UNIVERSITY OF CALIFORNIA
RIVERSIDE

Optimization Problems Concerning Tag SNP Selection, Haplotype Inference, and Detection of Horizontal Gene Transfers

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Wei Bung Wang

June 2011

Acknowledgments

It would not have been possible for me to complete this dissertation without the help from many people. I would like to express my heartfelt thanks to all of them.

The first person in my mind I would like to thank is my brilliant advisor Dr. Tao Jiang. His guidance during my graduate study has been always clear, definite, encouraging and stimulating. I always feel his perpetual enthusiasm to science. He ensures my every step on research is solid and rigorous. I have learned incredibly many skills of presentation and writing from him. Under his proficient and continuous revision, every single paper we published was precise and delicate. I got two best paper awards in my graduate study, and I have to give all credit to him.

I would also like to thank Dr. Marek Chrobak. When I was a teaching assistant working with him in winter 2007, I got invaluable advise from him. The teaching experience started to go extremely well. I got an outstanding teaching assistant award in 2008, and I want to give credit to Dr. Chrobak. I got much confidence from this award, and it encouaged me to do well in studies and research since then.

I would like to thank Dr. Stefano Lonardi and Dr. Shizhong Xu for being on my committee. CS234 taught by Dr. Lonardi opened my door to the world of Bioinformatics. As a botanist, Dr. Xu's suggestion is always valuable and ensures my research is on the right track. I also want to thank Dr. Kun-Mao Chao in Taiwan University for initiating my motivation in bioinformatics.

I made many friends in California, who made my life terrific these years. I would extend my gratitude to them for their friendship. I would like to thank my labmates

Guanqun Shi, Lan Liu, Zheng Fu, Qi Fu, Serdar Bozdag, Wei Li, Yu-Ting Huang, my friends Sheng-Feng Wu, I-Chen Peng, Ting-Kai Huang, Weidong Pei, and my best friend Chien-Yao Tseng.

Finally, I wish to thank my family for supporting me. My parents To-Far and Pi-Fen encouraged me to persue the PhD degree. My sister Grace and brother-in-law Kyle bring me good vacations. My lovely wife Xin always supports and comforts me. This dissertation is dedicated to all my family.

ABSTRACT OF THE DISSERTATION

Optimization Problems Concerning Tag SNP Selection, Haplotype Inference, and Detection of Horizontal Gene Transfers

by

Wei Bung Wang

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, June 2011
Professor Tao Jiang, Chairperson

In this dissertation, we study several topics in genetics, including tag SNP selection, haplotype inference, error detection, and horizontal gene transfer detection. We formulate these problems as computational optimization problems, discuss the complexity, present our novel algorithms, and demonstrate the experimental results.

We first study the genome-wide tag SNP selection problem, propose a new model of multi-marker correlation for the problem, and present a greedy algorithm to select a smallest possible set of tag SNPs according to the model. Our experimental results on several real datasets from the HapMap project demonstrate that the new model yields more succinct tag SNP sets than the previous methods.

We then study how to infer haplotypes from genotype data which may contain genotyping errors, *de novo* mutations and missing alleles. We assume that there are no recombinants in the genotype data, which is usually true for tightly linked markers.

We prove the problem is NP-hard, and propose a heuristic algorithm, the core of which is an *integer linear program* (ILP) using the system of linear equations over Galois field $GF(2)$. Our experimental results show that the algorithm can infer haplotypes with a very high accuracy, and recover 65%–94% of genotyping errors depending on the pedigree topology.

We also study the detection of mutations, sequencing errors, and horizontal gene transfers in a set of closely related microbial genomes which do not align well because of rearrangements. We use a new SNP definition to handle the rearrangement problem, divide the problem into several optimization subproblems, and propose a series of algorithms to tackle each subproblem. Results from simulation experiments show that we can detect 31%–61% of horizontal gene transfer events depending on the mutation and missing rates, and the precision of our detection is about 48%–90%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In October 2002, the international HapMap project was launched [72]. One of the main objectives of HapMap project is to identify the haplotype structures of humans and common haplotypes among different populations. A *haplotype* is a combination of alleles at multiple genetic marker (e.g., SNP) loci on the same chromosome. *Single nucleotide polymorphisms* (SNPs) represent the most frequent form of genetic variations in the human genome. They play an important role in genome-wide association studies that intend to help us understand the correlation between genetic variations and human diseases. Assaying (or genotyping) all SNP markers in the involved genomes would be desirable, but it is expensive and unnecessary. Since SNPs are often not independent, a subset of SNPs may be sufficiently informative and allow us to infer all the other SNPs. The *tag SNP selection* problem is thus to find a smallest possible set of tag SNPs that would enable us to infer all the other SNPs with a certain level of confidence [29]. Clearly, the smaller the tag SNP set, the more assaying cost it could help save. In many existing methods, the inference from tag SNPs to all other SNPs is based on the haplotype information [6, 21, 29, 48]. Therefore, the haplotype information of SNP markers is of tremendous value to tag SNP selection, linkage analysis and other genetic analyses (such as gene mapping).

However, in *diploid* organisms like human, chromosomes (other than the sex chromosomes) come in pairs. Each genetic marker on a pair of chromosomes occurs at the same location of both chromosomes. The marker may have different alleles on the two chromosomes. The set of its two alleles is called the *genotype* of the marker and the assignment of the two alleles to the paternal and maternal chromosomes is called the *phase* of the marker. Inferring haplotypes from genotypes over a set of marker loci is called *haplotype inference*, which is also referred to as *phasing*. Because of cost considerations, genotype data instead of haplotype data are routinely collected in practice, especially in large-scale sequencing projects. Since haplotype information gives a more accurate description of the inheritance process than the genotype information and benefits other genetic analyses, efficient and accurate computational methods for haplotype inference have been extensively studied in the literature. See Ref. [42] for a review of these methods as well as the basic concepts involved in haplotype inference.

A SNP disagreement between an organism and its parent could be caused by a mutation or *horizontal gene transfer*. Horizontal gene transfer (HGT) is the process that an organism inherits genetic materials from another organism other than its parents. Bacterial and viral genomes are often affected by HGT and they may contain regions that do not appear in close relatives. Therefore, HGT may play a role in virulence mechanisms, and detection of HGT is valuable for medical microbiology [63], microbial genetics, and bioforensics [58].

In this dissertation, we study three problems: tag SNP selection, haplotype inference on pedigrees, and detection of horizontal gene transfers in bacteria genomes based on their SNP patterns.

1.2 Related Work

1.2.1 Tag SNP selection

Two frameworks for tag SNP selection have been studied in the literature: block-based and genome-wide. The block-based tag SNP selection framework focuses on *haplotype* patterns in a population. The approach assumes that the chromosomes can be partitioned into blocks separated by recombination hotspots, so that there are few recombinations within a block. Then it attempts to identify a smallest possible set of tag SNPs for each block so that all the possible halpotype patterns formed by the SNPs in the block can be fully represented by the haplotype patterns formed by the tag SNPs [57]. The genome-wide framework does not partition a chromosome into blocks. Instead, it considers the correlation between SNP markers across the entire genome [6]. Typically, a SNP marker has two states in a population. The state with a higher frequency is called the *major allele* and the other is called the *minor allele*. In the other words, the SNP markers are usually *bi-allelic*. It is a common practice to consider only SNPs whose *minor allele frequency* (MAF) is at least

5%. Genome-wide tag SNP methods generally follow two approaches. Halldórsson et al. [18] define "informativeness" of SNPs and attempt to find the most informative set of SNPs. The other approach, such as the one adopted by Carlson et al. [6], usually evaluates the *linkage disequilibrium* (LD) between the states of two SNP markers using the correlation coefficient r^2, which indicates the dependency between the two markers, and aims at finding a smallest set of tag SNPs such that all the other SNPs are strongly linked to the selected tag SNPs in terms of the LD coefficient r^2 (more precisely, each of them is linked to some tag SNPs with an r^2 coefficient above a certain threshold). The tag SNPs selected by this approach are shown to be effective in disease association mapping studies, since the coefficient r^2 is directly related to the statistical power of association mapping. Genome-wide tag SNP selection based on the r^2 LD statistics has gained popularity among researchers in the SNP community [6, 9, 26, 48, 61, 83], because it has a comparable performance at a lower computational cost than many other methods [83, 70]. In this thesis, we will be focused on genome-wide tag SNP selection using the r^2 LD statistics.

Most of the existing tag SNP selection methods in this framework consider the r^2 coefficient between a pair of SNP markers [6, 47, 48, 61]. Hence, each of the SNPs is guaranteed to be tagged by a single tag SNP selected. Hao et al. [20, 21] extended the r^2 statistics to describe the statistical correlation between a group of (e.g., two or three) markers and another marker. We will simply refer to this as the *multi-marker correlation model*. In this model, a SNP is tagged by a group of tag SNPs if it is correlated to the group with an r^2 coefficient above a certain threshold. Hao et al. [20, 21] presented a greedy algorithm for selecting tag SNPs to cover a certain (large) fraction of a given set of SNPs and showed that the multi-marker correlation model is more effective than the traditional pairwise correlation model in terms of reducing the number of required tag SNPs.

1.2.2 Haplotype Inference on Pedigrees

Haplotype inference methods can be divided into three groups according to the type of given genotype data: methods for *population* data collected from unrelated individuals [17, 51, 68], methods for *pedigree* data collected from individuals (typically from an extended family) that are related by the parent-child relationship [2, 37, 40, 41, 60, 65, 80, 76, 82], and methods for *pooled samples* [74, 81]. Here, we are interested in only pedigree data.

Some real pedigree data may actually contain mutations. In particular, a *de novo mutation* is a mutation that is present for the first time in a family member as a result of a mutation in a germ cell (egg or sperm) of one of the parents or in the fertilized egg itself. It has been observed that the detection and analysis of mutations in a pedigree could provide a good alternative for some genetic variation research [4, 14, 53].

Most genotype data contain genotyping errors. Genotyping errors have a severe impact on subsequent analyses, such as linkage analysis [1, 11, 38]. Even slight amounts of genotyping error may significantly decrease haplotype frequency and haplotype reconstruction accuracy [34]. Moskvina et al. showed that even with low genotyping error rates (< 0.01), systematic differences in the error rate between samples may result in type I error (i.e., false positive) rates substantially above 0.05 in case-control association studies [50].

Genotyping errors and *de novo* mutations may cause violation of the Mendelian law of inheritance, and hence pedigree data with errors and mutations cannot be properly handled by the above existing haplotype inference methods. When these methods are faced with data containing errors and mutations, they typically delete or report the loci that appear to be inconsistent. For example, Simwalk2 simply reports inconsistencies and terminates [64]. Very few haplotype inference methods in the literature deal with pedigree data that contain mutations and errors (one such method is a genetic algorithm in Ref. [71]). Moreover, detecting genotyping errors is challenging, since these errors do not necessarily violate the Mendelian law of inheritance within nuclear families. In Figure 1.1, all genotype

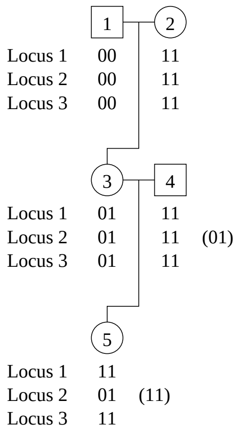


Figure 1.1: An example to show that it is insufficient to consider Mendelian consistency within only nuclear families. The genotypes are consistent within each nuclear family although a genotyping error has occurred. The genotypes in parentheses indicate two possible ways of correcting the error. In a conventional pedigree, a square indicates a male, and a circle indicates a female.

data follow the Mendelian law of inheritance, but it requires two recombinants to explain locus 2 of individual 5. A better explanation is that there is a genotyping error on locus 2 of individual 4 or 5, and alternative genotypes are shown in the parentheses. There have been some work on detecting genotyping errors in the literature [11, 12, 52, 84]. Douglas et al.

can detect 13%–77% of errors [12], and Zou et al. can detect $\leq$ 81% of errors without assuming equal allele frequencies [84]. However, none of these work considers haplotype inference simultaneously.

1.2.3 Detection of Horizontal Gene Transfers in Bacterial Genomes Based on Their SNP Patterns

Phylogenetic trees are commonly used to represent the evolutionary history of a set of extant species in biology. If all organisms only inherit their genetic materials vertically, i.e., from their parents, then the tree representation would be sufficient. However, there is evidence that organisms may get genetic materials from organisms other than their parents [32, 58, 59], and this process is called *horizontal gene transfer* (HGT). A *homologous HGT* is caused by a homologous recombination, in which the incoming DNA molecules are highly similar to those in the recipient genome. Homologous HGT may cause the incongruence between gene trees drawn by different genes, and may lead to inaccuracy of construction of phylogenetic trees [62]. Detection of homologous HGT will help construct a more accurate phylogenetic network [39].

To detect homologous HGT, one approach is to compare the gene trees and the species tree, construct the reconciled tree and detect the HGT (e.g. [23, 56]). These methods do not use the whole-genome information, and do not utilize the gene positional information. Methods based on alignments (e.g. [35, 78]) use the positional information and have a higher accuracy. The main drawback of the alignment approach is the poor scalability when dealing with the whole genomes of dozens of bacterial strains. Most researchers would choose only to align a few target genomes/genes instead of a whole-genome alignment. A small subset of genes may present a high risk in phylogenetic inference if the genes are involved in HGT [62]. If the species tree is drawn by selecting large numbers of characters that are distributed across the genomes, the influence of recombined single genomic regions in tree topology will be diminished, resulting in a tree that reflects the evolutionary history of the majority of the genomes [59] and helps detect the homoplastic[1] SNPs potentially involved in HGT or mutations and sequencing errors.

1.3 Organization

This dissertation is organized as follows. In Chapter 2, we study the tag SNP selection problem. In Chapter 3 and 4, we study the haplotype inference from genotypes on a pedigree. We allow mutations and missing alleles in Chapter 3. In Chapter 4, the problem is more general since we further allow genotyping errors. Both variations of haplotype inference are NP-hard and the proof of NP-hardness is presented in Chapter 4. We study the detection of horizontal gene transfers in Chapter 5. The conclusion is presented in Chapter 6.

[1]For clarification, homology means the similarity due to the common ancestor, and homoplasy means the similarity due to convergent evolution, but independent origins.

Chapter 2

A New Model of Multi-Marker Correlation for Genome-Wide Tag SNP Selection

2.1 Introduction

In this chapter, we generalize the multi-marker correlation model in [20, 21] to further improve its effectiveness. Comparing with the model in [20, 21], our model is more natural and supports more succinct tag SNP sets. We will also present a simple greedy algorithm to select a smallest possible set of tag SNPs according to this multi-marker model, and compare its performance with those of the previous methods on real HapMap data.

Genome-wide tag SNP selection methods can also be classified as *haplotype-based* or *haplotype-independent*, depending on how the r^2 statistics is obtained. For genotype data, the r^2 statistics is usually estimated using a maximum likelihood approach [24, 31], which could be time consuming on a large set of SNPs. However, when phased haplotypes are available, the r^2 coefficients can be calculated very easily and efficiently. The haplotype-based methods require phased haplotype data while the haplotype-independent methods do not. In this work, we will consider both types of data.

The rest of the chapter is organized as follows. In Section 2.2, we introduce a new multi-marker correlation model and discuss how to calculate the r^2 LD statistics under the model for both haplotype and genotype data. Section 2.3 presents the simple greedy algorithm for selecting tag SNPs. In Section 2.4, we discuss the implementation of the algorithm and test its performance on some real HapMap datasets. We also compare the performance of our algorithm with the algorithms on genome-wide tag SNP selection given in [20, 21, 47]. Section 2.5 concludes this chapter with a few remarks. For the ease of reading, we defer a detailed mathematical proof required in the calculation of the multi-marker correlation coefficient r^2 to Section 2.6. This chapter was published in Genome Informatics Vol. 21, pp. 27–41, 2008 [75].

2.2 The New Multi-Marker Correlation Model

In this section, we propose a new multi-marker correlation model that generalizes the model introduced in [20, 21]. We also discuss how to calculate the r^2 statistics under the new model for both haplotype and genotype data.

2.2.1 Multi-Marker Correlation on Haplotype Data

The statistical correlation between a group of k markers and another marker will be referred to as k-marker correlation. For simplicity, we define below the 2-marker correlation model. The generalization of the model to 3 or more markers is straightforward. Consider three bi-allelic SNPs A, B and C. Each of them has possible alleles A/a, B/b and C/c, respectively. Here, the uppercase letters represent both the SNPs as well their major alleles and the lowercase ones represent the minor alleles. Given the states (i.e., alleles) of SNPs A and B, it might be possible for us to infer the state of SNP C, if SNP C is correlated with both SNPs A and B. Clearly, if $Pr(C \mid AB) > 0.5$, we would opt to predict the major allele C instead of the minor allele c when the haplotype AB is observed.

For a fixed population of haplotype data and any haplotype h, let n_h denote the number of times that the haplotype h is observed in the population. Consider three SNPs A, B and C again. For each haplotype $h \in \{AB, Ab, aB, ab\}$, if $n_{hC} > n_{hc}$, then we would opt to predict allele C when observing haplotype h (assuming that the SNP C is unassayed). We put all the haplotypes $h \in \{AB, Ab, aB, ab\}$ such that $n_{hC} > n_{hc}$ into a *major bucket* and the others into a *minor bucket*. For example, if $n_{ABC} > n_{ABc}, n_{abC} > n_{abc}$ and $n_{AbC} < n_{AbC}, n_{aBC} < n_{aBc}$, then the major bucket will contain haplotypes $\{AB, ab\}$ while the minor bucket contains haplotypes $\{Ab, aB\}$. This would suggest a prediction of the allele C when any of the haplotypes $\{AB, ab\}$ in the major bucket is observed.

To define the r^2 correlation coefficient, we introduce a new bi-allelic (compound) marker M that combines the SNPs A and B. The major and minor alleles of M are M/m. We say that the marker M is in state (allele) M if any of the haplotypes in the major bucket is observed, or otherwise it is in state m. Hence, the numbers of observations of alleles M and m are defined as $n_M = n_{AB} + n_{ab}$ and $n_m = n_{Ab} + n_{aB}$. We can define the r^2 statistics between the two markers {A,B} and the marker C as the usual r^2 statistics between the new marker M and the marker C.

Occasionally, we may have a tie between haplotype counts in the population, such as $n_{hC} = n_{hc}$. In this case, we would have to decide whether to put the haplotype h in the major bucket or the minor bucket. The following claim shows that it is usually advantageous to put the haplotype in the minor bucket.

Claim 1 *Consider three SNP markers with alleles A/a, B/b, and C/c, and the correlation coefficient r^2 between the markers* {A, B} *and the marker* C. *If h is an observed haplotype on the markers* A *and* B, *and the numbers of observations satisfy $n_{hC} = n_{hc}$, then putting h in the minor bucket leads to a higher r^2 value most of the time.*

Proof. See Section 2.6. ■

Since there are 4 possible haplotypes on markers A and B, there are $2^4 = 16$ ways to fill the major bucket. After eliminating symmetric ways and the empty set, there are $2^4/2 - 1 = 7$ different ways to separate the 4 possible haplotypes into two buckets. Note that, a split of the four haplotypes like $\{AB, Ab\}/\{aB, ab\}$ really represents the single-marker correlation between markers A and C. Therefore, the seven different separations correspond to two single-marker and five 2-marker correlations.

In [20, 21], Hao et al. proposed a very similar 2-marker correlation model to define the correlation between markers {A,B} and marker C. However, they require that one of the buckets must contain exactly one haplotype (unless the split actually represents a single-marker correlation). For example, a split like $\{AB\}/\{Ab, aB, ab\}$ would be allowed but the split $\{AB, ab\}/\{Ab, aB\}$ is not. Therefore, the 2-marker correlation model in [20, 21] allows a total of $2 + 4 = 6$ different splits, two of which correspond to single-marker correlations. Clearly, our new model is more flexible and gives us the opportunity to cover more SNPs with the same set of tag SNPs. Therefore, it may help reduce the number of tag SNPs required. This flexibility is even more obvious when we consider the correlation between a group of three markers and another marker. To infer a fourth SNP D from three SNPs A, B and C, our model allows $2^{2^3}/2 - 1 = 127$ possible splits of the 8 haplotypes on the SNPs A, B, and C into the major and minor buckets (modulo symmetry). However, because the model of Hao et al. in [20, 21] requires that one of the buckets must contain exactly one haplotype, it only allows $3 + 3 \cdot 4 + 8 = 23$ different splits, including 3 splits corresponding to single-marker correlations and another 12 corresponding to two-marker correlations.

2.2.2 Calculating r^2 Values on Genotype Data

Obtaining r^2 values from haplotype data is trivial. However, if the SNP data is in the form of unphased genotypes, we cannot obtain r^2 values directly since the above definition is based on haplotype data. There are two ways to deal with genotype data. One is to use some haplotype inference program such as PHASE [49, 69] to convert the genotype data into a haplotype data. The other way is to estimate k-marker haplotype frequencies directly from the population without phasing. The former method is trivial. So, here we discuss the latter method.

