrate data
    x_star = np.array([(0, -1)])
    theta_star = np.array([ (1, 0, 0.5),
                            (0, 1, 0.5)])
    y = np.dot(x_star, theta_star)

    # Define the sparsity of x
    # cons: used in the solve_master_s to constraint the sum of
    # all the elements of |x_star|.
    # It would be easier to set cons three times of the sum of
    # all the absoute values for x.
    cons = np.sum(abs(x_star))

    (M, N)  = np.shape(y)
    (M, K) = np.shape(x_star)

    #print M, K, N
    print "Optimal ||y-x*theta||_2^2 = %0.2f" \
          % np.linalg.norm(y-np.dot(x_star, theta_star),2)**2

    # Define the problem parameters
    MAXITER = 2000
    e = 0.01

    # Initialize the problem parameters
    x_stor = []
    SUBD = np.inf
    MLBD = -np.inf

    # Initialize the problem decision variables
    #xBar = x_star + 0.1*np.random.normal(size=(M,K))

    # Randomly generate x between 0 and 1.
    xBar =  np.random.random_sample((M,K))
    print "Initial x is: %s" %xBar

    theta_U = np.zeros((K,N))
    theta_L = np.zeros((K,N))
    for i in xrange(N):
        for j in xrange(K):
            theta_U[j][i] = 1.0
            theta_L[j][i] = 0.0

    # record the optimal value
    thetaBar = []
```

```python
    lamBar = []
    muBar = []
    x_all = [xBar]
    MLBD_Bar = []
    SUBD_Bar = []
    # record all the MLBD of the generated nodes
    MLBD_all = []
    # record the chosen nodes with the lowest MLBD per iteration
    nodes_all = []
    # record the time for each iteration
    time_iteration = []

    print "Start optimizing..."

    index=0
    global num_node
    num_node = 0
    current_node = 0

    #Claim a tree
    tree = Tree()
    node = tree.add_node(current_node, theta_L, theta_U)
    num_node = num_node + 1

    start_all = time.clock()
    print x_all[-1]

    while index < MAXITER:
        start = time.clock()
        print "-------------iteration %d------------" % index

        ''' Solve the primal problem '''

        objOpt,thetaOpt,lamOpt,muOpt=sub.solve_subproblem(y, xBar)

        thetaBar.append(thetaOpt)
        lamBar.append(lamOpt)
        muBar.append(muOpt)

        SUBD = np.amin([objOpt, SUBD])

        print "THETA"
        print thetaOpt
        print "X"
        print xBar

        ''' Preprocessing:
        1. remove duplicate hyperplanes;
        2. remove all 0 coefficients hyperplanes.'''

        g_flag, replicated_marker, coefficients= pre.pre_process\
        (xBar, thetaOpt, lamOpt, muOpt, y)

        # print the flag and deplicate markers
        # print "g_flag", g_flag
        # print "replicated_marker", replicated_marker
        # print len(coefficients)
        # for co_index in xrange(len(coefficients)):
        # print coefficients[co_index]

        # Get all the unique hyperplanes and save the coefficients.
        linker, unique_coefficients = pre.unique_coeff(g_flag,\
            replicated_marker,  coefficients,  M,  K,  N)

        # Set a threshold as the distance used in cell enumeration
        distance = [np.spacing(1)]
        for i in xrange(len(unique_coefficients)):
            sum = 0.0
            sum += unique_coefficients[i][-1]
            for j in xrange(M*K):
                sum += xBar[j/K][j%K]* unique_coefficients[i][j]
            distance.append(np.fabs(sum))

        # Take the maximum of the 'distances' from the common point
        # to all the hyperplanes.
        # Make the threshold greater than or equal to np.spacing(1).
        threshold = max(distance)

        ''' Cell enumeration: Get the unique regions which are
        represented by thetaB_list (using parallel)'''

        pre_thetaB_list = p_cell.parallel_cell_numeration \
        (unique_coefficients, len(unique_coefficients), \
        M*K, threshold, pool)

        thetaB_list = pre.extend_back(pre_thetaB_list, linker, \
           g_flag, replicated_marker, K, N)

        print "\nthe length of thetaB_list is:",  len(thetaB_list)

        ''' Solve the relaxed dual problems
        defined by the thetaB_list '''

        x_stor, Q_stor, next_node, num_node, MLBD_stor = \
        p_ma.solve_master(tree, num_node, current_node, g_flag,\
        thetaB_list, SUBD,  coefficients,  xBar, thetaOpt, \
        lamOpt, muOpt, y,  cons, i, pool)

        MLBD_all.extend(MLBD_stor)

        # Set the master problem lower bound and the next x value
        current_node = tree.search_leaves(0, 0, np.inf)

        nodes_all.append(current_node)
        xBar = tree.nodes[current_node].xOpt
        MLBD = tree.nodes[current_node].MLBD
        tree.nodes[current_node].MLBD = np.inf

        # Calculate the time for each iteration
        end = time.clock()

        # record the x_all, MLBD_Bar and SUBD_Bar of
            # the chosen nodes with the lowest MLBD
        x_all.append(xBar)
        MLBD_Bar.append(MLBD)
        SUBD_Bar.append(SUBD)

        time_iter = '%0.2f'  %(end - start)
        print ('\nTime used for this iteration:%0.2f'%(end-start))
        time_iteration.append(time_iter)

        with h5.File('test.hdf5','w') as f:
            f.create_dataset('MLBD', data=MLBD_Bar)
            f.create_dataset('SUBD', data=SUBD_Bar)
            f.create_dataset('xOpt', data=x_all)
            f.create_dataset('thetaOpt', data=thetaBar)
            f.create_dataset('lamOpt', data=lamBar)
            f.create_dataset('muOpt', data=muBar)
            f.create_dataset('MLBD_all_nodes', data=MLBD_all)
            f.create_dataset('selected_nodes', data=nodes_all)
            f.create_dataset('time', data=time_iteration)

        print('Current bounds: [%0.2f, %0.2f]' % (MLBD, SUBD))

        # Try another convergence:
```

```
        #close = np.fabs(np.dot(x_all[-2], thetaOpt) - y) < 0.05

        # Set convergence of upper and lower bounds
        if SUBD - MLBD <= e:
            print "\n==========Optimal x*theta============"
            print np.dot(x_all[-2], thetaOpt)
            print "===============Exact y================="
            print y
            print "===============Optimal x==============="
            print x_all[-2]
            print "==================Exact x==============="
            print x_star
            print "==============Optimal theta============"
            print thetaOpt
            print "================Exact theta============"
            print theta_star

            index = MAXITER
            end_all = time.clock()
            print ('\nAll the iterations cost: %0.2f'\
                  %(end_all - start_all))
        index += 1
```

C.2.2 Primal problem

```python
__author__ = 'Fan Zhang'
# subprob.py

import numpy as np
import gurobipy as gb
import itertools as it
import matplotlib.pyplot as plt
from multiprocessing import Pool

import h5py as h5

from time import time

def solve_subproblem(y, x):
    '''Solve the GOP subproblem (dual) for a fixed value of x '''

    (M, N) = np.shape(y)
    K = np.shape(x)[1]

    # Create a new model
    m = gb.Model("sub")
    m.setParam('OutputFlag', False)

    # Create variables
    theta = [[0 for i in xrange(N)] for k in xrange(K)]
    for (k,i) in it.product(range(K), range(N)):
        theta[k][i] = m.addVar(lb = -gb.GRB.INFINITY,
            ub = gb.GRB.INFINITY, vtype = gb.GRB.CONTINUOUS,
            name = "theta_%d_%d" % (k,i) )

    #Integrate new variables
    m.update()

    #Construct the objective min_theta sum_i (y_i - x*theta_i)^2
    X2 = np.dot(x.T, x)

    obj = gb.QuadExpr()
    for i in xrange(N):
        yx = np.dot(y[:,i], x)

        # convert numpy.float64(0) to a native Python type.
        obj.addConstant(np.asscalar(np.dot(y[:,i].T, y[:,i])))

        for k1 in xrange(K):
                obj.addTerms(-2*yx[k1], theta[k1][i])
                for k2 in xrange(K):
                     obj.addTerms(X2[k1][k2],theta[k1][i],theta[k2][i])

    m.setObjective(obj, gb.GRB.MINIMIZE)

    # Add constraint: sum_k theta_{ki} = 1 for all i
    for i in xrange(N):
        m.addConstr(gb.quicksum(theta[k][i] for k in xrange(K))==1)

    # Add constraint: theta_{ki} >= 0 for all (k,i)
    for (k,i) in it.product(range(K), range(N)):
        m.addConstr(theta[k][i] >= 0)

    # Optimize the model
    m.optimize()
    assert(m.getAttr('Status') == 2)

    # Package the optimizing value of theta
    thetaOpt = np.empty(np.shape(theta))
    for (k,i) in it.product(range(K), range(N)):
        thetaOpt[k,i] = theta[k][i].x

    # Package the Lagrange dual variables values
    lamOpt = np.empty(N)
    muOpt = np.empty(shape=(K,N))
    constrs = m.getConstrs()

    for i in xrange(N):
        lamOpt[i] = constrs[i].getAttr("pi")

        for (k,i) in it.product(range(K), range(N)):
            muOpt[k,i] = constrs[N+k*N+i].getAttr("pi")

    # Return the objective(Upper bound), theta, lambda, mu
    return m.objVal, thetaOpt, lamOpt, muOpt



if __name__ == '__main__':
    pass
```

C.2.3 Preprecessing

```python
__author__ = 'Fan Zhang'
# pre_process.py

import numpy as np
import gurobipy as gb
import itertools as it
import sympy

def get_the_coefficient(g_expr,  M,  K,  m):
    '''  Get the coefficients of one qualifying constraint.
    More efficiently! We observe that VarName is named by "x_i_j".
    Therefore, we can extract i and j from the name.'''

    num = g_expr.size()
    x = [[m.getVarByName("x_%d_%d" % (j,k)) for k in xrange(K)] \
          for j in xrange(M)]

    coeff = np.zeros((M,  K))
    coeff_constant = g_expr.getConstant()
    for index in xrange(num):
        var_name = g_expr.getVar(index).VarName.encode('ascii', \
                    'ignore')
        indexes = [ind for ind, ltr in enumerate(var_name) \
                    if ltr == '_']
        i = int(var_name[indexes[0] + 1 : indexes[1]])
        j = int(var_name[indexes[1] + 1 : ])
        coeff[i][j] += g_expr.getCoeff(index)

    coefficient = []
    for i in xrange(M):
        for j in xrange(K): #set the precision
            coefficient.append(float(coeff[i][j]))
    coefficient.append(float(coeff_constant))
    return coefficient


def normalize_coefficient(coef):
    '''Normalization is not used in the current version.'''
    '''Normalize the coefficients of Ab by the largest one of A.'''

    #get the largest magnitude of A
    max_coef = max(coef[:-1])
    min_coef = min(coef[:-1])
    normalization = 0.0
    if np.fabs(max_coef) > np.fabs(min_coef):
        normalization = np.fabs(max_coef)
    else:
        normalization = np.fabs(min_coef)
    assert(normalization != 0.0), \
          "normalization should not equal to zero"
    #divide the largest magnitude
    normalized_coef = []
    for i in xrange(len(coef)):
        normalized_coef.append(float(coef[i]/normalization))
    return normalized_coef


def check_line(coeff,  M,  K):
    #Since we used the truncation,
    #we compare the coeff with 0.0001 directly.
    for index in xrange(M*K + 1):
        if np.fabs(coeff[index]) >= 0.0001:
            return False
    return True


def replicate_line_with_threshold(g_expr,g_expr_c,M,K,threshold):
    '''Compare whether two lines are the same lines.'''
    coeffi_1 = np.zeros((M,  K))
    coeffi_2 = np.zeros((M,  K))
    coeff1_constant = g_expr[-1]
    coeff2_constant = g_expr_c[-1]
    #calculate the sum of all coefficients
    sum_coeff1 = coeff1_constant
    sum_coeff2 = coeff2_constant
    for i in xrange(M):
        for j in xrange(K):
            coeffi_1[i][j] = g_expr[i*K + j]
            coeffi_2[i][j] = g_expr_c[i*K + j]
            sum_coeff1 += coeffi_1[i][j]
            sum_coeff2 += coeffi_2[i][j]

    #check constant
    if np.fabs(sum_coeff1 * coeff2_constant \
      - sum_coeff2 * coeff1_constant) > threshold:
        return 0
    #check all other coefficients
    for i in xrange(M):
        for j in xrange(K):
                if np.fabs(sum_coeff1 * coeffi_2[i][j] \
                    - sum_coeff2 * coeffi_1[i][j]) > threshold:

                    return 0
    #1 represents the two lines are the same equation
    return 1


def pre_process(xBar,  thetaOpt,  lamOpt,  muOpt,  y):
    '''Preprocessing step'''

    (M,  K) = np.shape(xBar)
    (K,  N) = np.shape(thetaOpt)
    #qualify_flag: mark whether the coefficients of the qualifying
    # constraint is all zero(0) or not(1)
    qualify_flag = np.zeros((K, N))
    #Coefficients: store the coefficients for KN lines.
    # The size if KN * (MK + 1)
    coefficients = []

    #Create a new model without optimization
    m = gb.Model("Just_get_the_coefficients")

    #Add variables
    x = [[0 for k in xrange(K)] for j in xrange(M)]
    for (j,k) in it.product(range(M), range(K)):
        x[j][k] = m.addVar(lb=-gb.GRB.INFINITY, ub=gb.GRB.INFINITY,
                    vtype=gb.GRB.CONTINUOUS, name="x_%d_%d" % (j,k) )
    m.update()

    #get the coefficients
    x0_x = np.dot(xBar.T, x)
    x0_x0 = np.dot(xBar.T, xBar)
    x0_x_x = x0_x + x0_x.T
    for i in xrange(N):
        for k in xrange(K):
            g_expr = gb.LinExpr()
            g_expr.addConstant(-2* np.dot(x0_x0[:, k], \
                   thetaOpt[:,i]) - lamOpt[i] - muOpt[k,i])
            S = 0
            for j in xrange(M):
                S += x[j][k] * float(y[j,i])

            g_expr.add(-2*S)
            g_expr.add(2 * np.dot(x0_x_x[:,k], thetaOpt[:,i]))
```

```
                coef = get_the_coefficient(g_expr, M, K, m)
                # Check and mark if a line has all the 0 coefficients
                Flag = check_line(coef, M, K)
                if Flag:
                    qualify_flag[k][i] = 0 # zero line
                    coefficients.append(coef)
                else:
                    qualify_flag[k][i] = 1 #not zero line
                    coefficients.append(coef)

    # Check a pair of two lines are the same or not.
    # Save the index of the duplicate qualifying constraints
    # in the repli_marker.
    repli_marker = [[] for x in xrange(N*K)]
    for i in xrange(len(coefficients)):
        if qualify_flag[i%K][i/K] != 0:
            for j in xrange(i+1, len(coefficients)):
                if qualify_flag[j%K][j/K] != 0:
                    if replicate_line_with_threshold\
                    (coefficients[i], coefficients[j], M, K, 1e-4):
                        repli_marker[i].extend([(j%K, j/K)])
                        repli_marker[j].extend([(i%K, i/K)])
    return qualify_flag, repli_marker, coefficients


def unique_coeff(g_flag, repli_marker, coefficients, M, K, N):
    '''Get the list of unique coefficients'''

    #Get the marker of the unqiue coefficient
    marker = [0 for i in xrange(len(coefficients))]
    for i in xrange(len(coefficients)):
        if marker[i] == 0:
            if g_flag[i%K][i/K] == 0:
                marker[i] = -1
            else:
                ## repli_marker[i] == empty
                if not repli_marker[i]:
                    marker[i] = 0
                else:
                    for j in xrange(len(repli_marker[i])):
                        marker[repli_marker[i][j][0] \
                        + repli_marker[i][j][1]*K] = -1
    #Get the unique coefficient based on the marker
    linker = []
    Unique_coefficient = []
    for i in xrange(len(marker)):
        if marker[i] == 0:
            linker.append(i)
            Unique_coefficient.append(coefficients[i])
    return linker, np.array(Unique_coefficient)


def extend_back(thetaB_list, linker, g_flag, repli_marker, K, N):
    '''change the formula of thetaB_list to the matrix.'''