Hill [24] proposed in 1974 a maximum likelihood method to estimate the degree of

LD between two loci (i.e., markers) given the frequencies of diploid genotypes in a random-mating population. Then he generalized the method to estimate haplotype frequencies at several loci in 1975 [25]. This method has been used to estimate LD r^2 statistics for more than 30 years. For example, it was used in [31] to estimate the LD among multi-allelic markers.

Hill's method works as follows. For simplicity, let us only consider estimating the frequency of 3-marker haplotypes. Consider a sample of population data from N random-mating individuals. Let n_g be the number of times that genotype g is observed in the sample. Denote as f_h the frequency of haplotype h. Let $\hat{f}_h$ be the maximum likelihood estimation of f_h. For three SNPs A, B and C, the frequency of haplotype ABC satisfies the following equation (due to Hardy-Weinberg equilibrium):

$$\begin{aligned}\hat{f}_{ABC} = \frac{1}{2N} \Bigg(& 2n_{AABBCC} + n_{AABBCc} + n_{AABbCC} + n_{AaBBCC} \\ & + n_{AABbCc} \frac{\hat{f}_{ABC}\hat{f}_{Abc}}{\hat{f}_{ABC}\hat{f}_{Abc} + \hat{f}_{ABc}\hat{f}_{AbC}} \\ & + n_{AaBBCc} \frac{\hat{f}_{ABC}\hat{f}_{aBc}}{\hat{f}_{ABC}\hat{f}_{aBc} + \hat{f}_{ABc}\hat{f}_{aBC}} \\ & + n_{AaBbCC} \frac{\hat{f}_{ABC}\hat{f}_{abC}}{\hat{f}_{ABC}\hat{f}_{abC} + \hat{f}_{AbC}\hat{f}_{aBC}} \\ & + n_{AaBbCc} \frac{\hat{f}_{ABC}\hat{f}_{abc}}{\hat{f}_{ABC}\hat{f}_{abc} + \hat{f}_{ABc}\hat{f}_{abC} + \hat{f}_{AbC}\hat{f}_{aBc} + \hat{f}_{Abc}\hat{f}_{aBC}} \Bigg). \end{aligned} \tag{2.1}$$

We can set up equations for the frequencies of the other seven haplotypes on SNPs A, B, and C similarly. Solving these equations can be done by a standard *expectation-maximization* (EM) algorithm [24, 31]. The EM algorithm is iterative. It begins with a random guess of the frequencies. The frequencies obtained at the left hand side in Equation (2.1) will be repeatedly inserted into the right hand side to improve the estimation. When the improvement is sufficiently small (e.g., smaller than a predetermined threshold, typically 10^{-15}), the algorithm terminates and starts a new round with another random guess. After a sufficient number of rounds, it outputs all feasible solutions. We merge the solutions with distances smaller than a threshold (e.g., $\epsilon = 10^{-4}$), and obtain the r^2 value using these estimated 3-marker haplotype frequencies.

There are two things that we have to be careful with when applying Hill's method. The first is that the method assumes the population was produced from random mating and Hardy-Weinberg equilibrium holds. Therefore, datasets consisting of related individuals (such as the CEU dataset in HapMap) would not be suitable. The CEU data consists of family trios, not random-mating individuals. The second is that errors caused by the EM algorithm may lead to wrong assignment of haplotypes into the major and minor buckets. For example, Claim 1 says that when $n_{hC} = n_{hc}$, it is advantageous to assign the haplotype h to the minor bucket instead of the major bucket. However, if $f_{hC} = f_{hc}$ but $\hat{f}_{hC}$ happens to be slightly higher than $\hat{f}_{hc}$ due to some error in the EM computation, we will assign h to the major bucket without caution. This could lead to a reduced r^2 value. To avoid this, we assign h to the minor bucket as long as $\hat{f}_{hC} < \hat{f}_{hc} + \epsilon$ for some small $\epsilon > 0$.

2.3 The Greedy Algorithm for Selecting Tag SNPs

In this section, we first define some notations that will be useful in the algorithm, and then describe the algorithm. For simplicity, we present the algorithm for the 2-marker correlation model first, and then generalize it to work for the multi-marker model. At the end of the section, we analyze the time complexity of the algorithm.

2.3.1 Some Notations

In the rest of this chapter, we call a group of three SNPs, which includes two potential *tagging* SNPs s_i, s_j and one SNP s_k to be tagged, a *triplet* and denote it as $(s_i, s_j \rhd s_k)$. Similarly, a *quartet* is a group of four SNPs including three potential tagging SNPs and SNP to be tagged. The triplets are used in the 2-marker correlation model and the quartets in the 3-marker correlation model. Each such triplet or quartet has a correlation coefficient r^2 value. We will only be interested in triplets and quartets whose correlation coefficient values r^2 are above a certain threshold. It is convenient to think of the triplets or quartets as edges in a hypergraph. Let us regard SNPs as vertices in the hypergraph. The tagging SNPs in a triplet or a quartet have an *outgoing edge* to the SNP to be tagged. This edge can be also thought of as an *incoming edge* of the tagged SNP from the tagging SNPs. Figure 2.1 shows an example hypergraph with five triplets.

During a tag SNP selection process, a SNP has three possible states: *uncovered*, *covered* and *picked*. A SNP is picked if it has been selected as a tag SNP. A SNP s is covered if either s is picked or there is a triplet $(s_i, s_j \rhd s)$ where s_i, s_j are picked. In this case we say that SNPs s_i, s_j *cover* s. A SNP is uncovered if it is not picked nor covered. Sometimes, we may use the term *partially covered*. A SNP s is partially covered if it is uncovered and there is a triplet $(s_i, s_j \rhd s)$ such that either s_i or s_j is picked but not both.

2.3.2 The Algorithm for the 2-Marker Correlation Model

An outline of our algorithm is shown in Figure 2.2. To avoid considering SNPs that cannot possibly be linked, we set a window size of W bps (in terms of the physical distance

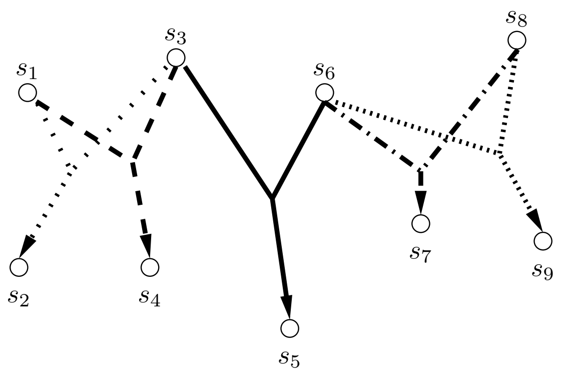


Figure 2.1: An example with five triplets: $(s_1, s_3 \rhd s_2)$, $(s_1, s_3 \rhd s_4)$, $(s_3, s_6 \rhd s_5)$, $(s_6, s_8 \rhd s_7)$ and $(s_6, s_8 \rhd s_9)$.

on a chromosome). For every triplet of SNPs within the window size, we compute its r^2 value as previously described. Then we run an iterative greedy-based algorithm to select a set of tag SNPs as follows. We first initialize all SNPs as uncovered. In each iteration, we pick an appropriate SNP, put it in the tag SNP set, and then check if any uncovered SNPs are now covered due to the newly selected SNP. We repeat this process until all SNPs are covered.

So the main issue is how to pick an appropriate SNP in each iteration. Our first preference is an uncovered SNP that has no incoming edges. A SNP without incoming edges

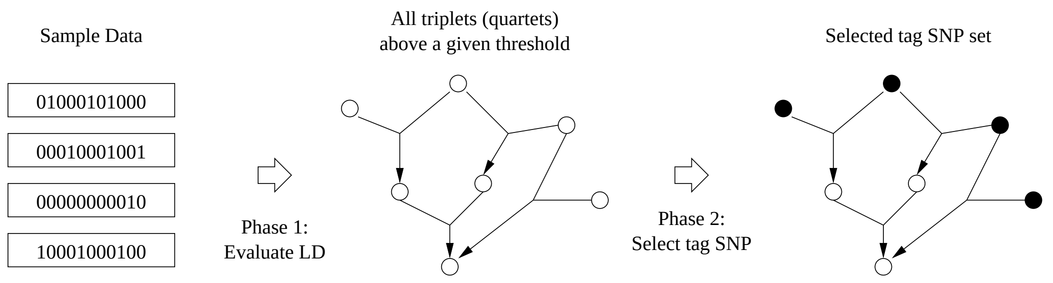


Figure 2.2: An outline of the greedy algorithm.

cannot be tagged by any other SNPs and has to be picked as a tag SNP sooner or later. Therefore, we always check if there is such a SNP. If all SNPs have incoming edges, we pick a SNP (covered or uncovered) that can cover the largest number of uncovered SNPs. If there is a tie, the SNP that partially covers the most uncovered SNPs is preferred. Note that, a covered SNP may also be picked in the above if it covers many other SNPs.

After picking each SNP, we need update and remove some triplets that are no longer useful. A triplet $t = (s_i, s_j \rhd s_k)$ should be removed if any one of the following conditions holds:

1. s_k is covered, and therefore t is useless.

2. s_i and s_j are both picked. In this case, s_i and s_j together tag s_k. After changing the state of s_k to covered, t is no longer useful.

3. There is another triplet $t' = (s_i, s'_j \rhd s_k)$ where s'_j is picked. In this case, the triplet t is superseded by the triplet t' and thus redundant.

Note that, although the condition 3 seems optional and unnecessary, it is actually important since keeping useless triplets in the algorithm may actually affect the final result when useless triplets are involved in the partial coverage of SNPs (and ties have to be broken in the algorithm).

Algorithm 2.3.1 illustrates the pseudocode of the algorithm. In the algorithm, lines 2–5 pick the next SNP. The subsequent lines update the states of the SNPs and remove useless/redundant triplets.

Algorithm 2.3.1 MMTagger(for 2-Marker Model)

Require: set of triplets

```
1:  while there are SNPs uncovered do
2:      if there is a SNP $s$ with no incoming edges then
3:          $s^* \leftarrow s$
4:      else
5:          $s^* \leftarrow$ a SNP that covers the most uncovered SNPs
6:      Put $s^*$ in the tag SNP set        /* $s^*$ is picked        */
7:      for each triplets $t$ of form $(s., s. \rhd s^*)$ do
8:          remove $t$ and its corresponding edges
9:      for each triplets $t$ of form $(s^*, s_i \rhd s_j)$ or $(s_i, s^* \rhd s_j)$ do
10:         if $s_i$ is picked then
11:             put $s_j$ into covered SNP set
12:             remove all triplets of form $(s., s. \rhd s_j)$ or $(s., s. \rhd s_j)$
13:         else
14:             remove all triplets of form $(s_i, s. \rhd s_j)$ or $(s., s_i \rhd s_j)$
```

2.3.3 Extension to the 3-Marker Correlation Model

The extension is straightforward. The outline in Figure 2.2 still works except that we need now calculate r^2 values for quartets. The above greedy algorithm can also be kept the same, although we should modify the removal of useless/redundant quartets slightly. The third condition should be changed to: if there is another quartet $q' = (s_i, s'_j, s'_k \rhd s_l)$ where s'_j, s'_k are picked, then we remove the quartet q.

It is also straightforward to extend the algorithm to the k-marker correlation model, although calculating r^2 values for groups of k SNPs from haplotype data could be very demanding when k is larger than 4, not to mention doing the calculation for genotype data.

2.3.4 Time Complexity

Suppose that there are m SNPs $s_1, s_2, \ldots, s_m$ on a chromosome sorted by their positions. For simplicity, we assume that there are at most w SNPs within each window of W bps. We need compute the r^2 values of all possible triplets involving three SNPs from the same windows. If the first SNP with the smallest index is among $s_1, s_2, \ldots, s_{m-w}$, there will be $\binom{w-1}{2}$ combinations for the second and the third SNPs. If the first SNP is among $s_{m-w+1}, \ldots, s_m$, then there are totally $\binom{w}{3}$ combinations for all three SNPs. The time complexity of computing the r^2 values is therefore $(m-w)\binom{w-1}{2} + \binom{w}{3} = O(mw^2)$. Similarly, the time complexity to compute r^2 values of all possible quartets is $O(mw^3)$.

Assume that there are T triplets with sufficiently high r^2 values. During the selection of tag SNPs, we maintain a data structure where each SNP has two linked-lists to the triplets containing the SNP. One list contains all the triplets corresponding to the outgoing edges and the other contains all the triplets corresponding to the incoming edges. For each SNP, we also keep track of the number of triplets containing the SNP, and various other statistics on these triplets. Therefore, in each iteration of the selection algorithm, we need only scan all the SNPs and use these numbers to pick an appropriate one. To keep the data structure up-to-date, we need update a triplet $t = (s_i, s_j, \triangleright s_k)$ when

1. s_i or s_j is picked;
2. s_k is covered and t needs to be removed; or
3. t is superseded by another triplet and needs to be removed.

If it takes $O(1)$ time to retrieve each triplet that we need update, then the time complexity will be reasonably low. In cases 1 and 2, we can access each of the involved triplets in $O(1)$ time given the data structure. To achieve $O(1)$ access time in case 3, we sort all the triplets in each linked list corresponding to outgoing edges in preprocessing. As a result, if s_i is picked as a tag SNP, then $(s_i, s_j \triangleright s_k)$ will supersede all triplets of the form $(s_h, s_j \triangleright s_k)$ for some h. These triplets $(s_h, s_j \triangleright s_k)$ must be neighbors of $(s_i, s_j \triangleright s_k)$ on s_j's outgoing linked list. Therefore, we can access to each of these triplets in $O(1)$ time. Since a triplet may be updated at most 3 times, the time to select tag SNPs is $O(T)$. The preprocessing may take $O(T \log T)$ time.

In practice, the algorithm spends most of its time on evaluating r^2 values. Therefore, we say that the time complexity of the algorithm is $O(mw^2)$ (or $O(mw^3)$) for the 2-marker correlation (or 3-marker correlation) models, respectively.

2.4 Experimental Result

We have implemented the above algorithm as a C program, simply called MMTagger. In this section, we compare MMTagger with the program LRTag in [47] and the program MultiTag in [21] on real datasets from the HapMap project. The following is a brief summary of the features of the three programs to be compared.

- LRTag [47] uses the traditional single-marker correlation model and works for a single population as well as multiple populations. The algorithm is based on a powerful combinatorial optimization technique called *Lagrangian relaxation.* According to the extensive tests in [47], LRTag outperforms other state-of-the-art single-marker programs such as FESTA [61] and LD-Select [6] in terms of the number of selected tag SNPs. It requires the pairwise r^2 statistics as the input.
- MultiTag [21] uses a multi-marker correlation model which is more restricted than our model. It is a greedy algorithm. The input to MultiTag must be a population haplotype data.
- MMTagger is a greedy algorithm using a more general multi-marker correlation model. Its input is a population data, either in the form of haplotypes or genotypes.

In order to compare these three programs, we need phased haplotype data. We downloaded the CEU ENCODE region data from the HapMap project[1] and use the first 5 of the 10 sample datasets. For LRTag, we need a preprocessing step to calculate the pairwise r^2 values. For both MMTagger and MultiTag, we use a window size W of 100K

[1]http://www.hapmap.org/downloads/phasing/2005-03_phaseI/ENCODE/

bps so that SNPs farther than W bps apart are not considered as correlated. To make it fair, we also apply this restriction when calculating r^2 values for LRTag.

Table 2.1 shows the numbers of the tag SNPs selected by LRTag, MultiTag and MMTagger using different parameters. The reduction of tag SNPs by using the multi-marker correlation models is obvious. However, the running time of the programs based on the multi-marker correlation models (MultiTag and MMTagger) is much longer. LRTag requires only pairwise r^2 values, but MultiTag and MMTagger need r^2 values for each group of three or four SNPs. In general, MMTagger selected fewer tag SNPs than MultiTag. In fact, the improvement is quite significant when the threshold for r^2 is 0.9 or larger.

When comparing the performance of MultiTag and MMTagger, we should also take into account the running time and memory usage. We thus downloaded the entire chromosomal data of the Japanese and Chinese populations from HapMap[2] and used chromosomes 19, 21 and 22 as our test data.

Hao [21] mentioned two different methods to implement his greedy algorithm and handle a large number of input SNPs: (1) Preprocess and compute all r^2 values, and (2) Calculate r^2 values on the fly while selecting tag SNPs. The former method would lead to heavy memory load and/or file I/O load. The latter method may lead to redundant r^2 value computation. MultiTag employs the latter method. In our implementation of MMTagger, we choose the former method to speed up the computation.

Table 2.2 illustrates a head-to-head comparison between MultiTag and MMTagger. Note that, for the memory usage, we were able to insert some code into MMTagger to obtain

[2]http://www.hapmap.org/downloads/phasing/2006-07_phaseII/phased/

Region	ENm010	ENm013	ENm014	ENr112	ENr113
# SNP	459	731	874	868	1035
$r^2 \geq 0.8$					
LRTag	119	88	134	148	133
2-marker MultiTag	75	57	80	87	75
2-marker MMTagger	72	52	78	85	73
3-marker MultiTag	68	53	75	78	64
3-marker MMTagger	62	48	75	68	59
$r^2 \geq 0.9$					
LRTag	148	121	172	204	190
2-marker MultiTag	100	76	111	118	122
2-marker MMTagger	92	73	100	109	115
3-marker MultiTag	91	66	102	101	100
3-marker MMTagger	79	58	85	81	81
$r^2 \geq 0.95$					
LRTag	192	148	196	268	247
2-marker MultiTag	127	96	131	157	156
2-marker MMTagger	117	92	122	141	149
3-marker MultiTag	120	83	119	138	145
3-marker MMTagger	97	66	102	107	112

Table 2.1: Numbers of tag SNPs selected in CEU ENCODE region

Chromosome	# SNP	mode	r^2	program	# SNPs Selected	Time (hours)	Memory (M bytes)
JPT+CHB chr19	28931	2-marker	0.9	MultiTag	9600	26hrs	30–35
				MMTagger	9145	2mins	125
		3-marker	0.95	MultiTag	N/A	>700hrs	30–35
				MMTagger	10032	<1hr	657
JPT+CHB chr21	28914	2-marker	0.9	MultiTag	7115	42hrs	30–35
				MMTagger	6766	2mins	187
		3-marker	0.95	MultiTag	N/A	>700hrs	30–35
				MMTagger	7404	<1hr	1210
JPT+CHB chr22	26595	2-marker	0.9	MultiTag	7557	93hrs	30–35
				MMTagger	7221	2mins	183
		3-marker	0.95	MultiTag	N/A	>700hrs	30–35
				MMTagger	7788	3hrs	1216

Table 2.2: MMTagger vs. MultiTag

the precise maximum memory used by the program. However, we were not able to get the precise memory usage numbers for MultiTag and could only provide a rough estimate. The following gives a detailed comparison between the two programs.

- MMTagger is able to achieve a smaller tag SNP set than MultiTag mostly because our multi-marker correlation model is more general and flexible.

- MMTagger's heuristic to always pick uncovered SNPs with no incoming edges first may also be a factor in its improved performance. This heuristic can be easily incorporated into MultiTag.

- MMTagger may pick a SNP that has been covered if it covers many other SNPs. However, MultiTag always picks an uncovered SNP. Modifying MultiTag to allow covered SNPs to be picked would cost its more time since it calculates r^2 values on the fly. However, this does not impact the running time of MMTagger much because it pre-calculates all r^2 values.

- MMTagger is much faster than MultiTag. Its running time mostly depends on the window size W, since it spends most time on calculating the r^2 values. The running time of MultiTag depends on both the window size W and the number of tag SNPs selected. Hence, it requires more time for higher r^2 thresholds since more tag SNPs would be required. Hao [21] reported that the program took about 300 hours to process the human chromosome 2 data on a typical workstation (Intel Xeon 2.80 GHz CPU and 512 MB memory).

- MMTagger requires much more memory. Its memory usage grows when the r^2 threshold decreases, as more triplets/quartets would be qualified. To run the program on a large chromosome such as human chromosome 2, it require about 4 GB of memory for the 3-marker correlation model when the r^2 threshold is 0.9. However, MultiTag's memory usage is pretty reasonable even for large chromosomes and low r^2 thresholds.

- MMTagger and MultiTag use the window size W in slightly different ways. MMTagger requires that all SNPs in a triplet/quartet should be in the same window, while MultiTag requires that a covered SNP and each of its tagging SNPs should not be farther than W. Therefore, the distance of the two tagging SNPs of a triplet may actually be as far as $2W$ in MultiTag.

As observed before, the 2-marker correlation model improves on the single-marker correlation model significantly. A similar significant improvement from the 2-marker model to the 3-marker model is also shown in Table 2.2. Although it is likely that the 4-marker model will show further improvements, we are not able to extend the results to the 4-marker model because MMTagger would require too much time and memory on any realistic datasets. For the same reason, MultiTag was only implemented for the 2-marker and 3-marker models in [20, 21]

2.5 Conclusion

We have introduced a new multi-marker correlation model that generalizes a previous result in the literature. A greedy algorithm is designed to select tag SNPs based on the model. Our experimental results on real datasets from the HapMap project demonstrate that the algorithm produces the most succinct tag SNP sets compared with the previous algorithms.