    extend_list = []
    for i in xrange(len(thetaB_list)):
      thetaB = np.zeros( (K, N))
      #set zero lines
      for n in xrange(N):
          for k in xrange(K):
              thetaB[k][n] = -2
              if g_flag[k][n] == 0:
                  thetaB[k][n] = -1
      #set value
      for j in xrange(len(linker)):
          row = linker[j]/K
          col = linker[j]%K
          thetaB[col][row] = int(thetaB_list[i][j])
      for j in xrange(len(repli_marker)):
          if repli_marker[j] != []:
              if thetaB[j%K][j/K] == -2:
                 for n in xrange(len(repli_marker[j])):
                   value = \
                   thetaB[repli_marker[j][n][0]][repli_marker[j][n][1]]
                   if value != -2:
                      thetaB[j%K][j/K] = value
      extend_list.append(thetaB)
    return extend_list

if __name__ == "__main__":
    pass
```

C.2.4 Cell enumeration

```python
__author__ = 'Fan Zhang & Chuangqi Wang'
# parallel_cell_enumeration.py

import interior_point as IP
import copy
import numpy as np
from time import time
import time
from multiprocessing import Pool
import itertools

def reflection(thetaB):
    ''' Get the thetaB for the reflected region: 0 ->1, 1->0
    thetaB: the sign of hyperplane '''

    ref_thetaB = []
    for i in xrange(len(thetaB)):
        ref_thetaB.append(thetaB[i])
        if thetaB[i] == 1:
            ref_thetaB[i] = 0
        elif thetaB[i] == 0:
            ref_thetaB[i] = 1
    return ref_thetaB

def initialize_region(coefficients,  nCuts, dim,  threshold):
    ''' Calculate the first feasible region
    coefficients: the coefficients of all hyperplanes.
    nCuts: the number of hyperplanes.
    Dim: the dimension of space.
    threshold: tolerance of the distance from the point to
    a hyperplane'''

    #test the region one by one unitl one feasible is found
    NBC = pow(2,  nCuts)
    for index in xrange(NBC):
        thetaBstr = "{0:b}".format(index).zfill(nCuts)
        marker = list(thetaBstr)
        result = [int(i) for i in marker]
        status, xOpt, obj = IP.interior_point(coefficients, \
                                 nCuts, dim, result)
        if status == 2 and obj >= threshold:
            return result

def check_candidate_hyperplanes(coefficients, \
            candidate_hyperplanes, nCuts, dim, region, \
            threshold, pool = None):
    ''' Check if each candidate hyperplane is the strict hyperplane.
    coefficients: the coefficients of all hyperplanes.
    candidate_hyperplanes: check this hyperplane is strict/redundant.
    nCuts: the number of hyperplanes.
    Dim: the dimension of space.
    region: the sign of the hyperplanes.
    Adjacency regions of this region will be return.
    threshold: tolerance of the distance from the point to
    a hyperplane'''

    #non-redundant hyperplane
    strict_hyperplanes = []

    if pool == None:

        #check the candidate hyperplanes
        for can_hp in candidate_hyperplanes:
            new_region = flip(region, can_hp)
            status,  xOpt,  obj = IP.interior_point(coefficients,\
                                  nCuts,  dim,  new_region)
            if status == 2 and obj >= threshold:
                strict_hyperplanes.append(can_hp)
    else:
        #Parallel
        results = [pool.apply_async(check_interior_point, \
                args = (coefficients,  nCuts,  dim,  region, \
                can_hp, threshold)) for can_hp \
                in candidate_hyperplanes]
        for p in results:
                #result = [can_hp, True or false]
                result = p.get()
                if result[1]:
                    strict_hyperplanes.append(result[0])
    return strict_hyperplanes

def check_interior_point(coefficients, nCuts, dim, region, can_hp,\
                            threshold):
    ''' In order to parallel, we use this function to
    check the new region exist or not'''
    new_region = flip(region,  can_hp)
    status, xOpt, obj = IP.interior_point(coefficients, \
                          nCuts, dim, new_region)
    if status == 2 and obj >= threshold:
```

```python
        return can_hp,  True
    else:
        return can_hp,  False


def cal_adjacency_regions(coefficients, nCuts, dim, region, \
    certain_hyperplane, threshold, pool =None, closedset=[]):
    ''' Calculate the adjacency regions of the given region.
    This function icludes two parts:
    1) calculate the strict hyperplanes.
    2) get the adjacency regions based on the strict hyperplanes.

    coefficients: the coefficients of all hyperplanes.
    nCuts: the number of hyperplanes.
    Dim: the dimension of space.
    region: the sign of the hyperplanes. Adjacency regions of
    this region will be return.
    certain_hyperplane: shows which hyperplane we are considering.
    threshold: tolerance of the distance from the point to
    a hyperplane'''

    #the calculated adjacency regions
    adj_regions = []

    #put all the hyperplanes into candidate_hyperplanes
    candidate_hyperplanes = list(xrange(nCuts))

    #test the other candidate hyperplanes.
    strict_hyperplanes = check_candidate_hyperplanes(coefficients,\
       candidate_hyperplanes, nCuts, dim, region, threshold, pool)

    #flip the strict hyperplane to get the adjacency regions.
    for i in strict_hyperplanes:
        if i != certain_hyperplane:
            flip_region = flip(region, i)
            if flip_region not in closedset:
                adj_regions.append(flip_region)

    #return the adjacency regions
    return adj_regions

def flip(region,  index):
    '''  flip the element of index. 1 -> 0,  0 -> 1
    region: the sign of hyperplanes
    index: the position should be fliped. '''

    flip_region = copy.copy(region)
    if flip_region[index] == 0:
        flip_region[index] = 1
    elif flip_region[index] == 1:
        flip_region[index] = 0
    else:
        assert(1 == 0),  'No 0 and 1 appears in the thetaB_list.'
    return flip_region

def parallel_cell_numeration(coefficients, nCuts, dim, threshold,\
            pool = None,  certain_hyperplane = 0):
    '''  Get the thetaB_list of all the regions.
    coefficients: the coefficients of all hyperplanes.
    nCuts: the number of hyperplanes.
    Dim: the dimension of space.
    threshold: the tolerance of the distance from the point
    to a hyperplane.
    certain_hyperplane: assume the sign of this hyperplane
    does not change. '''

    #sub_NUM_CORES = 10
    #Parallel
    #sub_pool = Pool(processes=sub_NUM_CORES)

    thetaB_list = []
    #the structure to manage the process of all the function.
    #openset = Queue.Queue()
    openset = []
    closedset = []

    Initialized_region = initialize_region(coefficients, \
                              nCuts, dim, threshold)

    #openset.put(Initialized_region)
    openset.append(Initialized_region)
    print ('\nfinding adjacent regions...')

    # non-parallel
    if pool == None:
        while len(openset) != 0:
          #get the region and delete it from the openset.
          region = openset.pop(0)
          closedset.append(region)

```

```
            #calculate the adjacency regions
            adj_regions = cal_adjacency_regions(coefficients,
              nCuts, dim, region, certain_hyperplane, threshold)

            for region in adj_regions:
              #if region never appears.
              if region not in openset and region not in closedset:
                    openset.append(region)
    # parallel
    else:
        start = time.clock()
        while len(openset) != 0:
          closedset.extend(openset)
          results = [pool.apply_async(cal_adjacency_regions,\
          args = (coefficients, nCuts, dim, region, \
          certain_hyperplane, threshold, None, closedset)) \
          for region in openset]

          #set openset is empty
          openset = []
          #get the result
          for p in results:
              openset.extend(p.get())
          #get the unique list
          openset.sort()
          openset=list(openset for openset, \
                  _ in itertools.groupby(openset))

        end = time.clock()
        time_adj_region = end - start
        print ('\nFinish finding adjacent regions using: %0.2f'\
               %(time_adj_region))

    #Reflection
    for region in closedset:
        thetaB_list.append(region)
        ref_thetaB = reflection(region)
        thetaB_list.append(ref_thetaB)

    return thetaB_list

if __name__ == "__main__":
    pass
```

C.2.5 Relaxed dual problems

```python
__author__ = 'Fan Zhang'
# parallel_masterprob.py

import numpy as np
import gurobipy as gb
import itertools as it
from itertools import repeat
import time


def add_qualifying_constraint(m, coefficients, \
    M, K, N, thetaB, g_flag, t):
    ''' Add the qualifying constraints to model m.  The qualifying
        constraint is formed by linearizing the Lagrangian with
        respect to x about x0. Then linearizing that with respect
        to theta about thetat.  '''

    qualifying_constraint = []
    x = [[m.getVarByName("x_%d_%d" % (j,k)) for k in xrange(K)] \
        for j in xrange(M)]


    for i in xrange(len(coefficients)):
        #calcuate the qualifying constraint formula
        g_expr = gb.LinExpr()
        g_expr.addConstant(coefficients[i][-1])
        for j in xrange(M*K):
            g_expr.add(x[j/K][j%K]* coefficients[i][j])
        qualifying_constraint.append(g_expr)
        #add the qualifying constraints
        if g_flag[i%K][i/K] != 0:
            # Add constraints: g_expr
            if thetaB[i%K][i/K] == 1:
                m.addConstr(g_expr <= np.spacing(1), \
                  name="qc%d_%d_%d" % (t,k,i))
            elif thetaB[i%K][i/K] == 0:
                m.addConstr(g_expr >= -np.spacing(1), \
                  name="qc%d_%d_%d" % (t,k,i))
            m.update()