2.6 The Proof of Claim 1

Let us consider the frequency table as shown in Table 2.3, where A is a SNP to be covered/tagged and M is a compound marker representing several (e.g., two or three) SNPs. Let n_{AM} denote the number of times that the haplotype AM is observed in the population, $n_A = n_{AM} + n_{Am}$, and n the total number of haplotypes.

	A	a	
M	n_{AM}	n_{aM}	n_M
m	n_{Am}	n_{am}	n_m
	n_A	n_a	n

Table 2.3: Number of observations of each haplotype

For any haplotype h on M, if $n_{Ah} > n_{ah}$, we would put h in the major bucket, otherwise we put it in the minor bucket. However, when $n_{Ah} = n_{ah}$, it seems that we could put h in either the major bucket or the minor bucket. We show in the following that putting h in the minor bucket leads to a bigger r^2 value between M and A. By definition of the r^2

statistics,

$$\begin{aligned}
r^2 &= \frac{(p_{AM} - p_A p_M)^2}{p_A p_a p_M p_m} \\
&= \frac{(n_{AM} \cdot n - n_A n_M)^2}{n_A n_a n_M n_m} \\
&= \frac{(n_{AM} n_{am} - n_{Am} n_{aM})^2}{(n_{AM} + n_{Am})(n_{aM} + n_{am})(n_{AM} + n_{aM})(n_{Am} + n_{am})}
\end{aligned}$$

We take the partial derivative of r^2 with respect to n_{AM} and obtain

$$\begin{aligned}
\frac{\partial r^2}{\partial n_{AM}} &= \frac{(n_{AM} n_{am} - n_{Am} n_{aM})}{n_A n_a n_M n_m} \cdot \\
&\quad \left(2n_{am} - \frac{(n_{AM} n_{am} - n_{Am} n_{aM})(2n_{AM} + n_{Am} + n_{aM})}{(n_{AM} + n_{Am})(n_{AM} + n_{aM})}\right)
\end{aligned}$$

By simplifying the equation, we get

$$\begin{aligned}
\frac{\partial r^2}{\partial n_{AM}} &= c\left(2n_{am} - \frac{X(n_A + n_M)}{n_A n_M}\right) \\
\frac{\partial r^2}{\partial n_{Am}} &= c\left(-2n_{aM} - \frac{X(n_A + n_m)}{n_A n_m}\right) \\
\frac{\partial r^2}{\partial n_{aM}} &= c\left(-2n_{Am} - \frac{X(n_a + n_M)}{n_a n_M}\right) \\
\frac{\partial r^2}{\partial n_{am}} &= c\left(2n_{AM} - \frac{X(n_a + n_m)}{n_a n_m}\right)
\end{aligned}$$

where $c = \frac{(n_{AM} n_{am} - n_{Am} n_{aM})}{n_A n_a n_M n_m}$, $X = (n_{AM} n_{am} - n_{Am} n_{aM})$.

Suppose that $n_{Ah} = n_{ah}$. If we put haplotype h in the major bucket, then the r^2 value would change by approximately $n_{Ah} \cdot \frac{\partial r^2}{\partial n_{AM}} + n_{ah} \cdot \frac{\partial r^2}{\partial n_{aM}}$. If we put h in the minor bucket, then the r^2 value would change by approximately $n_{Ah} \cdot \frac{\partial r^2}{\partial n_{Am}} + n_{ah} \cdot \frac{\partial r^2}{\partial n_{am}}$. Let

$$\begin{aligned}
\Delta_M &= \frac{\partial r^2}{\partial n_{AM}} + \frac{\partial r^2}{\partial n_{aM}} \\
&= c\left(2n_{am} - 2n_{Am} - X\left(\frac{1}{n_A} + \frac{1}{n_a} + \frac{2}{n_M}\right)\right) \\
\Delta_m &= \frac{\partial r^2}{\partial n_{Am}} + \frac{\partial r^2}{\partial n_{am}} \\
&= c\left(2n_{AM} - 2n_{aM} - X\left(\frac{1}{n_A} + \frac{1}{n_a} + \frac{2}{n_m}\right)\right)
\end{aligned}$$

We have

$$\begin{aligned}
\Delta_m - \Delta_M &= 2c(n_{AM} - n_{aM} + n_{Am} - n_{am}) + cX\left(\frac{2}{n_M} - \frac{2}{n_m}\right) \\
&= 2c(n_A - n_a) + 2cX\left(\frac{1}{n_M} - \frac{1}{n_m}\right)
\end{aligned}$$

We need check if $\Delta_m - \Delta_M \geq 0$ holds. By multiplying both side with $\frac{n_M n_m}{2c}$ we get

$$\begin{aligned}
&\frac{1}{2c} n_M n_m (\Delta_m - \Delta_M) \\
=\ & (n_A - n_a) n_M n_m + (n_{AM} n_{am} - n_{Am} n_{aM})(n_m - n_M) \\
=\ & (n_{AM} + n_{Am} - n_{aM} - n_{am})(n_{AM} + n_{aM})(n_{Am} + n_{am}) \\
& \quad + (n_{AM} n_{am} - n_{Am} n_{aM})(n_{Am} + n_{am} - n_{AM} - n_{aM}) \\
=\ & n_{AM}(n_{AM} + n_{aM}) n_{Am} + n_{Am} n_{AM}(n_{Am} + n_{am}) \\
& \quad - n_{aM}(n_{AM} + n_{aM}) n_{am} - n_{am} n_{aM}(n_{Am} + n_{am}) \\
=\ & n_{AM} n_{Am} \cdot n - n_{aM} n_{am} \\
=\ & n(n_{AM} n_{Am} - n_{aM} n_{am})
\end{aligned}$$

where $n = n_{AM} + n_{Am} + n_{aM} + n_{am}$. Therefore, $\Delta_m \geq \Delta_M$ if and only if $n_{AM} n_{Am} \geq n_{aM} n_{am}$.

When the latter inequality holds, putting the haplotype h in the minor bucket will result in a higher r^2 value.

Since $n_{AM} + n_{Am} = n_A > n_a = n_{aM} + n_{am}$, $n_{AM}n_{Am}$ tends to be greater than $n_{aM}n_{am}$ in practice. Moveover, even when $n_{AM}n_{Am} < n_{aM}n_{am}$, putting the haplotype h in the minor bucket would increase n_{Am} and n_{am} at the same time, and hence result in a greater increase in $n_{AM}n_{Am}$ than in $n_{aM}n_{am}$ since n_{AM} is usually larger than n_{aM}. This could help improve the r^2 value in the long run. Therefore, putting h in the minor bucket may still be better in this case. For example, suppose $n_{AM} = 100$, $n_{Am} = 0$, $n_{aM} = 5$, and $n_{am} = 20$ before haplotype h is considered. If $n_{Ah} = n_{ah} = 1$, then putting h in the major (or minor) bucket results in $r^2 = 0.7261$ (or $r^2 = 0.7235$, respectively). However, if $n_{Ah} = n_{ah} = 3$, then putting h in the major (or minor) bucket leads to $r^2 = 0.6628$ (or $r^2 = 0.6631$, respectively).

Note that, the tag SNP selection program MultiTag in [20, 21] considers all the possible splits of the haplotypes in question and picks the one that results in the highest r^2 value. So, ties between haplotype counts are not an issue. However, we cannot afford doing this in our tag SNP selection program MMTagger (to be introduced in Section 2.4) because our multi-marker correlation model allows for many more possible splits. Trying all such splits would be very inefficient. Since the above analysis shows that putting haplotype h in the minor bucket is generally better when we have a tie $n_{Ah} = n_{ah}$, MMTagger always puts h in the minor bucket when such a tie arises. $\square$

Chapter 3

Inferring Haplotypes from Genotypes on a Pedigree with Mutations and Missing Alleles

3.1 Introduction

In this chapter, we study haplotype inference on pedigree data on tightly linked markers that have no recombinants but may contain a small number of *de novo* mutations (or simply, mutations). Since mutation is a rare event, we formulate the problem as a combinatorial optimization problem, called the *minimum mutation haplotype configuration* (MMHC) problem, where we look for a haplotype solution consistent with the given genotype data that incur no recombinants and require the minimum number of mutations. Our hypothesis is a solution with the minimum number of mutations is likely the true solution.

Moreover, we are only interested in solutions where each locus has at most one mutation in the pedigree. This restriction is reasonable given Kimura's infinite-site model et al. [33] which suggests that the probability of multiple mutations at the same locus is low enough to be negligible. This extends the well studied *zero-recombinant haplotype configuration* (ZRHC) problem where we try to find a consistent haplotype solution incurring no recombinants or mutations. Although ZRHC is polynomial-time solvable [40], we can prove that MMHC is NP-hard by a reduction from NAE-3SAT (the proof is in Chapter 4, Section 4.5). We construct an *integer linear program* (ILP) for MMHC using the system of linear equations over the field $GF(2)$ that has been developed in [40, 45, 80] for solving ZRHC in almost linear time. Since the number of constraints in the ILP is quite large (exponentially large in general) when the input pedigree is large, we present an incremental approach for solving the ILP.

An outline of our incremental approach is as follows. Given a pedigree data, we set up a system of linear equations over $GF(2)$ introduced in [45, 80] for ZRHC, but conditional on mutations. We convert the linear system to an ILP instance for MMHC where the constraints generally describe the relation between the equations and mutations. A small set of the constraints in the ILP are identified as the *core* constraints, and a standard ILP solver GLPK (the GNU Linear Programming Kit from `http://www.gnu.org/softward/glpk`) is invoked on the partial ILP instance with only the core constraints. The ILP solution describes an assignment of mutations in the pedigree which can be used to remove the conditions in the linear system. By using Gaussian elimination, we can check if the linear system is consistent. If it is consistent, a haplotype configuration (with the minimum number of mutations) is returned. Otherwise, we find the inconsistent equations and add some new constraints to the core to force their consistency. This process is repeated until an ILP solution that satisfies its corresponding linear system has been found. Note that, the incremental approach to solving the ILP is crucial here because the ILP instance cannot be efficiently and explicitly constructed as its number of constraints grows exponentially in the pedigree size in general. Also note that, with the advance in sequencing technology, larger and larger pedigrees are being genotyped and analyzed in practice. For example, in [3, 5], haplotype inference was performed on pedigrees of sizes 368 and 1149, respectively.

We have implemented the algorithm and tested it on pedigree data that were simulated with random mutations and missing alleles. (Real pedigree data often have up to 20% missing alleles.) The experimental results demonstrate that our method can infer haplotypes with a very high accuracy. It can also detect most of the mutations and impute most of the missing alleles correctly. Moreover, it is found that the algorithm usually terminates after a small number of iterations without ever having to invoking ILP solver on the complete ILP instance consisting of all the constraints. As a comparison, we have also considered the straightforward approach for solving the ILP with all the constraints considered at once on binary tree pedigrees (i.e., each pair of parents has only one child). The ILP instance can be efficiently constructed for binary tree pedigrees. It is found that our algorithm is much faster than the straightforward approach.

The rest of this chapter is organized as follows. In Section 3.2, we incorporate mutations into the system of linear equations introduced in [45, 80] for ZRHC to obtain a system of conditional linear equations for MMHC. Section 3.3 describes the ILP formulation for MMHC, and the incremental approach for solving the ILP. In Section 3.4, we discuss the implementation of the algorithm and test its performance on some simulated pedigree data with random mutations and missing alleles. Section 3.5 concludes this chapter with a few remarks. This chapter appeared in the procedings of 20th Annual Symposium on Combinatorial Pattern Matching 2009, pp. 353–367 [76].

3.2 A System of Conditional Linear Equations for MMHC

We review the system of linear equations over $GF(2)$ introduced in [45, 80] for solving ZRHC and extend the system to take into account mutations.

3.2.1 The Linear System

Let n denote the number of the individuals in the input pedigree and m the number of marker loci of each individual. For simplicity, we assume in this chapter that all alleles are bi-allelic (denoted as 0 or 1) and the input pedigree is free of mating loops (and thus a tree pedigrees). Tree pedigrees are very common among human pedigrees. Our techniques can be extended to general pedigrees. The genotype of individual j is denoted as a ternary vector $\mathbf{g}_j$ whose kth entry $g_j[k]$ represents the genotype at locus k of individual j as follows:

$$\begin{cases} g_j[k] = 0 & \text{if both alleles are 0's} \\ g_j[k] = 1 & \text{if both alleles are 1's} \\ g_j[k] = 2 & \text{if the locus is heterozygous} \end{cases} \tag{3.1}$$

The value of $g_j[k]$ is *unknown* if the alleles are missing. For each locus k of individual j, we define a binary variable $p_j[k]$ over $GF(2)$ to indicate the paternal allele at the locus:

$$\begin{cases} p_j[k] = 0 & \text{if } g_j[k] = 0 \\ p_j[k] = 1 & \text{if } g_j[k] = 1 \\ p_j[k] = 0 & \text{if } g_j[k] = 2 \text{ and allele 0 is paternal} \\ p_j[k] = 1 & \text{if } g_j[k] = 2 \text{ and allele 1 is paternal} \end{cases} \tag{3.2}$$

In other words, the binary vector $\mathbf{p}_j$ represents the paternal haplotype of individual j. To represent the maternal haplotype, we need another binary vector $\mathbf{w}_j$ to indicate if each locus of individual j is heterozygous. That is, $w_j[k] = 0$ if $g_j[k] = 0$ or 1, and $w_j[k] = 1$ if $g_j[k] = 2$. Clearly, the sum $\mathbf{p}_j + \mathbf{w}_j$ (over $GF(2)$) represents the maternal haplotype of individual j.

Suppose that individual i is a parent of individual j. To unify the representation of the haplotype that j inherited from i, define a binary vector $\mathbf{d}_{i,j}$ as follows: $\mathbf{d}_{i,j} = 0$ if i is j's father and $\mathbf{d}_{i,j} = \mathbf{w}_j$ if i is j's mother. Therefore, $\mathbf{p}_j + \mathbf{d}_{i,j}$ represents the haplotype that j got from i. Define $h_{i,j} = 0$ if $\mathbf{p}_j + \mathbf{d}_{i,j}$ is i's paternal haplotype and $h_{i,j} = 1$ otherwise. Then $\mathbf{p}_i + h_{i,j} \cdot \mathbf{w}_i$ represents the haplotype that i passed to j. The binary variables $h_{i,j}$ thus fully describe the inheritance pattern in an ZRHC instance. Finally, define $\mu_{i,j}[k] = 1$ if the there is a mutation at locus k when i passes the haplotype $\mathbf{p}_i + h_{i,j} \cdot \mathbf{w}_i$ to j, and $\mu_{i,j}[k] = 0$ otherwise. For technical reasons, we view $\mu_{i,j}[k]$ as an integer from $\mathbb{Z}$ instead of $GF(2)$. For convenience, we make these three vectors symmetric by defining $\mathbf{d}_{j,i} = \mathbf{d}_{i,j}$, $h_{j,i} = h_{i,j}$, and $\mu_{j,i} = \mu_{i,j}$. Using these notations, we can derive a *conditional equation* over

$GF(2)$:

$$\begin{cases} p_i[k] + h_{i,j} \cdot w_i[k] = p_j[k] + d_{i,j}[k] & \text{if } \mu_{i,j}[k] = 0 \\ p_i[k] + h_{i,j} \cdot w_i[k] = p_j[k] + d_{i,j}[k] + 1 & \text{if } \mu_{i,j}[k] = 1 \end{cases} \tag{3.3}$$

Since we assume that each locus has at most one mutation in the pedigree,

$$0 \leq \sum_{i,j} \mu_{i,j}[k] \leq 1 \quad \forall k \tag{3.4}$$

Note that the summation is over $\mathbb{Z}$ instead of $GF(2)$. Hence,the MMHC problem can be formally defined as follows. Given an input pedigree and genotype data $\mathbf{g}_j$ for each individual j, find a solution to each $\mathbf{p}_j$, $h_{i,j}$ and $\mu_{i,j}$ that satisfies all the (conditional) constraints in Equations (3.3) and (3.4) and minimizes the sum $\sum_{i,j,k} \mu_{i,j}[k]$.

3.2.2 Pre-Determined Variables

The above linear system has $O(mn)$ variables and equations. As in [45, 80], we can convert the system to an equivalent linear system involving only the h-variables which is much smaller (there are only $O(n)$ h-variables). This requires us to *pre-determine* the values of some p-variables. The situation is complicated a little bit by the presence of the μ-variables.

Let us consider a p-variable $p_j[k]$ where the marker of individual j at locus k is not missing, and several scenarios.

1. $g_j[k] \neq 2$. By Equation (3.2), $p_j[k] = g_j[k]$. In this case, $p_j[k]$ is *pre-determined.* We will refer to $p_j[k]$ as the *intended p-value* of the locus, denoted as $v(j,k) = p_j[k]$.

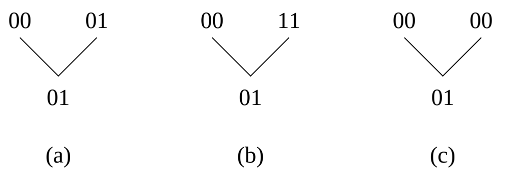


Figure 3.1: Determining a p-variable. Consider the p-value of the child in the trio. (a) It equals 0 as long as there is no mutation from the father and it is semi-determined. (b) It equals 0 and there cannot be any mutation. It is pre-determined. (c) It is undetermined but there must be a mutation. It is doubly-determined.

2. $g_j[k] = 2$ and exactly one parent, denoted as i, is homozygous at locus k. See Figure 3.1(a). We have $w_i[k] = 0$ by definition. According to Equation (3.3), $p_j[k]$ is known if and only if $\mu_{i,j}[k]$ is known. We say that $p_j[k]$ is *semi-determined* in this case. We also define $\mu_{i,j}[k]$ as the *anchor* of $p_j[k]$ and denote $a(j,k) = \{\mu_{i,j}[k]\}$. Since the value of $p_j[k]$ on the condition $\mu_{i,j}[k] = 0$ is preferred, we denote $v(j,k) = g_i[k]+d_{i,j}[k]$.

3. $g_j[k] = 2$, both parents i_1 and i_2 of j are homozygous at locus k, and $g_{i_1}[k] \neq g_{i_2}[k]$. See Figure 3.1(b). Since each locus has at most one mutation, $\mu_{i_1,j}[k]$ and $\mu_{i_2,j}[k]$ cannot both be 1. Hence, $\mu_{i_1,j}[k] = \mu_{i_2,j}[k] = 0$. In this case, $p_j[k]$ is pre-determined, and we denote $v(j,k) = p_{i_1}[k] + d_{i_1,j}[k]$.

4. $g_j[k] = 2$, both parents i_1 and i_2 are homozygous at locus k, and $g_{i_1}[k] = g_{i_2}[k]$. See Figure 3.1(c). In this case, one of $\mu_{i_1,j}[k]$ and $\mu_{i_2,j}[k]$ equals 1 and the other 0. Thus, $p_j[k]$ has two anchors: $a_1(j,k) = \{\mu_{i_1,j}[k]\}$ and $a_2(j,k) = \{\mu_{i_2,j}[k]\}$. Each anchor gives rise to a preferred value for $p_j[k]$, $v_1(j,k) = p_{i_1}[k]+d_{i_1,j}[k]$ and $v_2(j,k) = p_{i_2}[k] + d_{i_2,j}[k]$, respectively. In this case we call $p_j[k]$ *doubly-determined.*

5. All other cases. The variable $p_j[k]$ is *undetermined* and the variable $v(j,k)$ is unde-

fined.

If a $p_j[k]$ is pre-determined or undetermined, we define $a(j,k) = \emptyset$. Similarly, we might be able to pre-determine μ-variable $\mu_{i,j}[k]$ in some cases.

1. $g_i[k] = g_j[k] \neq 2$. Since the top equation in Equation (3.3) holds, we let $\mu_{i,j}[k] = 0$ and it is pre-determined.

2. $g_i[k] \neq g_j[k]$ and both loci are homozygous. Since the bottom equation in Equation (3.3) holds, we set $\mu_{i,j}[k] = 1$. This μ-variable is pre-determined. Moreover, all the other μ-variables at locus k must equal 0 and are pre-determined too.

3. Some p-variable at locus k is doubly determined. All the μ-variables at locus k other than this p-variable's anchors must equal 0 and are thus pre-determined.

4. All other cases. The variable $\mu_{i,j}[k]$ stays undetermined.

3.2.3 A More Compact Linear System

Following [45, 80], we can set up a linear system in terms of the h-variables. The idea is to consider paths in the pedigree connecting individuals with pre/semi/doubly-determined p-variables and derive (conditional) equality constraints on the h-variables on such paths based on Equation (3.3).