    #return qualifying constraints to calculate lagrange constraint
    return qualifying_constraint

```

```
def add_previous_lagrangian_constraint(m, lagrangian_coefficient,\
                                       M, K, N, t):
    '''Add previous linearized lagrangian constraints to model m.
    '''

    #Extract the variables from the model.
    Q = m.getVarByName("Q")
    x = [[m.getVarByName("x_%d_%d" % (j,k)) for k in xrange(K)] \
        for j in xrange(M)]

    #Initialize the constraint.
    L = gb.LinExpr()
    for j in xrange(M*K):
        L.add(x[j/K][j%K] * lagrangian_coefficient[j])
    L.addConstant(lagrangian_coefficient[-1])
    m.addConstr(Q >= L,  name = "Lagrangian_%d"%t)
    m.update()


def add_lagrangian_constraint(m, qualifying_constraint, xt, \
                              thetat, thetaB, lam, mu, y, t):
    ''' Add the linearized Lagrangian constraint to model m.
    The lagrangian is linearized with respect to theta0\
    and calculated at theta_i
    '''

    (M,N) = np.shape(y)
    K = np.shape(thetat)[0]

    # Extract the variables from the model
    Q = m.getVarByName("Q")
    x = [[m.getVarByName("x_%d_%d" % (j,k)) for k in xrange(K)] \
          for j in xrange(M)]

    # Initialize the constraint.
    # The lagrangian constraint is linear in theta
    L = gb.LinExpr()
    x0_x = np.dot(xt.T, x)
    x0_x0 = np.dot(xt.T, xt)

    for i in xrange(N):
        L0 = np.dot(y[:,i].T, y[:,i])
        L0 += -np.dot(thetat[:,i], np.dot(x0_x0, thetat[:,i]))
        L0 += -2 * np.dot( np.dot(y[:,i].T, x), thetat[:,i])
        L0 += 2 * np.dot(thetat[:,i], np.dot( x0_x, thetat[:,i]))
        L.add(L0)

        L.addConstant(-lam[i] * (np.sum(thetat[:,i]) - 1))
        L.addConstant( -np.dot(mu[:,i].T, thetat[:,i]) )

        for j in xrange(K):
            L.add( qualifying_constraint[i*K + j] \
              * (thetaB[j, i] - thetat[j, i]))

    m.addConstr(Q >= L, name = "Lagrangian_%d" % t)
    m.update()


    #Get the coefficients of lagrangian constraints
    num = L.size()
    coeff_constant = L.getConstant()
    coeff = np.zeros((M,  K))
    for index in xrange(num):
        var_name = L.getVar(index).VarName.encode('ascii','ignore')
        indexes = [ind for ind, \
        ltr in enumerate(var_name) if ltr == '_']
        i = int(var_name[indexes[0] + 1 : indexes[1]])
        j = int(var_name[indexes[1] + 1 : ])
        coeff[i][j] += L.getCoeff(index)

    lagrangian_coef = []
    for i in xrange(M):
        for j in xrange(K):
            lagrangian_coef.append(float(coeff[i][j]))
    lagrangian_coef.append(float(coeff_constant))
    return lagrangian_coef


def solve_master(tree, num_node, Current_node, g_flag, \
    thetaB_list, SUBD, coefficients, xBar, thetaOpt, \
    lamOpt, muOpt, y,  cons, iteration,  pool=None):
    '''we solve the relaxed master problems based on thetaB_list,
    then select the infimum of all minimum values.
    Parameters:
    About the tree : tree, Current_node
    About the subproblem: SUBD, xBar, thetaOpt, lamOpt, muOpt, y
    About the boundary: theta_L, theta_U'''

    (M, N) = np.shape(y)
    K = np.shape(xBar[-1])[0]

    x_stor = None
    Q_stor = np.inf
    next_node = -1

    #store all the MLBD
    MLBD_stor = []

    #store all the MLBD
    if pool == None:
        tree.nodes[Current_node].\
        set_parameters_qualifying_constraint(lamOpt, thetaOpt, \
        muOpt, xBar, SUBD, g_flag, coefficients)
        # check whether the coefficients are already stored
        # into the parents or not.

        print ('\n%d master problems are solving...' \
               %len(thetaB_list))

        for index in xrange(len(thetaB_list)):
            thetaB = thetaB_list[index].copy()
            status, objVal, xOpt, thetaB, lagrangian_coefficient =\
            solve_master_s(tree, Current_node, coefficients, \
            thetaOpt, xBar, lamOpt, muOpt, thetaB.copy(), \
            y, g_flag, cons)
            #print objVal, xOpt

            if status == 2 and objVal < SUBD - np.spacing(1):
                node = tree.add_node(num_node, 0, 1, Current_node)
                node.set_parameters_thetaB(thetaB,  xOpt, \
                                 objVal, lagrangian_coefficient)
                MLBD_stor.append(objVal)
                if objVal < Q_stor:
                    Q_stor = objVal
                    next_node = num_node
                    x_stor = xOpt
                num_node = num_node + 1

    else:
        tree.nodes[Current_node].\
        set_parameters_qualifying_constraint(lamOpt, thetaOpt,\
        muOpt, xBar, SUBD, g_flag, coefficients)
        len_thetaB = len(thetaB_list)

        print ('\n%d master problems are solving...'  %len_thetaB)
        results = [pool.apply_async(solve_master_s, args = (tree,\
        Current_node, coefficients, thetaOpt, xBar, lamOpt, muOpt,\
        thetaB.copy(), y, g_flag, cons)) for thetaB in thetaB_list]

        #put all the result into the tree.
        for p in results:
            #result = [status, objVal, xOpt, thetaB]
            result = p.get()
            if result[0] == 2 and result[1] < SUBD - np.spacing(1):
                node = tree.add_node(num_node, 0, 1, Current_node)
                node.set_parameters_thetaB(result[3], result[2],\
                                           result[1], result[4])

                MLBD_stor.append(result[1])
                if result[1] < Q_stor:
                    Q_stor = result[1]
                    next_node = num_node
                    x_stor =  result[2]
                num_node += 1

    return x_stor, Q_stor, next_node, num_node, MLBD_stor

def solve_master_s(tree, Current_node, coefficients, thetaOpt, \
                   xBar, lamOpt, muOpt, thetaB, y, g_flag, cons):
    '''Solve one master problem using Gurobipy'''

    (M,  K) = np.shape(xBar)
    (M,  N) = np.shape(y)

    ################# Create a new model ####################
    m = gb.Model("master")

    ############### Set optimization parameters #############
    m.setParam('OutputFlag', False)

    ########### Create decision variables ############
    x = [[0 for k in xrange(K)] for j in xrange(M)]
    for (j,k) in it.product(range(M), range(K)):
        x[j][k] = m.addVar(lb=-100, ub=100,
            vtype=gb.GRB.CONTINUOUS, name="x_%d_%d" % (j,k) )

    # Create the slack variable for the objective
    Q = m.addVar(lb=-gb.GRB.INFINITY, ub=gb.GRB.INFINITY, name="Q")

    # Integrate decision variables
    m.update()

    ################ Create the objective: min_x Q ##############
    m.setObjective(Q, gb.GRB.MINIMIZE)

    ################# Add constraints ########################
    # Add the sparsity constraint
    s = [[2 for k in xrange(K)] for j in xrange(M)]
    for (j,k) in it.product(range(M), range(K)):
        s[j][k] = m.addVar(lb=0.0, ub = gb.GRB.INFINITY, \
        name="s_%d_%d" % (j,k) )
    m.update()

    for (j,k) in it.product(range(M), range(K)):
      m.addConstr(x[j][k] <= s[j][k], name="L1ub_s_%d_%d" %(j,k))
      m.addConstr(x[j][k] >= -s[j][k], name="L1_lb_s_%d_%d"%(j,k))

    obj = gb.LinExpr()
    for (j,k) in it.product(range(M), range(K)):
        obj.add(s[j][k])
    m.addConstr( obj <= cons, name="L1_sum" )
    m.update()