Consider a locus k and a path $j_0, j_1, \ldots, j_r$ in the input (tree) pedigree, where individuals j_i and j_{i+1} have the parent-child relationship. Suppose that $p_{j_0}[k]$ and $p_{j_r}[k]$ are pre-determined, semi-determined or doubly-determined, and $g_{j_1}[k] = \cdots = g_{j_{r-1}}[k] = 2$. We call the path $j_0, j_1, \ldots, j_r$ an *all-heterozygous path* at locus k. If $p_{j_0}[k]$ and $p_{j_r}[k]$ are pre-determined or semi-determined, we define a *path constraint* connecting j_0 and j_r:

$$v(j_0,k)+v(j_r,k)+\sum_{i=0}^{r-1}\left(h_{j_i,j_{i+1}}+d_{j_i,j_{i+1}}[k]\right)=0$$

$$\text{if all elements in } a(j_0,k)\cup a(j_r,k)\cup\bigcup_{i=0}^{r-1}\{\mu_{j_i,j_{i+1}}[k]\} \text{ equal } 0 \tag{3.5}$$

If we denote $\mathcal{M}=a(j_0,k)\cup a(j_r,k)\cup\bigcup_{i=0}^{r-1}\{\mu_{j_i,j_{i+1}}[k]\}$, $\mathcal{H}=\bigcup_{i=0}^{r-1}\{h_{j_i,j_{i+1}}\}$, and $c=v(j_0,k)+v(j_r,k)+\sum_{i=0}^{r-1}d_{j_i,j_{i+1}}[k]$, then the path constraint can also be represented by the triple $(\mathcal{H},\mathcal{M},c)$ which denotes:

$$\sum_{h_{i,j}\in\mathcal{H}}h_{i,j}=c \qquad \text{iff } \mu_{i,j}[k]=0 \quad \forall\mu_{i,j}[k]\in\mathcal{M} \tag{3.6}$$

If j_0 or j_r is doubly-determined, we can construct two path constraints in the same way: one using $v_1(\cdot)$ and $a_1(\cdot)$ and the other using $v_2(\cdot)$ and $a_2(\cdot)$.

Consider a *local cycle* consisting of father i_1, mother i_2, and two adjacent children j_1, j_2. If both parents are heterozygous at locus k, we can obtain four conditional equations from Equation (3.3) by replacing i with i_1, i_2, and j with j_1, j_2. (See Figure 3.2.) The

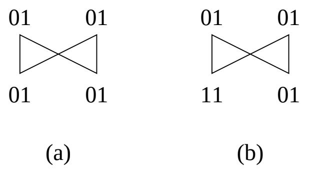


Figure 3.2: Two possible cycle constraints from a local cycle. (a) The sum of the four h-variables is 0. (b) The sum of the four h-variables is 1.

summation of these conditional equations forms a *cycle constraint*:

$$\begin{aligned} & h_{i_1,j_1} + h_{i_1,j_2} + h_{i_2,j_1} + h_{i_2,j_2} \\ & \quad = d_{i_1,j_1}[k] + d_{i_1,j_2}[k] + d_{i_2,j_1}[k] + d_{i_2,j_2}[k] \\ & \quad = w_{j_1}[k] + w_{j_2}[k] \\ & \qquad \text{iff } \mu_{i_1,j_1}[k] = \mu_{i_1,j_2}[k] = \mu_{i_2,j_1}[k] = \mu_{i_2,j_2}[k] = 0 \end{aligned} \tag{3.7}$$

This constraint will also be denoted as $(\mathcal{H}, \mathcal{M}, c)$ where $\mathcal{H} = \{h_{i_1,j_1}, h_{i_1,j_2}, h_{i_2,j_1}, h_{i_2,j_2}\}$, $\mathcal{M} = \{\mu_{i_1,j_1}[k], \mu_{i_1,j_2}[k], \mu_{i_2,j_1}[k], \mu_{i_2,j_2}[k]\}$, and $c = w_{j_1}[k] + w_{j_2}[k]$.

If both parents are homozygous at locus k, then the p-variables of both children must be pre-determined or doubly-determined. However, the two children are not connected by any all-heterozygous path and thus no path constraint is derived. On the other hand, if exactly one parent is heterozygous at locus k, then both children are semi-determined and there is a path constraint between the two children through the heterozygous parent.

For each locus and every pair of pre/semi/doubly-determined p-variables connected by an all-heterozygous path, we construct a path constraint (or two if one of the p-variables is doubly-determined, or four if both p-variables are doubly-determined) as above. Since the pedigree is a tree, the number of such path constraints is at most $O(mn)$. Similarly, for each locus and local cycle, if both parents are heterozygous at the locus, we construct a cycle constraint as above. The number of such cycle constraints is also bounded by $O(mn)$. Let $\mathcal{E}$ denote the set of these constraints.

The results in [45] show that the linear system formed by the above constraints (without the conditions) in terms of the h-variables is equivalent to the linear system defined by Equation (3.3) (without the conditions) in terms of the h- and p-variables. In other words, a feasible solution to the h-variable can be extended to a feasible solution to both the h- and p-variables. It is easy to see that the same equivalence holds with the conditions.

Note that, loci with missing alleles could be included in the linear system in Equation (3.3) (as p-variables). However, they are excluded from the above path/cycle constraints on h-variables. Some of the missing alleles will be imputed using Equation (3.3) after the h-variables are determined.

3.3 The ILP for MMHC and Incremental Approach

We construct an ILP for MMHC based on the above linear system in h-variables. Recall that the objective of the ILP is

$$\text{Minimize} \quad \sum_{i,j,k} \mu_{i,j}[k]. \tag{3.8}$$

We give all the constraints of the ILP in Sections 3.3.1 and 3.3.2. Section 3.3.3 presents more details of the incremental approach to solving the ILP. In Section 3.3.4, we describe how to obtain a solution for MMHC after solving the ILP (and the linear system) and deal with missing alleles.

3.3.1 The Core Constraints

All the constraints in Equation (3.4) are core constraints of the ILP. For each path/cycle constraint $(\mathcal{H}, \mathcal{M}, c)$ in $\mathcal{E}$, we introduce an *equation variable*:

$$E_{\mathcal{H}} = \sum_{h_{i,j} \in \mathcal{H}} h_{i,j} \tag{3.9}$$

We then add an *equation constraint* for each $(\mathcal{H}, \mathcal{M}, c)$:

$$\begin{cases} E_{\mathcal{H}} - \sum_{\mu_{i,j}[k] \in \mathcal{M}} \mu_{i,j}[k] = 0 & \text{if } c = 0 \\ E_{\mathcal{H}} + \sum_{\mu_{i,j}[k] \in \mathcal{M}} \mu_{i,j}[k] = 1 & \text{if } c = 1 \end{cases} \tag{3.10}$$

In other words, either the linear equation in $(\mathcal{H}, \mathcal{M}, c)$ holds, or there is exactly one mutation in $\mathcal{M}$.

Therefore, the core constraints of the ILP includes all the constraints in Equations (3.4), and (3.10). The number of these core constraints is clearly bounded by $O(mn)$.

3.3.2 Consistency Constraints

Now we need some constraints to make sure that the assignment of the equation variables are consistent with each other. Consider, for example, three sets of h-variables $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ that appear in the linear system such that $\mathcal{H}_1 \triangle \mathcal{H}_2 \triangle \mathcal{H}_3 = \emptyset$. (Here, $\triangle$ is the symmetric difference operator.) If $E_{\mathcal{H}_1} = 0$ and $E_{\mathcal{H}_2} = 0$, which are equivalent to $\sum_{h_{i,j} \in \mathcal{H}_1} h_{i,j} = 0$ and $\sum_{h_{i,j} \in \mathcal{H}_2} h_{i,j} = 0$, then we must have $\sum_{h_{i,j} \in \mathcal{H}_3} h_{i,j} = \sum_{h_{i,j} \in \mathcal{H}_1} h_{i,j} + \sum_{h_{i,j} \in \mathcal{H}_2} h_{i,j} = 0$, or equivalently $E_{\mathcal{H}_3} = 0$. The sum of $E_{\mathcal{H}_1}, E_{\mathcal{H}_2}, E_{\mathcal{H}_3}$ must be even. To guarantee such a relation among the three equation variables, we need include the following *consistency constraints*:

$$C(\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3) : \begin{cases} E_{\mathcal{H}_1} + E_{\mathcal{H}_2} + (1 - E_{\mathcal{H}_3}) \geq 1 \\ E_{\mathcal{H}_1} + (1 - E_{\mathcal{H}_2}) + E_{\mathcal{H}_3} \geq 1 \\ (1 - E_{\mathcal{H}_1}) + E_{\mathcal{H}_2} + E_{\mathcal{H}_3} \geq 1 \\ (1 - E_{\mathcal{H}_1}) + (1 - E_{\mathcal{H}_2}) + (1 - E_{\mathcal{H}_3}) \geq 1 \end{cases} \tag{3.11}$$

These constraints ensure that $(E_{\mathcal{H}_1}, E_{\mathcal{H}_2}, E_{\mathcal{H}_3}) \neq (0,0,1), (0,1,0), (1,0,0), (1,1,1)$, respectively. Therefore, illogical combinations of $E_{\mathcal{H}_1}, E_{\mathcal{H}_2}, E_{\mathcal{H}_3}$ are prohibited, and only legitimate combinations are allowed in a feasible solution.

In general, suppose that $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_r$ is any collection of sets of h-variables that appear in the linear system such that $\mathcal{H}_1 \triangle \mathcal{H}_2 \triangle \cdots \triangle \mathcal{H}_r = \emptyset$. To construct the consistency constraints for their corresponding equation variables, we introduce new variables $S_i = \triangle_{i=1}^{j} \mathcal{H}_i$ and their corresponding variables E_{S_i}. We then construct a series of consistency constraints:

$$C(\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_r) = C(\mathcal{H}_1, \mathcal{H}_2, S_2) \cup C(S_{r-2}, \mathcal{H}_{r-1}, \mathcal{H}_r) \cup \bigcup_{i=3}^{r-2} C(S_{i-1}, \mathcal{H}_i, S_i) \tag{3.12}$$

The core constraints and consistency constraints form the complete ILP instance. Note that the number of consistency constraints is generally exponential in n. The following lemma states that these constraints are sufficient for MMHI.

Lemma 2 *Consider a feasible solution to the (complete) ILP defined above. We can convert the conditional linear system in Section 3.2.3 to an unconditional linear system using the values of the equation variables in the solution. The linear system must be consistent.*

Proof. If the linear system is inconsistent, there exists a subset of equations $\left\{\sum_{h \in \mathcal{H}_i} h = c_i\right\}_{i=1}^{r}$ such that the summation of the righthand sides $\sum_{i=1}^{r} c_i = 1$ while the summation of the lefthand sides $\sum_{i=1}^{r} \sum_{h \in \mathcal{H}_r} h = 0$. This implies that $\triangle_{i=1}^{r} \mathcal{H}_i = \emptyset$, and a contradiction to that fact the consistency constraints $C(\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_r)$ are all satisfied. Therefore the linear system must be consistent. ■

3.3.3 The Incremental Approach

Since the complete ILP instance cannot be efficiently constructed in general, we start from an incomplete ILP instance with only the core constraints (no consistency constraints). A standard ILP solver GLPK is invoked to find a solution to the equation variables $E_{\mathcal{H}}$. The equation variable values specifies a set of (unconditional) linear equations from the conditional linear equations in $\mathcal{E}$. We can solve the this system of linear equations by applying Gaussian elimination. However, the linear system may be inconsistent, i.e., there may be a set of equation variable $E_{\mathcal{H}_1}, E_{\mathcal{H}_2}, \ldots, E_{\mathcal{H}_r}$ such that $\triangle_{i=1}^{r} \mathcal{H}_i = \emptyset$ but $\sum_{i=1}^{r} E_{\mathcal{H}_i}$ is odd determined by GLPK. When such an inconsistency occurs, there must be a subset of equations $\{\sum_{h \in \mathcal{H}_i} h = c_i\}_{i=1}^{r}$ such that $\sum_{i=1}^{r} c_i = 1$ but $\sum_{i=1}^{r} \sum_{h \in \mathcal{H}_r} h = 0$. Hence $\triangle_{i=1}^{r} \mathcal{H}_i = \emptyset$, and we add the consistency constraints shown in Equation (3.12) to the ILP instance. We then invoke GLPK again. This process is iterated until a solution is found to yield a consistent system of linear equations.

Although in theory this process may take many iterations, more than 95% of the time in our experiment a consistent solution was found in the very first iteration using only the core constraints. Moreover, the process never took more than three iterations in our experiment. This observation can be explained as follows. For each equation constraint in Equation (3.10), the ILP solver GLPK tends to assign c to the variable $E_{\mathcal{H}}$ given $(\mathcal{M}, \mathcal{H}, c)$ to minimize the number of mutations, if this assignment does not result in conflicting equations. Since the number of mutations is small, most equations should indeed hold. In addition, we usually have a lot of pre-determined μ-variables, which could force GLPK to assign the other variables correctly.

3.3.4 Phasing, Missing Allele Imputation, and Mutation Detection

Once a consistent (unconditional) linear system is found, solving the system by Gaussian elimination assigns the values of all h-variables. GLPK also assigns the values of all μ-variables in the last iteration. Therefore, we can resolve the p-variables by using the *propagation algorithm* in [45, 80]. The basic idea is to propagate known (i.e., pre/semi/doubly-determined) p-variable values to undetermined p-variables along the edges in the pedigree by repeatedly applying Equation (3.3). The p-variables that are left unresolved by the propagation algorithm will be deemed as free in the solution. Note that, the resolved p-variables could allow us to impute missing alleles at some loci (by possibly using some ancestral p-variable and relevant h-variables if necessary), although perhaps not at all loci.

If there are no missing alleles, then the above would produce a consistent solution to the MMHC instance. However, the presence of missing alleles may cause conflict between the assigned values of the μ-variables and those of the p-variables and h-variables. This is because some μ-variables do not appear in any conditional equation. These μ-variables

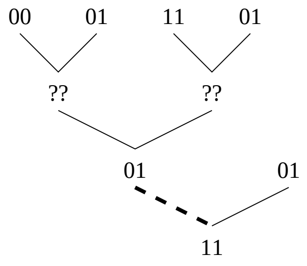


Figure 3.3: Missing alleles may prevent us from obtaining path/cycle constraints. In the figure, if there were no missing data, there should have been two path constraints through the dotted line. The μ-variable on the dotted line is free because the two path constraints are not included in the ILP instance.

only appear in the objective function and the constraints in Equation (3.4). Let us call this type of μ-variables *free*. (See Figure 3.3 for an example free μ-variable.) Clearly, the free μ-variables were set to 0 by GLPK to minimize the objective function. This assignment could be in conflict with the p-variable and h-variable values, because their associated path/cycle constraints were not included in the ILP instance. We will try to fix the problem by re-evaluating the free μ-variables using the determined p-variables and h-variables and Equation (3.3). For any free μ-variable in conflict, we change its value to 1 (which incurs a new mutation).

However, some of these changes might be incorrect (or redundant), and such incorrect changes may potentially lead to other conflicts with the p-variable and h-variable values. When a change leads to more conflicts, we know for sure that the change is wrong (because there can be at most one mutation at the same locus), as stated in the following lemma.

Lemma 3 *If assigning $\mu_{i_1,j_1}[k] = 1$ leads to another conflict that forces $\mu_{i_2,j_2}[k] = 1$, then both $\mu_{i_1,j_1}[k]$ and $\mu_{i_2,j_2}[k]$ should equal* 0*.*

Proof. It is obvious that $\mu_{i_1,j_1}[k]$ should be 0 since otherwise a second mutation at the same locus would be implied. By symmetry, $\mu_{i_2,j_2}[k] = 1$ would lead to $\mu_{i_1,j_1}[k] = 1$ as well. Therefore, both $\mu_{i_1,j_1}[k]$ and $\mu_{i_2,j_2}[k]$ should be 0. ■

Whenever we find two mutations at the same locus, we force their corresponding μ-variables to 0 in the ILP instance (by adding two new constraints), and run GLPK and the propagation algorithm again. Note that, these two μ-variables are no longer viewed as free since they now appear in some constraints in the ILP instance. This process is repeated until all μ-variable values are consistent with the p-variable and h-variable values.

3.4 Experimental Results

We have implemented our algorithm in C, denoted as MMPhase. A detailed pseudocode of MMPhase is given in Algorithm 3.3.1. In the pseudocode, a *founder* is an individual whose parents are not in the pedigree. In this section, we test MMPhase on pedigree data with randomly simulated genotypes, mutations and missing alleles to perform an empirical evaluation of its performance and efficiency. We also compare the speed of MMPhase with that of the straightforward method for solving the MMHC ILP (i.e., running GLPK on all the constraints in a single iteration).

We first compare the speeds of MMPhase and the straightforward method. Since the number of consistency constraints is exponential in the pedigree size n and locus number m in general (even for trees), we implement the straightforward method only for binary trees. When the pedigree is a binary tree, we do not have cycle constraints. For each path constraint along the path between j_1 and j_2, let $\mathcal{H}_1$ be the set of the h-variables on the path from the root of the binary tree to j_1, $\mathcal{H}_2$ the set of the h-variables on the path from the root to j_2, and $\mathcal{H}_3$ the set of h-variables on the path from j_1 to j_2. We put the consistency constraint $C(\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$ into the ILP instance. This will provide a sufficient set of consistency constraints which will guarantee a feasible solution to MMHC. Note that, the number of such consistency constraints is $O(mn)$. Interesting, the incremental approach implemented in MMPhase may theoretically use more consistency constraints in the worst case because of creating redundant variables, although it usually uses a smaller number

Algorithm 3.3.1 MMHC_Phase

```
Input: pedigree $G = (V, E)$ and genotypes $\{\mathbf{g}_j\}$
 1: for each founder $i$ and its first child $j$ do
 2:     $h_{i,j} \leftarrow 0$
 3:     /* Generate the path/cycle constraints */
 4: for each locus $k$ do
 5:     Identify the pre/semi/doubly-determined loci and their anchors
 6:     Identify the pre-determined $\mu$-variables
 7:     for each pair of pre/semi/doubly-determined loci do
 8:         if there is an all-heterozygous path between them then
 9:             Add the corresponding path constraint $(\mathcal{H}, \mathcal{M}, c)$ to $\mathcal{E}$
10:     for each local cycle do
11:         if it implies a cycle constraint then
12:             Add the corresponding cycle constraint $(\mathcal{H}, \mathcal{M}, c)$ to $\mathcal{E}$

13:     /* Constructing the core ILP instance $\mathcal{I}$ */
14: $\mathcal{I}.objective \leftarrow$ minimize $\sum_{i,j,k} \mu_{i,j}[k]$
15: for each locus $k$ do
16:     Add a constraint $\sum_{i,j} \mu_{i,j}[k] \leq 1$
17: for each $(\mathcal{H}, \mathcal{M}, c) \in \mathcal{E}$ do
18:     if $c = 0$ then
19:         Add the equation constraint $E_{\mathcal{H}} - \sum_{\mu_{i,j}[k] \in \mathcal{M}} \mu_{i,j}[k] = 0$
20:     else    /* $c = 1$ */
21:         Add the equation constraint $E_{\mathcal{H}} + \sum_{\mu_{i,j}[k] \in \mathcal{M}} \mu_{i,j}[k] = 1$

22:     /* The incremental method */
23: while true do
24:     Call GLPK to solve the ILP instance
25:     $S \leftarrow \emptyset$
26:     for each $E_{\mathcal{H}}$ do
27:         $S \leftarrow S \cup \left\{\sum_{h \in \mathcal{H}} h = E_{\mathcal{H}}\right\}$
28:     Solve linear system $S$ by Gaussian elimination
29:     if no feasible solution exists then
30:         Let $\left\{\sum_{h \in \mathcal{H}_i} h = E_{\mathcal{H}_i}\right\}_{i=1}^{r}$ be a set of inconsistent equations
31:         Add a consistency constraint $C(\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_r)$
32:     else
33:         Infer the $p$-variables by propagation
34:         Impute missing alleles as much as possible
35:         if the free $\mu$-variables pose no conflict then
36:             break
37:         else
38:             Reassign the conflicting $\mu$-variables
39: Output all $\mathbf{p}_j$ and $\mu$-variable values
```

of consistency constraints in practice. We consider full binary trees of sizes from 63 to 1023, and run both algorithms on 100 randomly generated genotype data with 50 loci, 10% missing alleles, and 3% mutations (i.e., 3% of the loci are mutated in inheritance). (Actually, haplotypes are generated in the simulations and then converted to genotypes as an input of the algorithms.) Table 3.1 shows the average running times of both algorithms on each full binary tree. We observe that MMPhase is much faster than the straightforward method, and the speedup ratio increases as the pedigree size gets bigger. For example, the ratio is about 14 on full binary trees of size 1023. We also observe that the solutions from both algorithms are sometimes slightly different but they always require the same number of mutations which is smaller than the actual number of mutations simulated (the detailed results are not shown).

Next we test the performance of MMPhase in terms of the percentage of correctly phased markers, the percentage of correctly imputed missing alleles, and the percentage of correctly detected mutations. (A simulated mutation is correctly detected if there is an inferred mutation that coincides with its location exactly.) We use three real human pedigrees from the literature as shown in Figure 3.4. 100 replicates of genotype data is simulated on each of these pedigrees with each of several configurations of the number of marker loci, the missing allele rate and the mutation rate. Our default setting of simulation uses the pedigree of size 52, 50 marker loci, 10% missing alleles, and 3% mutations. To observe how each of these parameters affects the performance MMPhase, we vary one parameter at a time in the test. Table 3.2 illustrates the test results. We observe that, as shown in Table 3.2(a), that higher missing rates lead to faster performance since fewer path/cycle

Pedigree size	Straightforward	Incremental
63	.443s	.144s
127	2.98s	.750s
255	20.3s	4.39s
511	180s	29.0s
1023	29.2m	2.13m

Table 3.1: The average running times on 100 randomly generated replicates for each pedigree size. The pedigrees are full binary trees.

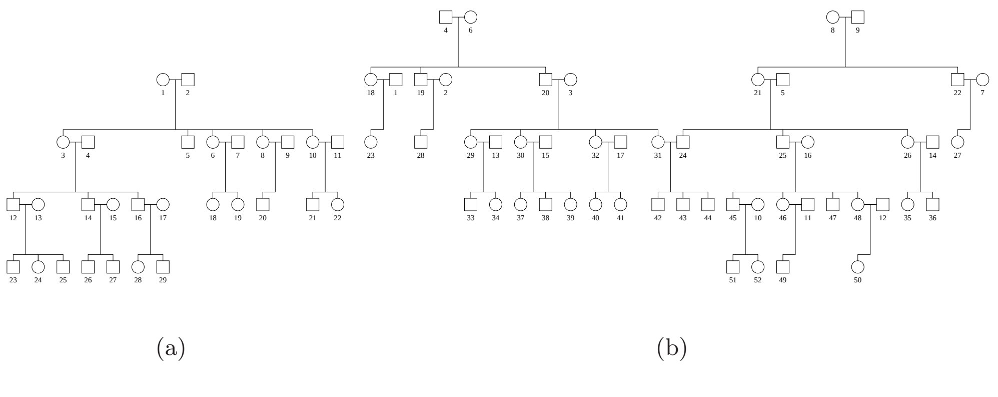


Figure 3.4: Three pedigrees are used to test the performance of MMPhase. The first has 29 individuals and is shown in (a). The second has 52 individuals and is shown in (b). The third has 128 individuals and is too large to fit in the page.

Missing rate	Correctly imputed missing alleles	Correctly detected mutations	Correctly phased markers	Running time
0%	—	78.31%	99.98%	2.02s
5%	74.70%	70.20%	98.49%	1.49s
10%	68.97%	62.58%	92.72%	1.24s
20%	69.03%	59.69%	92.75%	.900s

(a)

Mutation rate	Correctly imputed missing alleles	Correctly detected mutations	Correctly phased markers	Running time
1%	73.15%	73.33%	96.75%	1.23s
3%	68.97%	62.58%	92.72%	1.24s
10%	73.11%	69.57%	96.73%	1.47s

(b)

Pedigree size	Correctly imputed missing alleles	Correctly detected mutations	Correctly phased markers	Running time
29	75.34%	52.26%	94.51%	.298s
52	68.97%	62.58%	92.72%	1.24s
128	73.49%	52.11%	93.99%	27.0s

(c)

Locus number	Correctly imputed missing alleles	Correctly detected mutations	Correctly phased markers	Running time
20	73.13%	67.00%	96.78%	.250s
50	68.97%	62.58%	92.72%	1.24s
200	73.09%	65.42%	96.82%	10.8s

(d)

Table 3.2: The performance of MMPhase under various configurations of the parameters. The default setting includes the pedigree of size 52, 50 marker loci, 10% missing alleles, and 3% mutations. 100 replicated are generated for each configuration of the parameters. Starting from the default setting, we vary the missing rate in (a), the mutation rate in (b), the pedigree in (c), and the number of loci in (d).

constraints are added to the ILP instance. Not surprisingly, higher missing rates also result in fewer correctly detected mutations and fewer correctly phased markers. Table 3.2(b) shows that the performance is not very sensitive to the mutation rate. Table 3.2(c) and Table 3.2(d) show that the pedigree and number of marker loci mainly affect the running time.

In conclusion, MMPhase is very efficient and can infer haplotypes very accurately. It can also recover most of the mutation and missing alleles correctly. Note that, our criterion for correctly detecting a mutation is very stringent since in some cases the mutation could be shifted in the pedigree without affecting the feasibility of the solution (especially when missing alleles are present).

3.5 Concluding Remarks

It would be interesting to extend the method to deal with both mutations and genotyping errors. Genotyping errors are very common in practice and can easily be confused with mutations. In the next chapter, we will discuss how to handle both mutations and genotyping errors.

Chapter 4

Inferring Haplotypes from Genotypes on a Pedigree with Mutations, Genotyping Errors and Missing Alleles

4.1 Introduction

In this chapter, we study haplotype inference on pedigree data consisting of tightly linked markers that have no recombinants but may contain some genotyping errors, missing alleles and a very small number of *de novo* mutations (or simply, mutations). Since error and mutation events are rare, we formulate the problem as a combinatorial optimization problem, called the *haplotype configuration with mutations and errors* (HCME) problem, where we look for a haplotype solution consistent with the given genotype data that incurs no recombinants and requires the minimum number of mutations and errors. (Actually, we minimize a weighted summation of the numbers of mutations and errors.) Our hypothesis is that the configuration with the minimum number of mutations and errors is likely the true solution. This extends the well studied *zero-recombinant haplotype configuration* (ZRHC) problem where we try to find a consistent haplotype solution incurring no recombinants, errors or mutations. Although ZRHC is polynomial-time solvable [40], we prove that HCME is NP-hard by a reduction from NAE-3SAT. We construct an *integer linear program* (ILP) for HCME using the system of linear equations over Galois field $GF(2)$ that has been developed in Refs. [40, 80] and [46] for solving ZRHC in almost linear time. We use the incremental approach introduced in a previous work [76] to reduce the number of constraints in the ILP instance.

We have implemented the algorithm and tested it on both simulated data and real data. The experimental results demonstrate that our algorithm can infer haplotypes with a very high accuracy. It can also recover most of the errors and impute most of the missing alleles correctly. The error recovery rate is higher than the error detection rates in Refs. [12] and [84]. However, most mutations would be explained by genotyping errors because we give errors a smaller weight (since they are more frequent than mutations) in the objective function of the above ILP.

The rest of the chapter is organized as follows. In Section 4.2, we introduce the linear system, and formally define HCME as an optimization problem. In Section 4.3, we explain each part of our algorithm in a subsection. We explain how to construct constraints for the given genotype data in Section 4.3.1. We briefly describe how to tackle the problem by using ILP as a black box, which is similar to our previous work, in Section 4.3.2. We explain how to recover the optimal haplotype configuration in Section 4.3.3. Some detailed implementation issues are discussed in Section 4.3.4. In Section 4.4, we show our experimental results on both simulated data and real data, and discuss how each parameter may affect the accuracy and efficiency of our algorithms. In Section 4.5, we give a proof of the NP-hardness of HCME. Section 4.6 concludes this chapter with a few remarks. This chapter appeared in proceedins of Computational System Bioinformatics 2010, pp. 192–203. This chapter was also published in Journal of Bioinformatics and Computational Biology (JBCB), Vol. 9, No. 2 (2011) pp. 339–365 [77].

A pedigree can be drawn in three different forms: conventional form, formal form and graph form (Figure 4.1). A pedigree is a tree if its formal form is a tree. In this chapter, we will follow Ref. [76] to consider only tree pedigrees, which are very common among human pedigrees. From now on, when we talk about paths and cycles in a pedigree, we will consider it as a graph form. Note that a tree pedigree may have local cycles within nuclear families (in its graph form).

4.2 Preliminaries

In this section, we first review the linear system developed in Refs. [80] and [46] for dealing with pedigree data without mutations and errors. We then explain how to modify the linear system for handling mutations and errors, and give the formal definition of HCME.

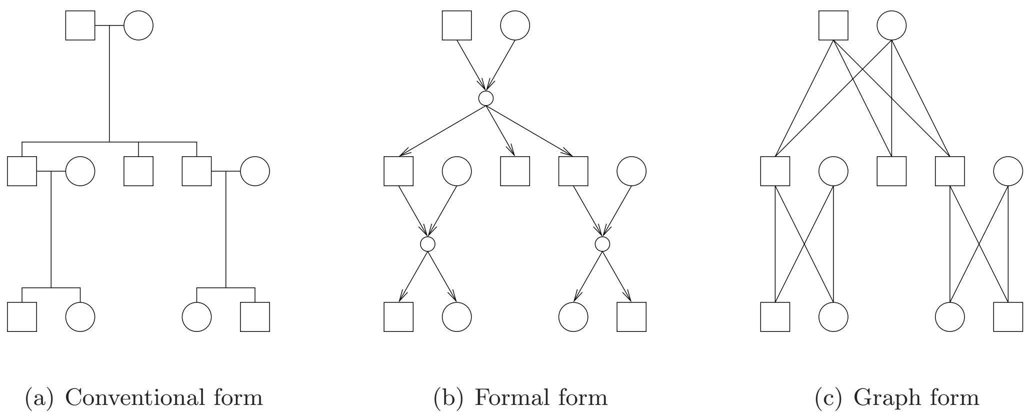


Figure 4.1: Three equivalent pedigrees in different forms. The formal form contains mating nodes (smaller circles). In the graph form, each pair of parent-child is connected with an edge.

4.2.1 The Linear System

Let n denote the number of individuals in the input pedigree and m the number of marker loci of each individual. In this chapter, we assume all alleles are bi-allelic (0 or 1). The genotypes of individual j is denoted as a ternary vector $\mathbf{g}_j$ and its kth entry $g_j[k]$ represents the genotype at locus k of individual j:

$$g_j[k] = \begin{cases} 0 & \text{if both alleles are 0's} \\ 1 & \text{if both alleles are 1's} \\ 2 & \text{if the locus is heterozygous} \end{cases}$$

For individual j, we define $\mathbf{p}_j \in GF(2)^m$ as the paternal haplotype of individual j. Each entry $p_j[k]$ of $\mathbf{p}_j$ is defined on $GF(2)$. Clearly, if $g_j[k] = 0$ or 1, then we can derive $p_j[k] = g_j[k]$ directly. To represent the maternal haplotype of individual j, we define $\mathbf{w}_j \in GF(2)^m$ to indicate if each locus of individual j is heterozygous. That is, $w_j[k] = 0$ if $g_j[k] = 0$ or 1, and $w_j[k] = 1$ if $g_j[k] = 2$. Clearly, the summation $\mathbf{p}_j + \mathbf{w}_j$ over $GF(2)$

represents the maternal haplotype of individual j.

Suppose that individual i is a parent of individual j. To unify the representation of the haplotype that j inherited from i, we define a binary vector $\mathbf{d}_{i,j}$ as follows: $\mathbf{d}_{i,j} = 0$ if i is j's father and $\mathbf{d}_{i,j} = \mathbf{w}_j$ if i is j's mother. Therefore, $\mathbf{p}_j + \mathbf{d}_{i,j}$ represents the haplotype that j got from i. We define $h_{i,j} \in GF(2)$ such that $h_{i,j} = 0$ if $\mathbf{p}_j + \mathbf{d}_{i,j}$ is i's paternal haplotype and $h_{i,j} = 1$ otherwise. Then $\mathbf{p}_i + h_{i,j} \cdot \mathbf{w}_i$ represents the haplotype that i passed to j. The binary variables $h_{i,j}$ thus fully describe the inheritance pattern in an ZRHC instance. Using these notations, we can derive an equation over $GF(2)$:

$$p_i[k] + h_{i,j} \cdot w_i[k] = p_j[k] + d_{i,j}[k] \quad \forall \text{ parent-child pair } (i,j), \forall \text{ locus } k \tag{4.1}$$

4.2.2 Impact of Mutations and Errors

We define a *mutation variable* $\mu_{i,j}[k] \in \mathbb{Z}$, $\mu_{i,j}[k] = 1$ if there is a mutation at locus k when i passes his haplotype to offspring j, and $\mu_{i,j}[k] = 0$ otherwise. For convenience, we make these three vectors symmetric by defining $\mathbf{d}_{j,i} = \mathbf{d}_{i,j}$, $h_{j,i} = h_{i,j}$, and $\mu_{j,i} = \mu_{i,j}$. Using these notations, we modify Equation (4.1) and obtain

$$p_i[k] + h_{i,j} \cdot w_i[k] = p_j[k] + d_{i,j}[k] + I(\mu_{i,j}[k] = 1) \tag{4.2}$$

where $I(\cdot)$ is the indicator function also defined on $GF(2)$.

An *error variable* $e_j[k] \in \mathbb{Z}$ of individual j at locus k is defined as $e_j[k] = 1$ if the observed genotype $g^o_j[k]$ is different from the actual genotype $g_j[k]$, and $e_j[k] = 0$ if there is no error, i.e., when $g^o_j[k] = g_j[k]$. These errors hinder us from getting correct $p_i[k]$, $p_j[k]$, $w_i[k]$, and $d_{i,j}[k]$ in Equation (4.2).

We define $w_j^o[k]$, $p_j^o[k]$, and $d_{i,j}^o[k]$ as $w_j[k]$, $p_j[k]$, and $d_{i,j}[k]$ derived from the observed genotype $g_j^o[k]$. If $g_j^o[k] = 0$ or 1, then $w_j^o[k] = 0$ and $p_j^o[k] = g_j^o[k]$. If $g_j^o[k] = 2$, then $w_j^o[k] = 1$ and $p_j^o[k] = p_j[k]$, which cannot be determined immediately. We define $d_{i,j}^o[k] = 0$ if i is j's father and $d_{i,j}^o[k] = w_j^o[k]$ otherwise.

As we initialize variables that we can derive from the genotype data, we need to consider possible errors. We start with what we can observe, $\mathbf{p}_j^o$ and $\mathbf{w}_j^o$, and derive a *conditional equation* over $GF(2)$:

$$p_i^o[k] + h_{i,j} \cdot w_i^o[k] = p_j^o[k] + d_{i,j}^o[k] + I(\mu_{i,j}[k] = 1) \qquad \text{if } e_i[k] = e_j[k] = 0 \tag{4.3}$$

The HCME problem can be formally defined as follows. Given an input pedigree and genotype data $\mathbf{g}_j^o$ for each individual j, find a solution to each $\mathbf{g}_j$, $\mathbf{p}_j$, $\mathbf{w}_j$, $h_{i,j}$, $\mu_{i,j}$ and e_j that satisfies Equation (4.2) and (4.3), and minimizes the *mutation-error score*

$$c_1 \sum\nolimits_{i,j,k} \mu_{i,j}[k] + c_2 \sum\nolimits_{j,k} e_j[k]$$

where $c_1, c_2 > 0$ are weights for mutations and errors. These adjustable weights allow us to change preference between mutations and errors as the rates of mutations and errors may change in different applications. Our default is that $c_1 = 1.5$ and $c_2 = 1$. We assign a bigger weight to each mutation because mutation events typically have a much lower frequency ($\sim 10^{-9}$) than errors (0.1%–5%) do [22]. On the other hand, the larger weight for mutations makes it difficult to detect mutations, i.e., they could easily be replaced by errors especially when the pedigree is shallow.

4.3 Method

We first construct an equivalent conditional linear system with fewer variables, which can be converted to an ILP instance. The objective function of the ILP instance is still to minimize the mutation-error score. A standard ILP solver GLPK (the GNU Linear Programming Kit from `http://www.gnu.org/softward/glpk`) is invoked to solve the ILP. The ILP solution describes an unconditional linear system, which may or may not be consistent. The consistency of the unconditional linear system can be checked by Gaussian elimination. If it is consistent, we obtain a temporary assignment of the h-variables. If it is not consistent, we add more constraints and then invoke the ILP solver again. The ILP solver together with the Gaussian elimination subroutine will be referred to as the *black box*, which was introduced in our previous work [76]. The temporary h-variable assignment from the black box is often optimal and, in this case, we can compute an optimal haplotype configuration with the assignment. However, because of the loss of information (see Section 4.3.1) due to genotyping errors, the temporary h-variable assignment may sometimes be suboptimal, although it is usually very close to an optimal h-variable assignment. We start from the temporary h-variable assignment and search if there is any better h-variable assignment. We use the best h-variable assignment that we have found to compute the final haplotype configuration.

Occasionally, the black box may have difficulty assigning variables and exceed a predetermined time limit. If the black box exceeds the time limit, we apply some heuristic rules to reduce the size of the ILP instance and then call the black box again.

Figure 4.2 shows the flowchart of our method. The construction of path constraints and cycle constraints will be explained in Section 4.3.1. The black box will be explained in Section 4.3.2. The search for better h-variables and recovery of haplotype configurations will be explained in Section 4.3.3. The time limit as well as some implementation issues that are not covered in the flowchart will be explained in Section 4.3.4.

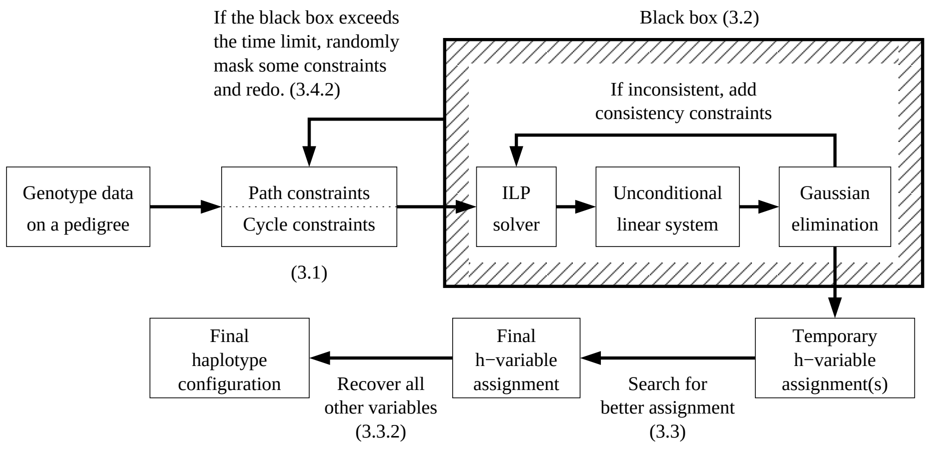

If the black box exceeds the time limit, randomly mask some constraints and redo. (3.4.2)
Black box (3.2)
If inconsistent, add consistency constraints
Genotype data on a pedigree
Path constraints
Cycle constraints
(3.1)
ILP solver
Unconditional linear system
Gaussian elimination
Final haplotype configuration
Final h−variable assignment
Temporary h−variable assignment(s)
Recover all other variables (3.3.2)
Search for better assignment (3.3)

Figure 4.2: The outline of our algorithm. Numbers in parentheses indicate the sections with the corresponding details.

4.3.1 Construction of Constraints

There are $O(mn)$ variables and equations in the linear system described in Section 4.2. As in Refs. [76] and [46], we can convert the system to an equivalent linear system involving only the h-variables, mutation variables, and error variables.

Equations with Fewer Variables

The idea is to consider paths in the pedigree (of the graph form) connecting individuals with homozygous markers and derive equality constraints on the h-variables on such paths based on Equation (4.3). Consider a locus k and a path $j_0, j_1, \ldots, j_r$ in the input pedigree, where individuals j_i and j_{i+1} form a parent-child or child-parent pair. Suppose that $g^o_{j_0}[k]$ and $g^o_{j_r}[k]$ are homozygous, and $g^o_{j_1}[k] = \cdots = g^o_{j_{r-1}}[k] = 2$. We call the path $j_0, j_1, \ldots, j_r$ an *all-heterozygous path* at locus k. Since $g^o_{j_0}[k]$ and $g^o_{j_r}[k]$ are homozygous, we have $p^o_{j_0}[k] = g^o_{j_0}[k]$ and $p^o_{j_r}[k] = g^o_{j_r}[k]$. We add up all conditional equations as given in Equation (4.3) for all parent-child pairs on path $j_0, j_1, \ldots, j_r$ to obtain a *path constraint* connecting j_0 and j_r:

$$\begin{aligned}
&\sum_{i=1}^{r-2} h_{j_i,j_{i+1}} + h_{j_0,j_1} \cdot I(j_1 \text{ is } j_0\text{'s parent}) + h_{j_{r-1},r} \cdot I(j_{r-1} \text{ is } j_r\text{'s parent}) \\
&\qquad = g^o_{j_0}[k] + g^o_{j_r}[k] + \sum_{i=0}^{r-1} d^o_{j_i,j_{i+1}}[k] + I\left(\sum_{i=0}^{r-1} \mu_{j_i,j_{i+1}}[k] \text{ is odd}\right) \qquad (4.4) \\
&\qquad \text{if } e_{j_i}[k] = 0, i = 0, \ldots, r.
\end{aligned}$$

We denote $\mathcal{M} = \bigcup_{i=0}^{r-1}\{\mu_{j_i,j_{i+1}}[k]\}$, $\mathcal{E} = \bigcup_{i=0}^{r}\{e_{j_i}[k]\}$, and $c = g^o_{j_0}[k]+g^o_{j_r}[k]+\sum_{i=0}^{r-1} d^o_{j_i,j_{i+1}}[k]$, and let $\mathcal{H}$ be the collection of h-variables that have coefficient 1 in Equation (4.4), i.e., $h_{j_0,j_1} \in \mathcal{H}$ if and only if j_1 is j_0's parent, etc. The path constraint can be represented by the quadruple $(\mathcal{H}, \mathcal{M}, \mathcal{E}, c)$ which denotes that

$$\sum\nolimits_{h_{i,j}\in\mathcal{H}} h_{i,j} = c + I\left(\sum\nolimits_{\mu\in\mathcal{M}} \mu \text{ is odd}\right) \qquad \text{if } \sum\nolimits_{e\in\mathcal{E}} e = 0 \qquad (4.5)$$

Note that the observed genotypes along the path may actually contain more than one error, but our path constraint could be satisfied with at most one error. Thus, we may underestimate the number of errors by using such a path constraint. For convenience, we will refer to this inherent limitation of path constraints as the *loss of information*, which is mainly caused by the fact that we do not know the actual genotypes.