    #Compose the Lagrangian and qualifying constraints
    identifier = Current_node
    t = 0

    while identifier != 0:
        parent = tree.return_parent(identifier)
        qualifying_constraint= add_qualifying_constraint(m,\
        tree.nodes[parent].coefficients, M, K, N, \
        tree.nodes[identifier].thetaB, tree.nodes[parent].g_flag,t)
        add_previous_lagrangian_constraint(m, \
        tree.nodes[identifier].lagrangian_coef, M, K, N, t)
        identifier = parent
        t += 1

    qualifying_constraint = add_qualifying_constraint \
         (m, coefficients, M, K, N, thetaB, g_flag, t)
    lagrangian_coefficient = add_lagrangian_constraint \
         (m, qualifying_constraint, xBar, thetaOpt, \
        thetaB, lamOpt, muOpt, y, t)


    ############## Optimize the master problem ################
    try:
        m.optimize()
    except gb.GurobiError as e:
        print e.message

    ############### Check optimization results #########
    if m.Status == gb.GRB.OPTIMAL:
        #Extract to optimal value and the optimum
        xOpt = np.empty((M, K))
        for (j, k) in it.product(range(M), range(K)):
            xOpt[j, k] = round(x[j][k].x,  4)

        return (m.Status, m.objVal, xOpt, \
                thetaB, lagrangian_coefficient)
    else:
        return(m.Status, np.inf, np.nan, \
                 np.nan, lagrangian_coefficient)
```

Bibliography

[1] Eric D Green, Mark S Guyer, National Human Genome Research Institute, et al. Charting a course for genomic medicine from base pairs to bedside. *Nature*, 470(7333):204–213, 2011.

[2] Andriy Marusyk and Kornelia Polyak. Tumor heterogeneity: causes and consequences. *Biochimica et Biophysica Acta (BBA)-Reviews on Cancer*, 1805(1):105–117, 2010.

[3] Qinghua Zhou and Zongzhi Liu. A pilot study of next-generation sequencing on cell-free DNA from blood plasma and bone marrow fluid for detecting leukemic clonal abnormalities. *North American Journal of Medicine and Science*, 7(4), 2014.

[4] Marco Gerlinger, Andrew J Rowan, Stuart Horswell, James Larkin, David Endesfelder, Eva Gronroos, Pierre Martinez, Nicholas Matthews, Aengus Stewart, Patrick Tarpey, Ignacio Varela, Benjamin Phillimore, Sharmin Begum, Neil Q McDonald, Adam Butler, David Jones, Keiran Raine, Calli Latimer, Claudio R Santos, Mahrokh Nohadani, Aron C Eklund, Bradley Spencer-Dene, Graham Clark, Lisa Pickering, Gordon Stamp, Martin Gore, Zoltan Szallasi, Julian Downward, P Andrew Futreal, and Charles Swanton. Intratumor heterogeneity and branched evolution revealed by multiregion sequencing. *N. Engl. J. Med.*, 366(10):883–92, March 2012.

[5] Therese Sørlie, Robert Tibshirani, Joel Parker, Trevor Hastie, James Stephen Marron, Andrew Nobel, Shibing Deng, Hilde Johnsen, Robert Pesich, Stephanie Geisler, et al. Repeated observation of breast tumor subtypes in independent gene expression data sets. *Proceedings of the National Academy of Sciences*, 100(14):8418–8423, 2003.

[6] Qingxuan Song, Sofia D Merajver, and Jun Z Li. Cancer classification in the genomic era: five contemporary problems. *Human genomics*, 9(1):1, 2015.

[7] Roel GW Verhaak, Katherine A Hoadley, Elizabeth Purdom, Victoria Wang, Yuan Qi, Matthew D Wilkerson, C Ryan Miller, Li Ding, Todd Golub, Jill P Mesirov, et al. Inte-

grated genomic analysis identifies clinically relevant subtypes of glioblastoma characterized by abnormalities in PDGFRA, IDH1, EGFR, and NF1. *Cancer cell*, 17(1):98–110, 2010.

[8] Stephan Pabinger, Andreas Dander, Maria Fischer, Rene Snajder, Michael Sperk, Mirjana Efremova, Birgit Krabichler, Michael R Speicher, Johannes Zschocke, and Zlatko Trajanoski. A survey of tools for variant analysis of next-generation genome sequencing data. *Briefings in bioinformatics*, 15(2):256–278, 2014.

[9] Riyue Bao, Lei Huang, Jorge Andrade, Wei Tan, Warren a Kibbe, Hongmei Jiang, and Gang Feng. Review of current methods, applications, and data management for the bioinformatics analysis of whole exome sequencing. *Libertas Academica*, 13:67–82, 2014.

[10] Elizabeth T. Cirulli and David B. Goldstein. Uncovering the roles of rare variants in common disease through whole-genome sequencing. *Nature Reviews Genetics*, 11(6):415–425, 2010.

[11] Kelly A. Frazer, Sarah S. Murray, Nicholas J. Schork, and Eric J. Topol. Human genetic variation and its contribution to complex traits. *Nature Reviews Genetics*, 10(4):241–251, 2009.

[12] Jennifer Asimit and Eleftheria Zeggini. Rare variant association analysis methods for complex traits. *Annual review of genetics*, 44:293–308, 2010.

[13] Patrick Flaherty, Georges Natsoulis, Omkar Muralidharan, Mark Winters, Jason Buenrostro, John Bell, Sheldon Brown, Mark Holodniy, Nancy Zhang, and Hanlee P. Ji. Ultrasensitive detection of rare mutations using next-generation targeted resequencing. *Nucleic Acids Research*, 40(1):1–12, 2012.

[14] Erin D Pleasance, R Keira Cheetham, Philip J Stephens, David J McBride, Sean J Humphray, Chris D Greenman, Ignacio Varela, Meng-Lay Lin, Gonzalo R Ordóñez, Graham R Bignell, et al. A comprehensive catalogue of somatic mutations from a human cancer genome. *Nature*, 463(7278):191–196, 2010.

[15] Nicola D Roberts, R Daniel Kortschak, Wendy T Parker, Andreas W Schreiber, Susan Branford, Hamish S Scott, Garique Glonek, and David L Adelson. A comparative analysis of algorithms for somatic SNV detection in cancer. *Bioinformatics*, page btt375, 2013.

[16] Kristian Cibulskis, Michael S Lawrence, Scott L Carter, Andrey Sivachenko, David Jaffe, Carrie Sougnez, Stacey Gabriel, Matthew Meyerson, Eric S Lander, and Gad Getz. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nature biotechnology*, 31(3):213–219, 2013.

[17] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, et al. The genome analysis toolkit: a mapreduce framework for analyzing next-generation DNA sequencing data. *Genome research*, 20(9):1297–1303, 2010.

[18] C. T. Saunders, W. S. W. Wong, S. Swamy, J. Becq, L. J. Murray, and R. K. Cheetham. Strelka: accurate somatic small-variant calling from sequenced tumor-normal sample pairs. *Bioinformatics*, 28(14):1811–1817, 2012.

[19] David E Larson, Christopher C Harris, Ken Chen, Daniel C Koboldt, Travis E Abbott, David J Dooling, Timothy J Ley, Elaine R Mardis, Richard K Wilson, and Li Ding. Somaticsniper: identification of somatic point mutations in whole genome sequencing data. *Bioinformatics*, 28(3):311–317, 2012.

[20] Yuichi Shiraishi, Yusuke Sato, Kenichi Chiba, Yusuke Okuno, Yasunobu Nagata, Kenichi Yoshida, Norio Shiba, Yasuhide Hayashi, Haruki Kume, Yukio Homma, et al. An empirical Bayesian framework for somatic mutation detection from cancer genome sequencing data. *Nucleic acids research*, 41(7):e89–e89, 2013.

[21] D. C. Koboldt, Q. Zhang, D. E. Larson, D. Shen, M. D. McLellan, L. Lin, C. A. Miller, E. R. Mardis, L. Ding, and R. K. Wilson. VarScan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome Research*, 22(3):568–576, 2012.

[22] Nancy F. Hansen, Jared J. Gartner, Lan Mei, Yardena Samuels, and James C. Mullikin. Shimmer: Detection of genetic alterations in tumors using next-generation sequence data. *Bioinformatics*, 29(12):1498–1503, 2013.