Consider a *local cycle* consisting of father i_1, mother i_2, and two adjacent children j_1, j_2. If both parents are heterozygous at locus k, we can obtain four conditional equations from Equation (4.3) by replacing i with i_1, i_2, and j with j_1, j_2 (see Figure 4.3). The summation of these conditional equations forms a *cycle constraint*:

$$h_{i_1,j_1} + h_{i_1,j_2} + h_{i_2,j_1} + h_{i_2,j_2} = w^o_{j_1}[k] + w^o_{j_2}[k] + I\left(\sum\nolimits_{\mu\in\mathcal{M}} \mu \text{ is odd}\right)$$
$$\text{if } \sum\nolimits_{e\in\mathcal{E}} e = 0$$

where $\mathcal{M} = \{\mu_{i_1,j_1}[k], \mu_{i_1,j_2}[k], \mu_{i_2,j_1}[k], \mu_{i_2,j_2}[k]\}$, $\mathcal{E} = \{e_{i_1}[k], e_{i_2}[k], e_{j_1}[k], e_{j_2}[k]\}$. This constraint will also be denoted as $(\mathcal{H}, \mathcal{M}, \mathcal{E}, c)$ if we let $\mathcal{H} = \{h_{i_1,j_1}, h_{i_1,j_2}, h_{i_2,j_1}, h_{i_2,j_2}\}$ and $c = w_{j_1}[k] + w_{j_2}[k]$.

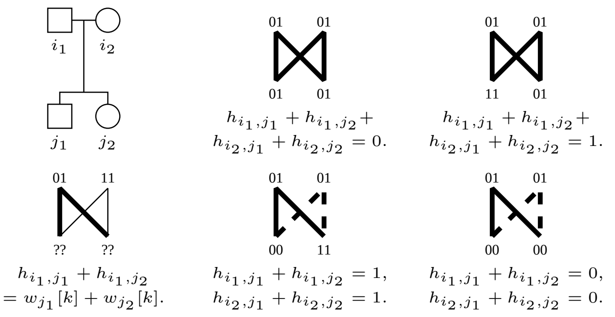

i_1
i_2
j_1
j_2
01 01
01 01
$h_{i_1,j_1} + h_{i_1,j_2} +$
$h_{i_2,j_1} + h_{i_2,j_2} = 0.$
01 01
11 01
$h_{i_1,j_1} + h_{i_1,j_2} +$
$h_{i_2,j_1} + h_{i_2,j_2} = 1.$
01 11
?? ??
$h_{i_1,j_1} + h_{i_1,j_2}$
$= w_{j_1}[k] + w_{j_2}[k].$
01 01
00 11
$h_{i_1,j_1} + h_{i_1,j_2} = 1,$
$h_{i_2,j_1} + h_{i_2,j_2} = 1.$
01 01
00 00
$h_{i_1,j_1} + h_{i_1,j_2} = 0,$
$h_{i_2,j_1} + h_{i_2,j_2} = 0.$

Figure 4.3: If both parents are heterozygous, and at least one child is heterozygous, then we will a cycle constraint. If both parents are homozygous, then there is no constraint. Otherwise, we will obtain either one or two path constraints, depending on the situation.

Let us look at the path constraints generated in the above nuclear family more closely. If both parents in the local cycle are homozygous at locus k, then the corresponding path constraint from one parent to the other will consist of no h-variables, and thus no path constraint will be derived. If one parent is heterozygous, the other is homozygous and both children are heterozygous at locus k, then we can derive a path constraint from the homozygous parent through one child, the other parent, the other child, and back to the homozygous parent. If there is exactly one homozygous parent and one homozygous child, the path constraint should be derived through the heterozygous child and the other parent. Otherwise, no path constraint will be derived for this local cycle.

For each locus and every pair of homozygous markers, we construct a path constraint as above. Since the pedigree is a tree, the number of such path constraints is at most $O(mn)$. Similarly, for each locus and local cycle, if both parents are heterozygous at the locus, we construct a cycle constraint as above. The number of such cycle constraints is also bounded by $O(mn)$. Let $\mathcal{S}$ denote the set of these constraints. The results in Ref. [46] show that the linear system formed by the above constraints (without the conditions) in terms of the h-variables is equivalent to the linear system defined by Equation (4.3) (without the conditions) in terms of the h- and p-variables. In HCME with mutations and errors, a feasible assignment to the h-, μ-, and e-variables, can be extended to a feasible solution to all the h-, p-, μ- and e-variables.

Note that, loci with missing alleles could possibly be included in the linear system in Equation (4.3) (as p-variables). However, they are excluded from the above path/cycle constraints on h-variables. Some of the missing alleles will be imputed after the h-variables are determined.

The ILP Instance

We construct an ILP instance for HCME based on the above linear system in h-variables. Recall that the objective function of the ILP is the mutation-error score

$$c_1 \sum\nolimits_{i,j,k} \mu_{i,j}[k] + c_2 \sum\nolimits_{j,k} e_j[k]$$

In our ILP instance, the path/cycle constraint $(\mathcal{H}, \mathcal{M}, \mathcal{E}, c)$ as given in Equation (4.5) is actually modified as

$$\sum\nolimits_{h_{i,j}\in\mathcal{H}} h_{i,j} = c, \quad \text{if } \sum\nolimits_{\mu\in\mathcal{M}} \mu = \sum\nolimits_{e\in\mathcal{E}} e = 0 \tag{4.6}$$

with three technical reasons. First, there are rarely two or more mutations on a locus [33]. Second, Equation (4.5) is not accurate because of the loss of information, and thus the ILP solver produces suboptimal intermediate results anyway. Third, Equation (4.6) generates a smaller ILP instance and is more efficient.

For each path/cycle constraint $(\mathcal{H}, \mathcal{M}, \mathcal{E}, c)$ in $\mathcal{S}$, we introduce a binary *equation variable* as in Ref. [76]

$$E_{\mathcal{H}} = \sum\nolimits_{h_{i,j}\in\mathcal{H}} h_{i,j} \tag{4.7}$$

and require that the corresponding quadruple $(E_{\mathcal{H}}, \sum_{\mu\in\mathcal{M}} \mu, \sum_{e\in\mathcal{E}} e, c)$ must not be $(0, 0, 0, 1)$ or $(1, 0, 0, 0)$.

$$\left\{ \begin{array}{ccccccccc} E_{\mathcal{H}} & + & \sum_{\mu\in\mathcal{M}} \mu & + & \sum_{e\in\mathcal{E}} e & + & (1-c) & \geq 1 \\ (1-E_{\mathcal{H}}) & + & \sum_{\mu\in\mathcal{M}} \mu & + & \sum_{e\in\mathcal{E}} e & + & c & \geq 1 \end{array} \right. \tag{4.8}$$

In other words, if there are no mutations and errors, the equation in Equation (4.6) must hold. The final ILP instance includes all the constraints in Equation (4.8). The number of constraints is clearly bounded by $O(mn)$.

4.3.2 The Black Box

The black box consists of two elements: the ILP solver and the Gaussian elimination subroutine. After we set up the ILP instance as above, we invoke the ILP solver. The

ILP solver will return an assignment of the mutation variables, error variables and equation variables. The assignment of the mutation and error variables is not accurate because of the loss of information and will be ignored. The assignment of the equation variables implies an unconditional linear system of the h-variables. The linear system can be solved by Gaussian elimination and we then obtain a solution of the h-variables. The linear system is usually consistent. If the linear system is not consistent, our Gaussian elimination subroutine will detect inconsistent equation variables. Since the linear system is on $GF(2)$, inconsistencies occur only if there are 3 or more equations such that the summation of the their polynomial (i.e., the left-hand side) parts is 0 while the summation of their constant (i.e., the right-hand side) parts is 1. Such inconsistency can be prevented by introducing some *consistency constraints* as done in our previous work [76].

Consistency constraints make sure that the assignments of the involved equation variables in the constraints will be consistent with each other. For example, suppose that there are 3 equation variables $E_{\mathcal{H}_1}, E_{\mathcal{H}_2}, E_{\mathcal{H}_3}$ and the summation of their polynomials is 0, i.e., $\sum_{h \in \mathcal{H}_1} h + \sum_{h \in \mathcal{H}_2} h + \sum_{h \in \mathcal{H}_3} h = 0$ on $GF(2)$. The assignment of $(E_{\mathcal{H}_1}, E_{\mathcal{H}_2}, E_{\mathcal{H}_3})$ must not be $(0,0,1), (0,1,0), (1,0,0), (1,1,1)$, because otherwise the summation of the constant part would be 1, which results in inconsistency. For instance, the consistency constraints for the three equation variables $E_{\mathcal{H}_1}, E_{\mathcal{H}_2}, E_{\mathcal{H}_3}$ in the above example would include 4 inequalities:

$$C(\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3) : \begin{cases} E_{\mathcal{H}_1} + E_{\mathcal{H}_2} + (1 - E_{\mathcal{H}_3}) \geq 1 \\ E_{\mathcal{H}_1} + (1 - E_{\mathcal{H}_2}) + E_{\mathcal{H}_3} \geq 1 \\ (1 - E_{\mathcal{H}_1}) + E_{\mathcal{H}_2} + E_{\mathcal{H}_3} \geq 1 \\ (1 - E_{\mathcal{H}_1}) + (1 - E_{\mathcal{H}_2}) + (1 - E_{\mathcal{H}_3}) \geq 1 \end{cases}$$

The details and general form of consistency constraints can be found in Ref. [76].

When the Gaussian elimination subroutine detects inconsistent equation variables, we add the corresponding consistency constraints to the ILP instance and invoke the ILP solver again. Inconsistency may happen repeatedly, and we keep adding consistency constraints until the assignment of all equation variables is consistent. Note that by not including consistency constraints at the beginning, we enhance the efficiency of our program without losing any accuracy.

4.3.3 Recovery of Haplotype Configurations

Search for a Better h-variable Assignment

The temporary h-variable assignment retrieved from Gaussian elimination is usually an optimal assignment. If it is not, it usually differs from an optimal assignment at very few h-variables. We evaluate an h-variable assignment by computing the minimum possible mutation-error score which is consistent with the h-variable assignment. The computation of mutation-error score is explained in Section 4.3.3.

Let t be the number of h-variables and $\mathbf{H}^{(0)} = [h_{i_1,j_1}, h_{i_2,j_2}, \ldots, h_{i_t,j_t}]$ be the temporary assignment of h-variables obtained from Gaussian elimination. Let u_r be a unit vector on $GF(2)^t$ with a 1 on the rth entry. Assume the $\mathbf{H}^{(k)}$ is the h-variable assignment after k rounds of searching. In the $(k+1)$st round, we consider t possible h-variable assignment $\mathbf{H}_r^{(k+1)} = \mathbf{H}^{(k)} + u_r$, $r = 1, \ldots, t$. We compute the minimum mutation-error score for each assignment, and $\mathbf{H}^{(k+1)}$ will be the assignment with the minimum score among $\mathbf{H}^{(k)}$ and $\mathbf{H}_r^{(k+1)}$, $r = 1, \ldots, t$. This search continues until $\mathbf{H}^{(k)} = \mathbf{H}^{(k+1)}$ for some k, and $\mathbf{H}^{(k)}$ will be the final h-variable assignment.

Computing Mutation-Error Scores and Haplotype Configurations

In this section, we assume that the h-variable assignment is fixed. Once the h-variables are assigned, we need to assign errors and mutations that minimize the mutation-error score as given in Equation (4.7). We use the dynamic programming method to find the minimum mutation-error score as follows. Here, each locus is considered separately.

We pick an arbitrary node (individual) as the root of the input (tree) pedigree. For each node, we consider all four possible genotypes and compute the best mutation-error score of the subtree under the node for each genotype. The score could be computed with the score of children with respect to the rooted tree of all possible genotypes. We compute the score iteratively from the leaves to the root. This dynamic programming procedure determines an optimal assignment of all mutations and errors by a simple backtracking subroutine, where ties are broken arbitrarily.

After assigning errors, we mark genotypes where errors are assigned as missing alleles. For each pair of alleles of a non-missing marker, if there is no error and the genotype is homozygous, we mark the alleles as *confirmed*. We start from confirmed alleles, update parent's/children's alleles with the given h-variables and the mutation assignment, and confirm these newly updated alleles. We keep updating and confirming alleles until there are no more alleles that we can update. Alleles that are not confirmed remain unknown. After updating all the alleles, we obtain the whole haplotype configuration with the locations of mutations and correction of errors. We output the configuration if the h-variable assignment is the final assignment.

4.3.4 A Few Implementation Issues

Free Variables

Occasionally, the solution of the unconditional linear system obtained from ILP solver is not unique. If there are two or more solutions, then we have multiple h-variable assignments as starting points of the search. We simply do the search described in Section 4.3.3 for all starting points and select the best one. We record h-variable assignments that have been searched, and avoid redundant search.

Time Limit

Sometimes, the ILP solver may have difficulty in determining errors, mutations, and equation variables. It is usually because there are some "troublesome" loci with errors. If the ILP solver fails to assign errors to these loci and attempts to assign errors to other loci that do not actually have errors, it may take a very long time for the ILP solver to find a feasible assignment, or the ILP solver may keep returning inconsistent assignments of equation variables. To avoid these situations, we set a time limit. The default time limit is chosen as 3 minutes based on empirical observations. When the black box fails to return an assignment of h-variables within the time limit, we terminate the black box. We modify the ILP instance as follows and then run the black box again. We repeat this process if the black box keeps exceeding the time limit.

The first time the black box exceeds the time limit, we check if we can fix the values of some equation variables to reduce the ILP solver's work. For example, suppose that there are 9 or more constraints extracted from different loci with the same $\mathcal{H}$ and c, i.e., $E_{\mathcal{H}} = c$ is suggested 9 times with different conditions, and there are none or only 1 constraint supporting $E_{\mathcal{H}} = 1 - c$. The odds strongly favor $E_{\mathcal{H}} = c$ and thus we can simply assign c to $E_{\mathcal{H}}$. Let us call such a constraint that suggests $E_{\mathcal{H}} = 1 - c$ *conflicting*. If there are two or more conflicting constraints (over different paths or cycles) extracted from the same locus, then the locus is considered *troublesome*. We *mask* all troublesome loci by dropping all constraints extracted from them.

If the black box exceeds the time limit the second time, we randomly select a locus and mask it. If the black box keeps exceeding the time limit, we will increase the number of masked loci by 1 each time. Occasionally, the black box may keep running and failing for a long time. We thus have a global time limit. If our program exceeds the global time limit, we terminate the computation and concede. The default global time limit is currently set as 30 minutes.

4.4 Experimental Results

We have implemented our algorithms in C, denoted as MePhase. In this section, we test MePhase on both simulated data and real data to perform an empirical evaluation of its accuracy and efficiency. For simulated data, we generate both tree pedigrees and genotype data randomly. We design experiments to test how the pedigree topology and data quality (i.e., the missing and genotyping error rates) may affect the performance of MePhase. We do not consider a very large number of loci since the zero-recombinant assumption holds only for tightly linked markers. For real data, we use the SNP microarray data published by Wirtenberger et al. [79]. They genotyped 16 members of a three-generation family using the GeneChip Human Mapping 10K Array (Affymetrix). Since Wirtenberger et al. did not delete Mendelian inheritance errors in their published data, the data along with their report of recombinant regions are ideal for us to test MePhase. According to Ref. [79], there are 6.24% missing alleles and 0.29% Mendelian inheritance errors detected in all family trios in the data. However, Hao et al. have reported that the genotyping error rate of GeneChip Mapping 10K Array is about 0.1% [22], which is much smaller than the error rate given in Ref. [79]. We think that the number of Hao et al. is more accurate because they focused on the estimation the of error rate while Wirtenberger et al. did not. Our results also support the error rate given by Hao et al.

4.4.1 Simulated Data

Thomas et al. have proposed algorithms to generate tree pedigrees uniformly randomly with specific numbers of individuals and mating notes [73]. Their algorithms have been implemented in Java, which will be used to generate pedigrees in this chapter. Zou et al. showed that the size of a nuclear family may affect the accuracy of error detection [84]. Thus, we consider the average nuclear family size as a parameter along with pedigree size and analyze how nuclear family and pedigree sizes may affect the accuracy and efficiency of our algorithms. Let f be the average nuclear family size and n again denote the number of individuals. Because tree pedigrees are considered, f is determined by n and the number of mating nodes:

$$f = \frac{n + \#\text{ of mating nodes} - 1}{\#\text{ of mating nodes}}$$

Therefore, we can alter the number of mating nodes to obtain pedigrees of different average nuclear family sizes. We will generate pedigrees with $n = 29, 52, 67, 82$ and $f = 3, 4, 5, 6$. For each of the 16 combinations of n and f, we generate 5 random pedigrees. This results in 80 different pedigrees in total. Note that for $n = 29$, we could only achieve $f = 5.67$ instead of 6 because the number of mating nodes should be an integer.

To generate genotype data, we use haplotype data downloaded from HapMap (`http://hapmap.ncbi.nlm.nih.gov/downloads/phasing/2009-02_phaseIII/HapMap3_r2/`). For each run of the experiment, we randomly select an interval of SNPs. Then for each founder in the pedigree, we randomly pick two haplotypes to form the genotype of the founder. Each individual randomly passes one of its two haplotypes to each of its offsprings, where mutations are incorporated randomly according to the mutation rate. This results in a genotype for each individual. Finally, missing alleles and genotyping errors are (uniformly) randomly added to the genotypes according to their rates. We will also perform simulations using the false homozygosity error model, in which heterozygous loci are falsely typed as homozygous loci [66].

We run MePhase on each simulated data and compare the haplotype configurations given by MePhase with the true haplotype configurations of the simulated data. We will consider the ability of MePhase in imputing missing alleles and detecting mutations and genotyping errors. When the location of a detected error is slightly different from that of the true error in the pedigree, we say that the error is *shifted*. Table 4.1 shows the impact of n and f on both efficiency and accuracy of MePhase in phasing as well as error detection and correction. MePhase infers haplotypes with a very high accuracy in all settings. Note that MePhase can correct up to 94% of errors, while the method in Zou et al. [84] can only detect 81% of errors in a similar setting. In general, a bigger f leads to denser constraints, less freedom of haplotype assignments, and a higher accuracy. On the other hand, a smaller f leads to more founders. When an error event happens to the genotype of a founder, there is a good chance that the erroneous genotype does not violate the Mendelian law of inheritance. This is likely to keep the error undetectable. Therefore, a smaller f will lead to more founders and more undetected errors. Table 4.1 also clearly illustrates that a bigger n leads to a longer running time. Moreover, a bigger f also leads to a longer running time, since a bigger f leads to more path and cycle constraints.

Most mutations (85%–95%) are explained by errors because we give errors a smaller weight. A mutation on a founder will often be explained by an error, or remain undetectable if the Mendelian law of inheritance is not violated. A mutated allele will for sure be explained by an error if it does not get passed to any offspring. Most mutations can be explained by errors with some small shifts within the pedigree, especially when the mutations are near

Average nuclear family size	Pedigree size (n)	Recovered errors	Shifted errors	Undetected errors	Correctly phased markers	Suboptimal solution	Average time (sec)	Failure rate
$f = 3$	29	67.1%	9.9%	23.0%	99.2%	0.0%	0.03	0.0%
	52	67.6%	9.2%	23.2%	99.5%	0.3%	1.37	0.0%
	67	66.2%	8.9%	25.0%	99.7%	0.9%	1.88	0.0%
	82	65.9%	8.6%	25.4%	99.7%	0.6%	4.99	0.0%
$f = 4$	29	83.9%	5.8%	10.3%	99.3%	0.2%	4.71	0.0%
	52	82.7%	5.6%	11.7%	99.6%	0.2%	18.3	2.0%
	67	83.4%	5.0%	11.6%	99.7%	0.8%	30.3	0.7%
	82	83.4%	5.4%	11.3%	99.8%	0.8%	49.1	0.0%
$f = 5$	29	91.7%	3.7%	4.5%	99.3%	0.0%	4.92	0.7%
	52	90.7%	2.9%	6.4%	99.5%	0.5%	46.5	1.3%
	67	91.1%	3.0%	5.9%	99.7%	0.7%	402	0.7%
	82	90.8%	3.4%	5.8%	99.9%	0.9%	498	2.7%
$f = 6$	29	93.1%	2.8%	4.1%	99.3%	0.1%	22.1	1.3%
	52	94.5%	2.1%	3.4%	99.6%	0.4%	661	0.0%
	67	94.4%	1.8%	3.8%	99.7%	0.6%	619	0.0%
	82	94.4%	2.0%	3.6%	99.8%	1.0%	740	2.7%

Table 4.1: The impact of pedigree size n and average nuclear family size f on the efficiency and accuracy of MePhase in phasing and error detection and correction. Here, the number of loci $m = 50$, the missing rate = 0, the error rate = 0.5%, and the mutation rate = 1 per pedigree on average. For testing the accuracy of MePhase and the optimality of its solution, we ran 200 replicates for each pedigree and set the global time as 10 minutes. For testing the time efficiency and failure rate, we ran 30 replicates for each pedigree and set the global time limit as 30 minutes (but 180 minutes for $(n, f) = (52, 6), (62, 5), (62, 6), (82, 5), (82, 6)$). Clearly, MePhase rarely fails to find a solution under this setting.

founders or children with no offsprings. Our combinatorial optimization model is incapable of catching such mutations precisely.

As shown in Table 4.1, MePhase may produce a suboptimal solution, especially when most genotypes are homozygous, since it may return an h-variable assignment that is locally optimal during the search for a better h-variable assignment. The suboptimal h-variable assignment usually differs from the true assignment at many h-variables, but requires only one more error than the true haplotype solution.

In our experiment, we observed that if the black box in MePhase fails to return an assignment within the time limit once, it will tend to fail many times. The standard ILP solver GLPK uses a branch-and-bound algorithm to find integral solutions. When GLPK falls into a bad branch, it may take very long time for GLPK to get out of the branch. This creates a big variance in running time. For bigger n and f, we had to give MePhase a longer global time limit.

The quality (i.e., the missing and genotyping error rates) of data may also have an impact on both accuracy and efficiency of MePhase. We run MePhase on 5 pedigrees of $n = 29$ and $f = 4$ with various missing and error rates as shown in Table 4.2. A large variance in running time is observed which might explain why there is no clear impact of data quality on running time. The missing rate has some impact on the running time, while the error rate does not affect the running time much. However, the impact of data quality on accuracy is small but noticeable. When the missing rate goes higher, the accuracies in phasing, error correction and missing allele imputation slightly decrease. The genotyping error rate has a smaller impact on the accuracy of MePhase, but the trend is clear.

Error rate	Missing rate	Correctly recovered errors	Correctly imputed missing alleles	Correctly phased markers	Suboptimal solution	Average time	Failure rate
0.0%	0%	–	–	99.7%	0.0%	2.30	0.0%
	5%	–	79.3%	98.6%	0.0%	58.2	3.3%
	10%	–	77.4%	97.2%	0.0%	82.3	6.0%
	20%	–	72.2%	93.5%	0.0%	97.3	11.3%
0.5%	0%	85.0%	–	99.6%	0.0%	13.0	0.0%
	5%	83.3%	78.5%	98.4%	0.7%	24.2	5.3%
	10%	83.3%	76.6%	97.1%	0.7%	65.0	6.7%
	20%	79.4%	72.1%	93.4%	0.7%	70.9	8.0%
1.0%	0%	85.3%	–	99.3%	0.0%	2.37	0.7%
	5%	83.2%	78.8%	98.2%	0.0%	44.5	1.3%
	10%	79.5%	76.6%	96.7%	0.0%	64.1	6.7%
	20%	75.1%	72.0%	93.2%	0.7%	39.9	7.3%
2.0%	0%	83.2%	–	98.9%	0.7%	27.8	2.0%
	5%	80.6%	77.9%	97.6%	0.7%	44.7	1.3%
	10%	79.3%	75.2%	96.1%	0.7%	38.7	3.3%
	20%	75.6%	70.7%	92.5%	0.7%	46.5	6.0%
3.0%	0%	82.4%	–	98.4%	0.7%	30.8	4.0%
	5%	80.2%	76.8%	96.9%	0.7%	58.1	4.7%
	10%	78.6%	74.4%	95.6%	0.7%	61.2	4.7%
	20%	74.9%	69.9%	91.7%	1.3%	66.1	3.3%

Table 4.2: The impact of data quality on accuracy and efficiency. Here, $n = 29$, $f = 4$, $m = 50$, the mutation rate = 1 per pedigree on average, the global time limit is set as 30 minutes, and we ran 30 replicates for each setting.

We have also repeated the above experiment with genotyping errors simulated using the false homozygosity model [66]. For simplicity, we allow heterozygous markers to be miscalled as homozygous markers, but assume that homozygous markers are always called correctly. The result is similar to those in Table 4.2 except that the percentage of correctly recovered errors decreases by about 8–13% on the average. Note that in our general error model, a homozygous marker (e.g., 00) may be miscalled as the complement homozygous marker (i.e., 11). This type of errors bring inconsistencies to both parents and hardly remain undetectable. Since these easily-detected errors are not included in false homozygosity error model, the error recovery rate of the program decreases. The detailed result is omitted in this chapter.

4.4.2 Real Data

Figure 4.4(a) shows the pedigree used in Ref. [79]. Individuals 17 and 18 (i.e., the founders) are missing, and they have only two children (i.e., individuals 1 and 13). Therefore, these two children are under no constraints. We partition the pedigree into pedigrees A and B, and run MePhase on both pedigrees separately. Wirtenberger et al. reported a set of recombinant regions (i.e., intervals where a recombinant might be located) in the SNP microarray data. We preprocess the SNP data by eliminating all SNP loci in the recombinant regions. A set of loci between two neighboring recombinant regions is considered as a *block*. We then run MePhase on each block separately.

Table 4.3 shows the number of errors that MePhase found in both pedigrees A and B. The table also shows the number of *hidden errors*, which are errors that cannot be found

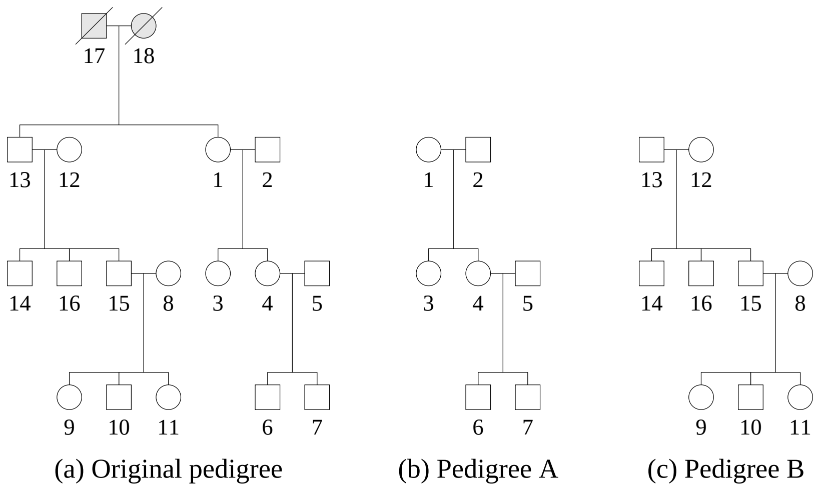


Figure 4.4: The diagram in (a) shows the pedigree of the SNP microarray data reported by Wirtenberger et al [79]. Since the two founders are completely missing, we divide the pedigree into two disjoint pedigrees A and B.

by checking the Mendelian law of inheritance within nuclear families, found by MePhase. MePhase also identified a mutation on chromosome 6. The overall genotyping error rate in non-recombinant regions found by MePhase is 0.175%. If we exclude the hidden errors, the error rate would be reduced to 0.146%, which is close to the error rate reported in Ref. [22].

To estimate the accuracy of our reported numbers, we also run experiments with simulated data on pedigrees A and B. The result (data not included here) shows that MePhase is able to detect 85% of the errors on pedigree A and 87% of the errors on pedigree B. By extrapolation, we estimate that the genotyping errors in the non-recombinant regions of this SNP microarray data is close to 0.2% taking into account the errors that MePhase might have failed to detect.

			Pedigree A		Pedigree B	
Chromosome number	Number of blocks	Number of loci	Errors found	Hidden errors	Errors found	Hidden errors
1	26	372	2	1	8	3
2	23	484	2	0	9	1
3	19	385	0	0	10	1
4	22	428	7	0	4	0
5	19	406	2	1	7	0
6*	16	508	7	0	4	1
7	21	353	7	1	8	1
8	13	307	1	0	8	1
9	15	307	1	0	3	0
10	22	245	2	1	5	2
11	17	406	7	2	13	1
12	17	312	2	1	6	1
13	18	294	2	0	9	1
14	13	182	0	0	3	0
15	12	197	3	2	4	0
16	6	135	2	1	2	0
17	6	93	1	0	4	3
18	9	231	2	0	5	2
19	4	29	0	0	0	0
20	7	105	0	0	2	0
21	5	141	2	0	0	0
22	2	6	0	0	0	0

Table 4.3: The number of blocks, loci, and detected errors on each chromosome in each pedigree. The asterisk indicates that a mutation is reported on chromosome 6.

4.5 NP-hardness of HCME

In this section, we prove that the HCME problem is NP-hard even if the pedigree is a binary tree. We will reduce NAE-3SAT, which is NP-hard [55], to the binary-tree HCME problem. NAE-3SAT is a variant of 3SAT in which a clause is satisfied if all three literals are not equal in truth value. For an NAE-3SAT instance ϕ, we convert it to an HCME instance $\mathcal{I}$ of polynomial size. We will prove that ϕ is satisfiable if and only if $\mathcal{I}$ has a feasible solution with a given number of mutations/errors.

We first introduce two gadgets called the *phase control* and *clause verifier*. In

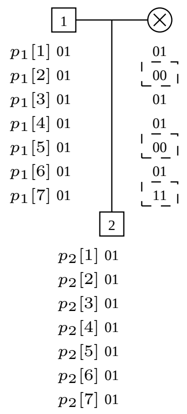


Figure 4.5: Phase control. The individual marked with $\times$ is a phase control. The markers of the phase control ensure that $p_2[2] = p_2[5] \neq p_2[7]$ if there is no mutation/error.

Figure 4.5, the individual marked with $\times$ serves a phase control. Consider the p-variables $p_2[1], \ldots, p_2[7]$ of the child, the phase control on the right ensures that $p_2[2] = p_2[5] \neq p_2[7]$ if there is no mutation/error. Since the markers of child and father are all heterozygous, we have either $\mathbf{p}_2 = \mathbf{p}_1$ or $\mathbf{p}_2 = 1 - \mathbf{p}_1$. Therefore, we also have $p_1[2] = p_1[5] \neq p_1[7]$ for the father. Phase controls are designed to set up equations/inequations among marker loci. For loci we don't consider, we simply set them heterozygous in the phase control and it allows all possible combination of values of p-variables.

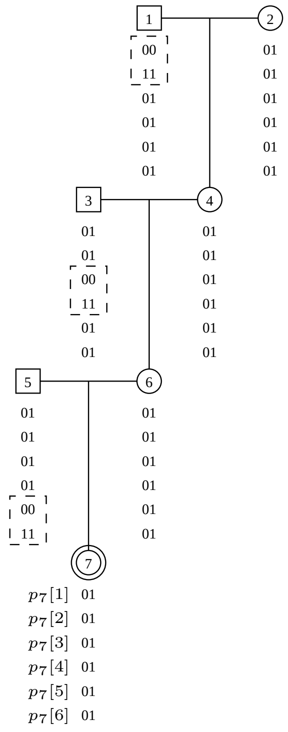


Figure 4.6: Clause verifier. This gadget ensures that $p_7[1] \neq p_7[2]$, $p_7[3] \neq p_7[4]$, and $p_7[5] \neq p_7[6]$ if there is no mutation/error. Individual 7 (marked by a double circle) is the "port" of the clause verifier.

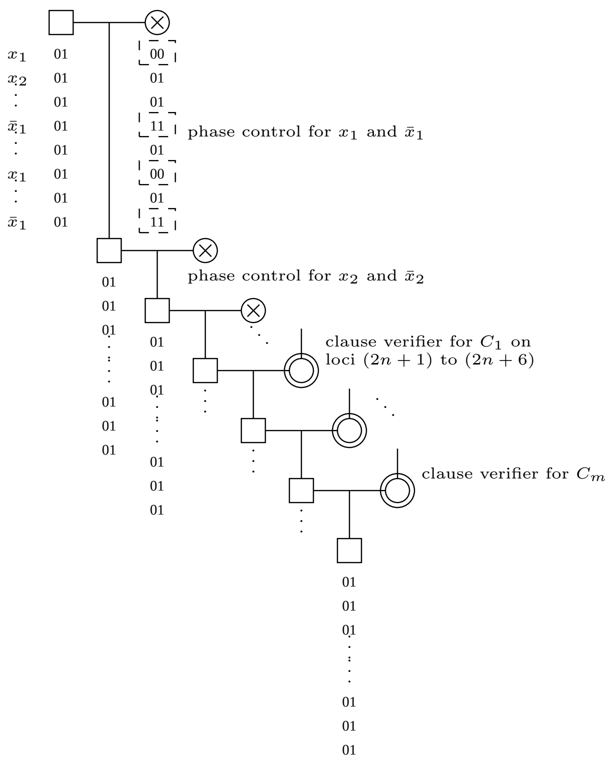
Figure 4.6 shows a clause verifier. If there is no mutation/error in the gadget, then

$$p_7[1] \neq p_7[2]$$

$$p_7[3] \neq p_7[4]$$

$$p_7[5] \neq p_7[6]$$

Figure 4.7 shows the HCME instance $\mathcal{I}$ constructed according to a given NAE-3SAT instance as follows. There is a main path from the top-left individual to the bottom



x_1
x_2
$\bar{x}_1$
x_1
$\bar{x}_1$
00
11
phase control for x_1 and $\bar{x}_1$
phase control for x_2 and $\bar{x}_2$
clause verifier for C_1 on loci $(2n+1)$ to $(2n+6)$
clause verifier for C_m
01

Figure 4.7: A generic construction of the HCME instance for a given NAE-3SAT instance. Double circles and crosses indicate where clause verifiers and phase controls should be installed. Phase controls and clause verifiers are set up according to the NAE-3SAT instance.

last child. Every individual on the main path has only heterozygous markers. Suppose that the NAE-3SAT instance has n variables $x_1, \ldots, x_n$ and m clause $C_1, \ldots, C_m$. We construct a binary-tree HCME instance $\mathcal{I}$ with $(2n+6m)$ loci indexed with $1, 2, \ldots, 2n+6m$. We label the first $2n$ loci with $x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$. For the remaining $6m$ loci, we partition them into m groups of 6 loci. Each group of 6 loci is associated with a clause in ϕ. We define $l_r(i) = 2n + 6(r-1) + i$, which represents the ith locus of the rth group. The rth group is associated with the rth clause in ϕ. For the rth clause $C_r = (y_i \vee y_j \vee y_k)$, we label loci $l_r(1)$ to $l_r(6)$ with $y_i, y_j, y_j, y_k, y_k, y_i$.

There are $(n+m)$ mating events on the main path. For each of the first n mating events, we install a phase control. The rth phase control has 00 on the loci labeled with x_r, 11 on the loci labeled with $\bar{x}_r$, and 01 on the remaining loci. For each of the last m mating events, we install a clause verifier. The rth clause verifier is set up on loci $l_r(1)$ to $l_r(6)$ and contain only heterozygous markers on all other loci.

To prove that the HCME problem is NP-hard, our goal is to show that the NAE-3SAT instance ϕ has a feasible solution if and only if the corresponding HCME instance $\mathcal{I}$ has a feasible haplotype configuration with exactly m mutations/errors distributed from loci $l_1(1)$ to $l_m(6)$. We can show that the collection of p variables of the last child c (or any other individual) of the main path on the first n loci $(p_c[1], \ldots, p_c[n])$ will be a feasible assignment for the NAE-3SAT instance ϕ. The complement $(1-p_c[1], \ldots, 1-p_c[n])$ is also a feasible assignment.

Lemma 4 *Consider an individual u and one of its descendants v, both on the main path. Suppose that there is no mutation/error on loci i, j and k in the whole pedigree. If there is phase control that guarantees the condition $p_u[i] = p_u[j] \neq p_u[k]$, then the condition propagates to the individual v, i.e., $p_v[i] = p_v[j] \neq p_v[k]$.*

Proof. Let $\mathbf{p}_u^*, \mathbf{p}_v^*$ be obtained from $\mathbf{p}_u, \mathbf{p}_v$ of a feasible haplotype configuration by omitting loci that have mutations/errors. If v is u's child, since u and v have only heterozygous markers, we have $\mathbf{p}_v^* \in \{\mathbf{p}_u^*, 1-\mathbf{p}_u^*\}$. By induction, $\mathbf{p}_v^* \in \{\mathbf{p}_u^*, 1-\mathbf{p}_u^*\}$ holds as long as v is u's descendant. Given $p_u[i] = p_u[j] \neq p_u[k]$, $\mathbf{p}_v^* \in \{\mathbf{p}_u^*, 1-\mathbf{p}_u^*\}$ ensures $p_v[i] = p_v[j] \neq p_v[k]$. ■

Lemma 5 *For any feasible haplotype configuration of $\mathcal{I}$, there are at least m mutations/errors. For each $r = 1, \ldots, m$, there is at least one mutation or error among loci $l_r(1), \ldots, l_r(6)$.*

Proof. Consider p-variables $p_c[l_r(1)], \ldots, p_c[l_r(6)]$ of the last child c. Suppose that there is no mutation/error on loci $l_r(1), \ldots, l_r(6)$. By Lemma 4, the phase controls ensure

$$p_c[l_r(1)] = p_c[l_r(6)]$$

$$p_c[l_r(2)] = p_c[l_r(3)]$$

$$p_c[l_r(4)] = p_c[l_r(5)]$$

and the clause verifier for these loci ensure

$$p_c[l_r(1)] \neq p_c[l_r(2)]$$

$$p_c[l_r(3)] \neq p_c[l_r(4)]$$

$$p_c[l_r(5)] \neq p_c[l_r(6)]$$

The above system of six linear equations/inequations has no feasible solution on *GF*(2). Therefore, there must be a mutation or error among loci $l_r(1), \ldots, l_r(6)$. Since we have at least one mutation/error among loci $l_r(1), \ldots, l_r(6)$ for each $r = 1, \ldots, m$, there must be at least m mutations/errors in any feasible haplotype configuration of $\mathcal{I}$. ■

Lemma 6 *If there is a feasible assignment $\phi(t_1, \ldots, t_n)$ for the NAE-3SAT instance, then there is a feasible haplotype configuration with exactly m mutations/errors, one in each group of 6 loci $l_r(1), \ldots, l_r(6)$, $r = 1, \ldots, m$.*

Proof. Let y be a literal. We denote $\phi_T(y)$ as the value of y (true $= 1$ and false $= 0$) in the assignment $\phi(t_1, \ldots, t_n)$. For each individual on the main path, we simply assign the values of its p-variables according to the corresponding labels as follows. For all loci labeled with x_i, we assign $\phi_T(x_i)$ to the involved p-variables. For all loci labeled with $\bar{x}_i$, we assign $\phi_T(\bar{x}_i)$ to the involved p-variables. The h-variables are all 0 through the main path. The h-variables on edges connecting the main path and phase controls, i.e., individuals marked with $\times$'s are all set to be 0. For each clause $C_r = (y_i, y_j, y_k)$, $r = 1, \ldots, m$, we assign

$$\begin{aligned}
h^{(r)}_{7,\circ} &= \phi_T(y_k) \\
h^{(r)}_{6,7} &= \phi_T(y_j) + h^{(r)}_{7,\circ} + 1 \\
h^{(r)}_{4,6} &= \phi_T(y_i) + h^{(r)}_{7,\circ} + h^{(r)}_{4,6}
\end{aligned}$$

Here, the superscript (r) refers to the rth clause verifier, the subscripts refer to the individual number in Figure 4.6 and $\circ$ refers to the individual on the main path that is the child of individual 7 of the rth clause verifier. We then either assign a mutation as

$$\begin{aligned}
\mu^{(r)}_{1,4}[l_r(2)] &= I\left(\phi_T(y_i) = \phi_T(y_j)\right) \\
\mu^{(r)}_{3,6}[l_r(4)] &= I\left(\phi_T(y_j) = \phi_T(y_k)\right) \\
\mu^{(r)}_{5,7}[l_r(6)] &= I\left(\phi_T(y_k) = \phi_T(y_i)\right)
\end{aligned}$$

or alternatively assign an error as

$$e_1^{(r)}[l_r(2)] = I\left(\phi_T(y_i) = \phi_T(y_j)\right)$$

$$e_3^{(r)}[l_r(4)] = I\left(\phi_T(y_j) = \phi_T(y_k)\right)$$

$$e_5^{(r)}[l_r(6)] = I\left(\phi_T(y_k) = \phi_T(y_i)\right)$$

The rest of the h-variables can be assigned arbitrarily, and the p-variables in clause verifiers can be assigned accordingly. Since $\phi_T(y_i), \phi_T(y_j), \phi_T(y_k)$ are from a feasible truth assignment and y_i, y_j, y_k all appear in the same clause, exact one of the following three equations holds: $\phi_T(y_i) = \phi_T(y_j)$, $\phi_T(y_j) = \phi_T(y_k)$, $\phi_T(y_k) = \phi_T(y_i)$. Hence we have exactly one mutation or error among loci $l_r(1), \ldots, l_r(6)$. ■

Lemma 7 *Given an HCME instance $\mathcal{I}$ as above, if there is a feasible haplotype configuration with m mutations/errors, then the NAE-3SAT instance ϕ is satisfiable. Moreover, if c is the last child on the main path, then $(p_c[1], \ldots, p_c[n])$ is a feasible assignment of ϕ.*

Proof. Since there are exactly m mutations/errors, there must be exact one mutation or error in each group of 6 loci $l_r(1), \ldots, l_r(6)$, $r = 1, \ldots, m$ by Lemma 5, and there must be no mutations/errors on the first $2n$ loci. Let c be the last child on the main path. Consider all loci that contain no mutation/error. If two loci i and j have the same label, then the p-variable $p_c[i]$ and $p_c[j]$ must be equal. This is guaranteed by phase controls and the lack of interference from mutations/errors.