[23] Rasmus Nielsen, Joshua S. Paul, Anders Albrechtsen, and Yun S. Song. Genotype and SNP calling from next-generation sequencing data. *Nature Reviews Genetics*, 12(6):443–451, 2011.

[24] Moritz Gerstung, Christian Beisel, Markus Rechsteiner, Peter Wild, Peter Schraml, Holger Moch, and Niko Beerenwinkel. Reliable detection of subclonal single-nucleotide variants in tumour cell populations. *Nature communications*, 3:811, 2012.

[25] Alexis Christoforides, John D Carpten, Glen J Weiss, Michael J Demeure, Daniel D Von Hoff, and David W Craig. Identification of somatic mutations in cancer through Bayesian-based analysis of sequenced genome pairs. *BMC genomics*, 14(1):302, 2013.

[26] Heidi Schwarzenbach, Dave SB Hoon, and Klaus Pantel. Cell-free nucleic acids as biomarkers in cancer patients. *Nature Reviews Cancer*, 11(6):426–437, 2011.

[27] Oliver A Zill, Claire Greene, Dragan Sebisanovic, Lai Mun Siew, Jim Leng, Mary Vu, Andrew E Hendifar, Zhen Wang, Chloe E Atreya, Robin K Kelley, et al. Cell-free DNA next-generation sequencing in pancreatobiliary carcinomas. *Cancer discovery*, 5(10):1040–1048, 2015.

[28] Tim Forshew, Muhammed Murtaza, Christine Parkinson, Davina Gale, Dana WY Tsui, Fiona Kaper, Sarah-Jane Dawson, Anna M Piskorz, Mercedes Jimenez-Linan, David Bentley, et al. Noninvasive identification and monitoring of cancer mutations by targeted deep sequencing of plasma DNA. *Science translational medicine*, 4(136):136ra68–136ra68, 2012.

[29] Huilei Xu, John DiCarlo, Ravi Vijaya Satya, Quan Peng, and Yexun Wang. Comparison of somatic mutation calling methods in amplicon and whole exome sequence data. *BMC genomics*, 15(1):244, 2014.

[30] Howard W. Huang, James C. Mullikin, and Nancy F. Hansen. Evaluation of variant detection software for pooled next-generation sequence data. *BMC Bioinformatics*, 16(1):235, 2015.

[31] Yuting He, Fan Zhang, and Patrick Flaherty. RVD2: An ultra-sensitive variant detection model for low-depth heterogeneous next-generation sequencing data. *Bioinformatics*, page btv275, 2015.

[32] Nathan D Olson, Steven P Lund, Rebecca E Colman, Jeffrey T Foster, Jason W Sahl, James M Schupp, Paul Keim, Jayne B Morrow, Marc L Salit, and Justin M Zook. Best practices for evaluating single nucleotide variant calling methods for microbial genomics. *Frontiers in genetics*, 6, 2015.

[33] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[34] Carsten Peterson and Eric Hartman. Explorations of the mean field theory learning algorithm. *Neural Networks*, 2(6):475–494, 1989.

[35] Patrick Flaherty, Guri Giaever, Jochen Kumm, Michael I Jordan, and Adam P Arkin. A latent variable model for chemogenomic profiling. *Bioinformatics*, 21(15):3286–3293, 2005.

[36] Ravi K Patel and Mukesh Jain. NGS QC Toolkit: a toolkit for quality control of next generation sequencing data. *PloS one*, 7(2):e30619, 2012.

[37] Qi Liu, Yan Guo, Jiang Li, Jirong Long, Bing Zhang, and Yu Shyr. Steps to ensure accuracy in genotype and snp calling from illumina sequencing data. *BMC genomics*, 13(8):1, 2012.

[38] Joseph A Neuman, Ofer Isakov, and Noam Shomron. Analysis of insertion–deletion from deep-sequencing data: software evaluation for optimal detection. *Briefings in Bioinformatics*, 14(1):46–55, 2013.

[39] Peter Krawitz, Christian Rödelsperger, Marten Jäger, Luke Jostins, Sebastian Bauer, and Peter N Robinson. Microindel detection in short-read sequence data. *Bioinformatics*, 26(6):722–729, 2010.

[40] A. Y. Cheng, Y.-Y. Teo, and R. T.-H. Ong. Assessing single nucleotide variant detection and genotype calling on whole-genome sequenced individuals. *Bioinformatics*, 30(12):1707–1713, 2014.

[41] Xiangtao Liu, Shizhong Han, Zuoheng Wang, Joel Gelernter, and Bao-Zhu Yang. Variant callers for next-generation sequencing data: a comparison study. *PloS one*, 8(9):e75619, 2013.

[42] Soumya Raychaudhuri. Mapping rare and common causal alleles for complex human diseases. *Cell*, 147(1):57–69, 2011.

[43] Christian Schlötterer, Raymond Tobler, Robert Kofler, and Viola Nolte. Sequencing pools of individuals - mining genome-wide polymorphism data without big funding. *Nature Reviews Genetics*, 15(11):749–763, 2014.

[44] Chad Garner. Confounded by sequencing depth in association studies of rare alleles. *Genetic epidemiology*, 35(4):261–268, 2011.

[45] Kensuke Nakamura, Taku Oshima, Takuya Morimoto, Shun Ikeda, Hirofumi Yoshikawa, Yuh Shiwa, Shu Ishikawa, Margaret C Linak, Aki Hirai, Hiroki Takahashi, et al. Sequence-specific error profile of illumina sequencers. *Nucleic acids research*, page gkr344, 2011.

[46] Lira Mamanova, Alison J Coffey, Carol E Scott, Iwanka Kozarewa, Emily H Turner, Akash Kumar, Eleanor Howard, Jay Shendure, and Daniel J Turner. Target-enrichment strategies for next-generation sequencing. *Nature methods*, 7(2):111–118, 2010.

[47] Michael D Linderman, Tracy Brandt, Lisa Edelmann, Omar Jabado, Yumi Kasai, Ruth Kornreich, Milind Mahajan, Hardik Shah, Andrew Kasarskis, and Eric E Schadt. Analytical validation of whole exome and whole genome sequencing for clinical applications. *BMC medical genomics*, 7(1):1, 2014.

[48] Jay Shendure and Hanlee Ji. Next-generation DNA sequencing. *Nature biotechnology*, 26(10):1135–1145, 2008.

[49] Ali May, Sanne Abeln, Mark J Buijs, Jaap Heringa, Wim Crielaard, and Bernd W Brandt. NGS-eval: NGS error analysis and novel sequence variant detection tool. *Nucleic acids research*, page gkv346, 2015.

[50] Si Quang Le and Richard Durbin. SNP detection and genotyping from low-coverage sequencing data on multiple diploid samples. *Genome research*, 21(6):952–960, 2011.

[51] Mark A DePristo, Eric Banks, Ryan Poplin, Kiran V Garimella, Jared R Maguire, Christopher Hartl, Anthony A Philippakis, Guillermo Del Angel, Manuel A Rivas, Matt Hanna, et al. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics*, 43(5):491–498, 2011.

[52] Heng Li, Jue Ruan, and Richard Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome research*, 18(11):1851–1858, 2008.

[53] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, et al. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[54] Heng Li. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, 27(21):2987–2993, 2011.

[55] Goncalo Abecasis. Abecasis Lab GLF Tools. `http://csg.sph.umich.edu//abecasis/glfTools/`, 2010.

[56] G. Peng, Y. Fan, T. B. Palculict, P. Shen, E. C. Ruteshouser, A.-K. Chi, R. W. Davis, V. Huff, C. Scharfe, and W. Wang. Rare variant detection using family-based sequencing analysis. *Proceedings of the National Academy of Sciences*, 110(10):3985–3990, 2013.

[57] Andrew Roth, Jiarui Ding, Ryan Morin, Anamaria Crisan, Gavin Ha, Ryan Giuliany, Ali Bashashati, Martin Hirst, Gulisa Turashvili, Arusha Oloumi, et al. Jointsnvmix: a probabilistic model for accurate detection of somatic mutations in normal/tumour paired next-generation sequencing data. *Bioinformatics*, 28(7):907–913, 2012.

[58] Sangwoo Kim, Kyowon Jeong, Kunal Bhutani, Jeong Lee, Anand Patel, Eric Scott, Hojung Nam, Hayan Lee, Joseph G Gleeson, and Vineet Bafna. Virmid: accurate detection of somatic mutations with sample impurity inference. *Genome Biology*, 14(8):R90, 2013.

[59] Quan Chen and Fengzhu Sun. A unified approach for allele frequency estimation, SNP detection and association studies based on pooled sequencing data using EM algorithms. *BMC genomics*, 14(1):1, 2013.

[60] Erik Garrison and Gabor Marth. Haplotype-based variant detection from short-read sequencing. *arXiv preprint arXiv:1207.3907*, page 9, 2012.

[61] Baiyu Zhou. An empirical Bayes mixture model for SNP detection in pooled sequencing data. *Bioinformatics*, 28(20):2569–2575, 2012.

[62] Emanuele Raineri, Luca Ferretti, Anna Esteve-Codina, Bruno Nevado, Simon Heath, and Miguel Pérez-Enciso. SNP calling by sequencing pooled samples. *BMC Bioinformatics*, 13(1):239, 2012.

[63] Rodrigo Goya, Mark G F Sun, Ryan D. Morin, Gillian Leung, Gavin Ha, Kimberley C. Wiegand, Janine Senz, Anamaria Crisan, Marco A. Marra, Martin Hirst, David Huntsman, Kevin P. Murphy, Sam Aparicio, and Sohrab P. Shah. SNVMix: Predicting single nucleotide variants from next-generation sequencing of tumors. *Bioinformatics*, 26(6):730–736, 2010.

[64] Ruiqiang Li, Yingrui Li, Xiaodong Fang, Huanming Yang, Jian Wang, Karsten Kristiansen, and Jun Wang. SNP detection for massively parallel whole-genome resequencing. *Genome Research*, 19(6):1124–1132, 2009.

[65] Bingshan Li, Wei Chen, Xiaowei Zhan, Fabio Busonero, Serena Sanna, Carlo Sidore, Francesco Cucca, Hyun M Kang, and Gonçalo R Abecasis. A likelihood-based framework for variant calling and de novo mutation detection in families. *PLoS Genet*, 8(10):e1002944, 2012.

[66] Todd E Druley, Francesco L M Vallania, Daniel J Wegner, Katherine E Varley, Olivia L Knowles, Jacqueline A Bonds, Sarah W Robison, Scott W Doniger, Aaron Hamvas, F Sessions Cole, Justin C Fay, and Robi D Mitra. Quantification of rare allelic variants from pooled genomic DNA. *Nature Methods*, 6(4):263–265, 2009.

[67] David H. Spencer, Manoj Tyagi, Francesco Vallania, Andrew J. Bredemeyer, John D. Pfeifer, Rob D. Mitra, and Eric J. Duncavage. Performance of common analysis methods for detecting low-frequency single nucleotide variants in targeted next-generation sequence data. *Journal of Molecular Diagnostics*, 16(1):75–88, 2014.

[68] Koen Van der Borght, Kim Thys, Yves Wetzels, Lieven Clement, Bie Verbist, Joke Reumers, Herman van Vlijmen, and Jeroen Aerssens. QQ-SNV: single nucleotide variant detection at low frequency by comparing the quality quantiles. *BMC Bioinformatics*, 16(1):379, 2015.

[69] Vikas Bansal. A statistical method for the detection of variants from next-generation resequencing of DNA pools. *Bioinformatics*, 26(12):i318–i324, 2010.

[70] Zhi Wei, Wei Wang, Pingzhao Hu, Gholson J Lyon, and Hakon Hakonarson. SNVer: a statistical tool for variant calling in analysis of pooled or individual next-generation sequencing data. *Nucleic acids research*, 39(19):e132–e132, 2011.

[71] Gabor T Marth, Ian Korf, Mark D Yandell, Raymond T Yeh, Zhijie Gu, Hamideh Zakeri, Nathan O Stitziel, LaDeana Hillier, Pui-Yan Kwok, and Warren R Gish. A general approach to single-nucleotide polymorphism discovery. *Nature genetics*, 23(4):452–456, 1999.

[72] Weichun Huang, Leping Li, Jason R Myers, and Gabor T Marth. ART: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2012.

[73] Robert C Elston and John Stewart. A general model for the genetic analysis of pedigree data. *Human heredity*, 21(6):523–542, 1971.

[74] Fan Zhang and Patrick Flaherty. Variational inference for rare variant detection in deep, heterogeneous next-generation sequencing data. *arXiv preprint arXiv:1604.04280*, 2016.

[75] Danny Challis, Jin Yu, Uday S Evani, Andrew R Jackson, Sameer Paithankar, Cristian Coarfa, Aleksandar Milosavljevic, Richard A Gibbs, and Fuli Yu. An integrative variant analysis suite for whole exome next-generation sequencing data. *BMC bioinformatics*, 13(1):1, 2012.

[76] Jiarui Ding, Ali Bashashati, Andrew Roth, Arusha Oloumi, Kane Tse, Thomas Zeng, Gholamreza Haffari, Martin Hirst, Marco A. Marra, Anne Condon, Samuel Aparicio, and Sohrab P. Shah. Feature-based classifiers for somatic mutation detection in tumour-normal paired sequencing data. *Bioinformatics*, 28(2):167–175, 2012.

[77] Mamunur Rashid, Carla Daniela Robles-Espinoza, Alistair G Rust, and David J Adams. Cake: a bioinformatics pipeline for the integrated analysis of somatic variants in cancer genomes. *Bioinformatics*, 29(17):2208–2210, 2013.

[78] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 289–300, 1995.

[79] Bradley Efron. *Large-scale inference: empirical Bayes methods for estimation, testing, and prediction*, volume 1. Cambridge University Press, 2012.

[80] Qingguo Wang, Peilin Jia, Fei Li, Haiquan Chen, Hongbin Ji, Donald Hucks, Kimberly Brown Dahlman, William Pao, Zhongming Zhao, et al. Detecting somatic point mutations in cancer genome sequencing data: a comparison of mutation callers. *Genome Med*, 5(10):91, 2013.

[81] Anne Bruun Krøigård, Mads Thomassen, Anne-Vibeke Lænkholm, Torben A Kruse, and Martin Jakob Larsen. Evaluation of nine somatic variant callers for detection of somatic mutations in exome and targeted deep sequencing data. *PloS one*, 11(3):e0151664, 2016.

[82] Su Yeon Kim, Laurent Jacob, and Terence P Speed. Combining calls from multiple somatic mutation-callers. *BMC bioinformatics*, 15(1):1, 2014.

[83] Christof Angermueller, Tanel Pärnamaa, Leopold Parts, and Oliver Stegle. Deep learning for computational biology. *Molecular Systems Biology*, 12(7):878, 2016.

[84] Zhongming Zhao and Eric Boerwinkle. Neighboring-nucleotide effects on single nucleotide polymorphisms: a study of 2.6 million polymorphisms across the human genome. *Genome research*, 12(11):1679–1686, 2002.

[85] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. *Nature biotechnology*, 2015.

[86] Daniel C Koboldt, Karyn Meltz Steinberg, David E Larson, Richard K Wilson, and Elaine R Mardis. The next-generation sequencing revolution and its impact on genomics. *Cell*, 155(1):27–38, 2013.

[87] Elodie Ghedin, Jennifer Laplante, Jay DePasse, David E Wentworth, Roberto P Santos, Martha L Lepow, Joanne Porter, Kathleen Stellrecht, Xudong Lin, Darwin Operario, et al. Deep sequencing reveals mixed infection with 2009 pandemic influenza a (H1N1) virus strains and the emergence of oseltamivir resistance. *Journal of Infectious Diseases*, 203(2):168–174, 2011.

[88] Nicholas Navin, Alexander Krasnitz, Linda Rodgers, Kerry Cook, Jennifer Meth, Jude Kendall, Michael Riggs, Yvonne Eberling, Jennifer Troge, Vladimir Grubor, et al. Inferring tumor progression from genomic heterogeneity. *Genome research*, 20(1):68–80, 2010.

[89] Daniel J Kvitek and Gavin Sherlock. Whole genome, whole population sequencing reveals that loss of signaling networks is the major adaptive strategy in a constant environment. *PLoS Genet*, 9(11):e1003972, 2013.

[90] Christopher T Saunders, Wendy SW Wong, Sajani Swamy, Jennifer Becq, Lisa J Murray, and R Keira Cheetham. Strelka: accurate somatic small-variant calling from sequenced tumor–normal sample pairs. *Bioinformatics*, 28(14):1811–1817, 2012.

[91] Daniel C Koboldt, Qunyuan Zhang, David E Larson, Dong Shen, Michael D McLellan, Ling Lin, Christopher A Miller, Elaine R Mardis, Li Ding, and Richard K Wilson. Varscan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome research*, 22(3):568–576, 2012.

[92] Dieter Kraft. A software package for sequential quadratic programming. Technical Report DFVLR-FB 88-28, Institut für Dynamik der Flugsysteme, Oberpfaffenhofen, July 1988.

[93] Katy C Kao and Gavin Sherlock. Molecular characterization of clonal interference during adaptive evolution in asexual populations of Saccharomyces cerevisiae. *Nature genetics*, 40(12):1499–1504, 2008.

[94] Mark Johnston. Feasting, fasting and fermenting: glucose sensing in yeast and other cells. *Trends in Genetics*, 15(1):29–33, 1999.

[95] Carl De Boor, Carl De Boor, Carl De Boor, and Carl De Boor. *A practical guide to splines*, volume 27. Springer-Verlag, New York, 1978.

[96] Hachem Saddiki, Jon McAuliffe, and Patrick Flaherty. GLAD: a mixed-membership model for heterogeneous tumor subtype classification. *Bioinformatics*, page btu618, 2014.

[97] Chong Wang and David M Blei. Variational inference in nonconjugate models. *The Journal of Machine Learning Research*, 14(1):1005–1031, 2013.

[98] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[99] Lester W Mackey, David Weiss, and Michael I Jordan. Mixed membership matrix factorization. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 711–718, 2010.

[100] J K Pritchard, M Stephens, and P Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155:945–959, 2000.

[101] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. *J. Mach. Learn. Res.*, pages 1981–2014, September.

[102] Ajit P Singh and Geoffrey J Gordon. A unified view of matrix factorization models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 358–373. Springer, 2008.

[103] D D Lee and H S Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, pages 788–791.

[104] Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development information retrieval*, pages 267–273. ACM, 2003.

[105] YW Teh, MI Jordan, MJ Beal, and DM Blei. Sharing clusters among related groups: Hierarchical Dirichlet processes. In *Proc. Neural Information Processing Systems (NIPS)*, 2005.

[106] Ata Kabán. On Bayesian classification with Laplace priors. *Pattern Recognition Letters*, 28(10):1271–1282, 2007.

[107] David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.

[108] Matt Taddy. Multinomial inverse regression for text analysis. *Journal of the American Statistical Association*, 108(503):755–770, 2013.

[109] David M. Blei and John D. Lafferty. Correlated topic models. In *Proc. Int. Conf. Mach. Learn.*, pages 113–120, 2006.

[110] Han Xiao and Thomas Stibor. Efficient collapsed gibbs sampling for latent Dirichlet allocation. In *ACML*, pages 63–78, 2010.

[111] Christodoulos A. Floudas and C E Gounaris. A review of recent advances in global optimization. *J. Glob. Optim.*, 45:3–38, 2008.

[112] Reiner Horst and Hoang Tuy. *Global optimization: Deterministic approaches*. Springer Science & Business Media, 2013.

[113] Fan Zhang, Chuangqi Wang, Andrew C. Trapp, and Patrick Flaherty. A global optimization algorithm for sparse mixed membership matrix factorization. *arXiv preprint arXiv:1610.06145*, 2016.

[114] Jacques F Benders. Partitioning procedures for solving mixed-variables programming problems. *Numer. Math.*, 4(1):238—-252, 1962.

[115] A. M. Geoffrion. Generalized benders decomposition. *J. Optim. Theory Appl.*, 10:237–260, 1972.

[116] Christodoulos A Floudas. *Deterministic global optimization: theory, methods and applications*, volume 37. Springer Science & Business Media, 2013.

[117] Jochen Gorski, Frank Pfeuffer, and Kathrin Klamroth. Biconvex sets and optimization with biconvex functions: A survey and extensions. *Math. Methods Oper. Res.*, 66:373–407, 2007.

[118] Christodoulos A. Floudas and V Visweswaran. A global optimization algorithm (GOP) for certain classes of nonconvex NLPs. *Comput. Chem. Eng.*, pages 1–34, 1990.

[119] Gurobi Optimization et al. Gurobi optimizer reference manual. *URL: http://www. gurobi. com*, 2:1–3, 2012.

[120] Thomas Zaslavsky. *Facing up to Arrangements: Face-Count Formulas for Partitions of Space by Hyperplanes: Face-count Formulas for Partitions of Space by Hyperplanes*, volume 154. American Mathematical Soc., 1975.

[121] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University press, 2004.

[122] Pei-Wen Chiang, Juan Wang, Yang Chen, Quan Fu, Jing Zhong, Yanhua Chen, Xin Yi, Renhua Wu, Haixue Gan, Yong Shi, et al. Exome sequencing identifies NMNAT1 mutations as a cause of leber congenital amaurosis. *Nature genetics*, 44(9):972–974, 2012.

[123] Elizabeth A Worthey, Alan N Mayer, Grant D Syverson, Daniel Helbling, Benedetta B Bonacci, Brennan Decker, Jaime M Serpe, Trivikram Dasu, Michael R Tschannen, Regan L Veith, et al. Making a definitive diagnosis: successful clinical application of whole exome sequencing in a child with intractable inflammatory bowel disease. *Genetics in Medicine*, 13(3):255–262, 2011.

[124] Anton Nekrutenko and James Taylor. Next-generation sequencing data interpretation: enhancing reproducibility and accessibility. *Nature Reviews Genetics*, 13(9):667–672, 2012.

[125] Benjamin M Neale, Manuel A Rivas, Benjamin F Voight, David Altshuler, Bernie Devlin, Marju Orho-Melander, Sekar Kathiresan, Shaun M Purcell, Kathryn Roeder, and Mark J Daly. Testing for an unusual distribution of rare variants. *PLoS Genet*, 7(3):e1001322, 2011.

[126] Jack A Kosmicki, Claire L Churchhouse, Manuel A Rivas, and Benjamin M Neale. Discovery of rare variants for complex phenotypes. *Human Genetics*, pages 1–10, 2016.

[127] Manuel A Rivas, Mélissa Beaudoin, Agnes Gardet, Christine Stevens, Yashoda Sharma, Clarence K Zhang, Gabrielle Boucher, Stephan Ripke, David Ellinghaus, Noel Burtt, et al. Deep resequencing of GWAS loci identifies independent rare variants associated with inflammatory bowel disease. *Nature genetics*, 43(11):1066–1073, 2011.

[128] Jonathan C Cohen, Robert S Kiss, Alexander Pertsemlidis, Yves L Marcel, Ruth McPherson, and Helen H Hobbs. Multiple rare alleles contribute to low plasma levels of HDL cholesterol. *Science*, 305(5685):869–872, 2004.

[129] Eli M Roth, James M McKenney, Corinne Hanotin, Gaelle Asset, and Evan A Stein. Atorvastatin with or without an antibody to PCSK9 in primary hypercholesterolemia. *New England Journal of Medicine*, 367(20):1891–1900, 2012.

[130] Michael S Lawrence, Petar Stojanov, Paz Polak, Gregory V Kryukov, Kristian Cibulskis, Andrey Sivachenko, Scott L Carter, Chip Stewart, Craig H Mermel, Steven A Roberts, et al. Mutational heterogeneity in cancer and the search for new cancer-associated genes. *Nature*, 499(7457):214–218, 2013.

[131] David J Hand et al. Classifier technology and the illusion of progress. *Statistical science*, 21(1):1–14, 2006.

[132] Cancer Genome Atlas Network et al. Comprehensive molecular portraits of human breast tumours. *Nature*, 490(7418):61–70, 2012.

[133] Joel S Parker, Michael Mullins, Maggie CU Cheang, Samuel Leung, David Voduc, Tammi Vickery, Sherri Davies, Christiane Fauron, Xiaping He, Zhiyuan Hu, et al. Supervised risk predictor of breast cancer based on intrinsic subtypes. *Journal of clinical oncology*, 27(8):1160–1167, 2009.

[134] Peter Lancaster, Miron Tismenetsky, et al. *The theory of matrices: with applications*. Elsevier, 1985.