For each group of 6 loci $l_r(1), \ldots l_r(6)$, $r = 1, \ldots, m$, since there is only one mutation or error involved, there is at least a pair (i, j) out of the three pairs $(1, 2), (3, 4), (5, 6)$

such that $p_c[l_r(i)] \neq p_c[l_r(j)]$ due to the clause verifiers and lack of interference from mutations/errors. Therefore, the clause associated with the group of 6 loci is satisfied. The instance ϕ has a feasible assignment. ■

Theorem 8 *The HCME problem is NP-hard.*

Proof. Given any NAE-3SAT instance ϕ with n variables and m clauses, we can construct an HCME instance $\mathcal{I}$ of quadratic size with $2n+6m$ loci and $4n+8m+1$ individuals in polynomial time. The NAE-3SAT instance ϕ has a feasible assignment if and only if the minimum mutation/error haplotype configuration of $\mathcal{I}$ has exactly m mutations/errors. Therefore, NAE-3SAT is polynomial time reducible to HCME, which makes HCME NP-hard. ■

Note that HCME is NP-hard as long as the weights of mutations and errors c_1, c_2 are greater than 0. This implies that the special cases of HCME where only mutations are allowed (i.e., $c_2 = \infty$) or only errors are allowed (i.e., $c_1 = \infty$) are also NP-hard.

4.6 Conclusion

We have introduced a combinatorial optimization model for haplotype inference on pedigrees in the presence of mutations and genotyping errors that generalizes the previous models in the literature. We have designed and implemented an heuristic algorithm for the model based on the previously developed system of linear equations and ILP. Our experimental results on simulated data demonstrate that our program can infer haplotypes with a very high accuracy, impute most missing alleles, detect and correct most genotyping errors, and identify some mutations (although many mutations could be confused with errors).

Chapter 5

Detection of Horizontal Gene Transfers in Bacterial Genomes

5.1 Introduction

In this chapter, we adhere to a new definition of SNPs based on surrounding sequences proposed in [16, 43] instead of genomic positions as done traditional. This new definition accomplishes several needs: (1) it can be applied across different species, (2) it avoids focusing in a small region which may be affected by an HGT to increase the accuracy, and (3) it avoids whole-genome alignment, which may not scale.

A SNP locus is defined as a region of length k that is conserved among one or more genomes in the target set except at the center base, the SNP allele, which varies among one or more genomes. Thus, the sequence context of $(k-1)/2$ bases on both sides of the SNP allele describe the SNP locus, and must be conserved among at least a subset of the target genomes. We allow a SNP locus to occur more than once in a genome on either strand if and only if all occurrences within that genome have the identical allele, which allows us to include more of the sequence variations in genomes with duplications and draft genomes with gaps and assembly errors. If a locus occurs more than once within a genome and the allele differs, that SNP locus is omitted and considered missing from that genome, although that locus may still be considered a valid SNP in other genomes without allele conflicts. SNP-based trees can be created from a SNP matrix of the alleles for each genome concatenated together like a sequence alignment listing the alleles in each genome. Then a maximum likelihood tree can be generated using RAxML v7.2.7 [67]. We let $k = 25$ for bacterial genomes and $k = 13$ for virus.

In this chapter, we study the detection of mutations, HGTs and errors given the SNPs and SNP positions of a set of closely related strains with an evolutionary species tree. The SNPs of all leaf nodes are mostly known with some missing, but the SNPs of all internal nodes are unknown. Some known SNPs might be incorrect because of sequencing errors. Some genomes might be in the form of contigs, i.e., the SNP positions are only in the correct order and orientation within a contig. We want to reconstruct the SNPs of internal nodes with regard to 3 possible events. (1) Mutations. A single SNP may change when an internal node passes its SNPs to its child node. (2) HGTs. A node may get a segment of SNPs from any other node which is not one of its descendants. (3) Sequencing errors. The data we have may be wrong.

We cannot distinguish sequencing errors from mutations that occur on the leaf nodes. For simplicity, all SNP disagreements between a leaf node and its parent node are considered as “errors” (although in reality some may be true SNP variations). Therefore, mutations refer to SNPs mapping to internal nodes, and errors refer to SNPs mapping to leaf nodes. Each event has a weight. The weights of mutation/HGT/error are w_m, w_x, and w_e, respectively. We want to reconstruct the events and SNPs of all nodes (including leaf nodes because there might be errors), while minimizing the total weight. The frequencies of mutation/HGT/error events are low, and the assignment that minimizes the total weight would give a reasonable explanation [59]. Note that the error weight w_e is always less than the mutation weight w_m, since SNP variations on leaf nodes are always considered as errors. Considering a horizontal gene transfer, if the source and the destination consist of different SNP surrounding sequences, then we cannot distinguish it from a duplication, or other rearrangement events. Therefore, we only consider HGTs that have the same SNP surrounding sequences in the same order and orientation in both the source and destination, i.e., homologous HGTs.[1] Moreover, there should be no inversion endpoints within the HGT region in source and destination strains, otherwise the SNP surrounding sequences would be different in source and destination. We use a greedy algorithm to partition genomes into *blocks* in which inversions do not take place. We then use the dynamic programming technique to assign mutations/HGTs/errors in each block. We also consider possible HGT from an out-group, i.e., some species not in the given evolutionary species tree. Figure 5.1 shows an example of the detection of HGTs problem within a block. There are six SNPs loci, and the SNPs on the leaf nodes (2, 4, 6, 7) are known. We can explain SNPs on node 6 by three mutations or one HGT, and we believe one HGT is more likely than three

[1]There are exceptions, and we will explain the exceptions in Section 5.2.1.

neighboring mutations.

We have implemented our algorithms that partition genomes into blocks and assign mutations/HGTs/errors. We have tested the program on both simulated data and real data. The experimental simulation results demonstrate that there are many HGT and mutation events that leave no evidence to be detected, and the accuracy mainly depends on the mutation rate, HGT rate, and the size of the evolutionary tree.

The rest of this chapter is organized as follows. In Section 5.2, we explain three different parts of our algorithm. We explain how we partition the genomes into blocks in Section 5.2.1, how we apply dynamic programming technique in Section 5.2.2, and how we determine if an HGT is from an out-group of the given strains in Section 5.2.3. In Section 5.3, we show our experimental results on both simulated data and real data, and discuss how each parameter may affect the accuracy. Section 5.4 concludes this chapter with a few remarks.

5.2 Method

The sequences of source and destination of a homologous HGT should be similar, i.e., there should be the same set of SNPs in the same order and orientation in the HGT regions of both donor and recipient genomes. However, the SNP order/orientation may not be totally the same throughout all genomes, because of genome rearrangement events, i.e., inversions and transpositions, and we have to focus on regions that all genomes have the same SNP order and orientation. We define a *block* as a region in which there is no evidence of genome inversion. We do not consider transpositions since transpositions can

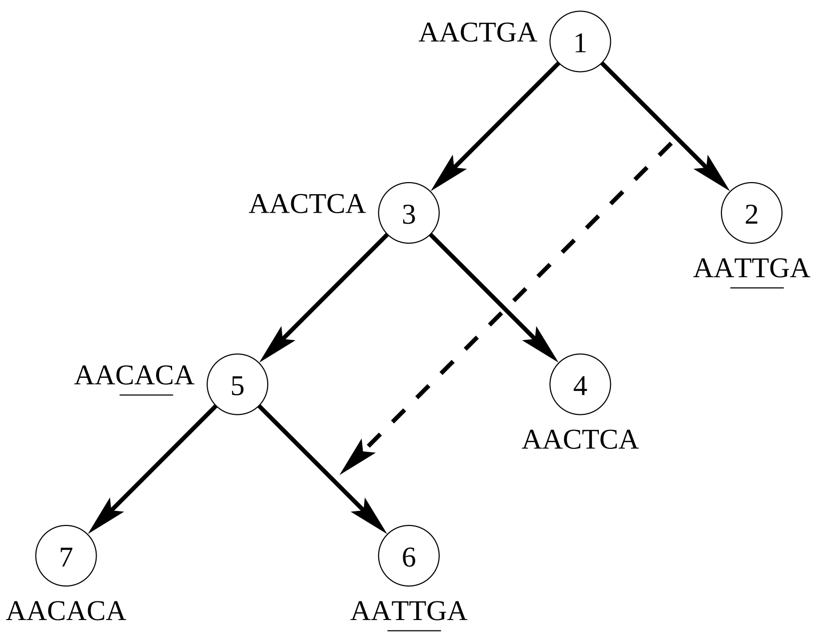


Figure 5.1: An example of detection of HGTs. The SNPs on node 6 are better explained by an HGT from node 2 than inheritance from node 5 with three mutations.

be mimicked by inversions. SNPs in a block should be in the same order across all genomes with some exceptions explained in Section 5.2.1.

We first partition the genomes into blocks by a greedy block extension algorithm, then we consider each block separately. Within each block, for each SNP locus, we use dynamic programming to reconstruct the SNPs of internal nodes in the evolutionary tree with the minimum number of mutations. Then within each block, we check if we can assign HGTs to further reduce the total weight by dynamic programming. We also consider possible HGTs from an out-group species not in the given strains. After assigning mutations/HGTs/errors, we trace each SNP where it is originally from and evaluate if there is any evidence that indicates an HGT from an out-group.

5.2.1 Computing Blocks with Duplications and Missing SNPs

Considering a possible block B and a genome G, we say that G *agrees* with B if, given the genome G, there is no evidence that suggests an inversion within the block B. A straightforward example is, if all the SNPs in B appear consecutively in G in the same order and orientation, or all in the reverse order and complement orientation, then G agrees with B. Different orders usually suggest inversions, but there are some exceptions.

1. Missing. A SNP may appear in B but be absent in G, and it does not suggest an inversion. For example, $B = bcd$, G contains a SNP sequence $abde$ and c is absent in G, then G should agree with B.
2. Duplication. There might be duplicated SNPs inserted to G and they could alter the SNP order. For example, $B = bcd$, G contains a SNP sequence $abcbde$, then the second b in G should be considered as a duplicated SNP, and G should agree with B.
3. Contigs. The genome may be in contig form, which makes the SNP order in G unclear. For example, $B = abcd$, G contains a contig ending with SNP sequence ab and a contig starting with cd, then G should agree with B.

We formally define the notion of agreement as follows.

Definition 9 *Let Σ be the set of forward and reverse complement of all SNPs. A block is a string $B \in \Sigma^+$ and a genome is a set of strings $G = \{g_1, g_2, \cdots, g_k\}$, $g_i \in \Sigma^+$ ($k > 1$ if in contig form). Let $B|_G$ be the subsequence of B obtained by deleting SNPs absent in G. Let D_G be the set of SNPs that appear more than once in G. We say the genome G agrees* *with the block B if and only if there exists a string S such that the two following statements both hold. (1) There exists a concatenation $G^* = g_{j_1} g_{j_2} \cdots g_{j_k}$ allowing reverse complement and S is a substring of G^*. (2) $B|_G$ is a subsequence of S and S can be obtained from $B|_G$ by inserting only SNPs in D_G.*

When considering if a genome G agrees with a block B, we try to match the SNP order and orientation in B and G. If a SNP s appears in B but does not appear in G, then s should be skipped in B in the matching. If a SNP s appears in G more than once, then we can choose to skip s in G or not, based on if it makes the SNP order/orientation in G different from those in B. When we try to match the SNP order/orientation but the comparison reaches the end of a contig, then the next match in G can start from any other end of a contig. Let s be the SNP in B we want to match when the comparison reaches the end of a contig in G. We check all occurrences of s and see if any occurrence of s is at the end of a contig (or only duplicated SNPs between s and the end of the contig) and if the occurrence of s is in the correct orientation. If there is a such occurrence, we can keep matching from the occurrence. If there are multiple such occurrences, then there are multiple ways to match s and we have to enumerate and check all possibilities. We call this a *jump-over-contig* step.

We try to explain all genomes with the minimum number of inversion endpoints, i.e., as few blocks as possible. We use a greedy block extension algorithm so that every block is maximal, and minimize the number of blocks. The block extension algorithm works as follows. A block starts from a single SNP. Each round we try to extend a block B, we pick a SNP s which is next to B in some genome, and test if all other genomes agree with the new block candidate Bs. If all genomes agree with Bs, then we extend B to Bs and start the next round. If there is any genome that does not agree with Bs, then we pick up another SNP s' which is next to B in some genome. If there is no such SNP that extends B in either forward or reverse direction, then we stop extending and output B as a block. Algorithms 5.2.1 and 5.2.2 outline the main idea of the algorithm.

Algorithm 5.2.1 GetBlock

```
1: $blocks \leftarrow \emptyset$
2: for each genome do
3:     for each SNP $s$ do
4:         if $s$ has not been included by any block then
5:             $B \leftarrow$ BlockExtension($s$)
6:             mark all SNPs in $B$ as included
7:             $blocks \leftarrow blocks \cup \{B\}$
8: return $blocks$
```

Algorithm 5.2.2 BlockExtension(block B)

```
1: for both forward and reverse direction do          /* reverse $B$ when needed          */
2:     for each genome $G$ do
3:         Let $s$ be the next SNP after the block $B$ in genome $G$
4:         if $Bs$ has not been tested then
5:             if all genomes agree with $Bs$ then
6:                 $B \leftarrow Bs$
```

The time complexity of the algorithm is determined by how fast we can determine if a genome agrees with a block. Assume B^* is returned by Algorithm 5.2.2 and there is no duplication, then a straightforward implementation will take $O(n|B^*|^2 \sum j_i)$ time, where n is the number of genomes, $|B^*|$ is the length of the block, and j_i is the product of all jump-over-contig enumerations on genome i. Note that duplications make it possible that a genome may agree with a small block in multiple ways in our algorithm, which theoretically increases the time complexity, and complicates the optimization. We choose not to optimize the implementation because our experiments show that a straightforward implementation yields a reasonable running time,[2] given the fact that duplications and jump-over-contigs do not occur very frequently.

In our algorithm, if a SNP s is absent in a genome G, then s will never make G disagree with a block. If s is next to a inversion endpoint, then s may appear in two different blocks. For example, genome G_1 has a SNP sequence $abcde$ and genome G_2 has ab and de but c is absent in G_2. Our algorithm will produce two blocks abc and cde, and we say these two blocks *overlap*. Duplications may also create overlapping blocks. For example, G_1 has SNP sequence $abcdef$ and G_2 has $abdcef$ and c, d elsewhere. Our algorithm will get two blocks $abcef$ and $abdef$. Therefore, after getting blocks, the summation of number of SNPs in all blocks, denoted as *increased* number of SNPs, is usually much more than the number of given SNPs.

[2]It takes 2 minutes for the *Bulkhorderia pseudomallei* dataset with 122 thousand SNPs and 26 strains

5.2.2 Inside of a Block with No Inversions

We now consider a single block, and the corresponding SNPs of the block in all genomes. The SNP order should be the same but there might be missing SNPs. For each SNP locus, we reconstruct the SNPs of internal nodes assuming there are only mutations and errors, and minimize the total weight of mutations (w_m) and errors (w_e) at the same time. This is a weighted small parsimony problem and can be solved by dynamic programming in linear time [30].

After inferring the SNPs of the internal nodes, we then compute if we can assign HGT. Let $1, 2, \cdots, b$ be the SNP indices of the block we consider. For each internal node t as a possible HGT destination, we define $S[i][j]$ as the minimum total weight considering SNPs $1, 2, \cdots, j$ assuming node t inherits SNP j from node i. Let p be the parent node of t, n the number of nodes, and $snp[k][j]$ SNP j of node k. We derive the recurrence relations for $S[\cdot][\cdot]$: $(i \neq p)$

$$S[p][1] = 0 \tag{5.1}$$

$$S[i][1] = w_x \tag{5.2}$$

$$S[p][j] = w_m \cdot I(snp[p][j] \neq snp[t][j]) + \min_k S[k][j-1] \tag{5.3}$$

$$S[i][j] = w_m \cdot I(snp[i][j] \neq snp[t][j]) + \min \begin{cases} S[i][j-1] \\ \min_k S[k][j-1] + w_x \end{cases} \tag{5.4}$$

$I(\cdot)$ is the indicator function in the above equations. Equation (5.3) represents the case that SNP j is not from an HGT, and Equation (5.4) represents the case that SNP

j is extending an existing HGT (top option in bracket) or starting a new HGT (bottom option in bracket). In Equation (5.3) and (5.4), k is enumerated from all possible source nodes, i.e., all other nodes that are not descentants of node t. We charge the weight of an HGT at the beginning of the HGT (Equation (5.4)), but do not charge at the end (Equation (5.3)). Note that Equation (5.4) also allows us to have mutations on a segment of HGT. For the leaf nodes, the recurrence relations are identical except each w_m is replaced by w_e. With the recurrence relations established, a standard dynamic programming technique with backtracking would be sufficient to assign mutations/HGTs optimally [7, 30]. There are nb entries in $S[\cdot][\cdot]$, and it takes $O(n)$ time to compute each entry. The time complexity is $O(n^2b)$ for a single node, and $O(n^3b)$ for all nodes. Let m be the increased number of SNPs, and the total time complexity is $O(n^3m)$.

5.2.3 Detection of Possible HGTs from the Out-Groups

If there are several consecutive mismatches of SNPs of a node and its parent node, it is likely that the segment is affected by some HGT. However, there might be no similar SNP segment in the given data, and we suspect it might be an HGT from an out-group. Suppose we try to assign an HGT from the out-group, since there are no known SNPs, we are free to create whatever SNPs we need to match the SNPs of the node we consider. If the weight of such HGT is a constant, it may lead to matching all the SNPs with an HGT from the out-groups. We borrow the idea of affine gap penalty in sequence alignment [30]. For the out-group HGT, we introduce the opening weight w_{oo} and the extending weight w_{oe}. Let $S[0][j]$ be defined the same as $S[i][j]$ but SNP j is inherited from the out-groups. The recurrence relation derived in Section 5.2.2 remain mostly the same except the enumeration of k in Equation (5.3) and (5.4) should include the the out-groups. We derive the recurrence relations for the out-groups as follows.

$$S[0][0] = w_{oo}$$

$$S[0][j] = \min \begin{cases} S[0][j-1] + w_{oe} \\ \min\limits_{k} S[k][j-1] + w_{oo} \end{cases}$$

These recurrence relations can be solved by standard dynamic programming with backtracking technique, and help assign sparse mismatches as mutations/errors and dense mismatches as out-group HGTs.

Sometimes the algorithm may assign two HGT events of the same segment to two nodes, and they inherit the HGT segment from each other. We consider this scenario as an evidence of an out-group HGT and we try to detect it. After assigning mutations/HGTs/errors by dynamic programming and backtracking, for each SNP of a node, we trace where the SNP is inherited from. A SNP within an HGT segment is inherited from the HGT source, and a SNP not in an HGT segment is inherited from the parent node. If there is no HGT from the out-groups, we should be able to trace all SNPs all the way to the root. If the tracing falls into a cycle, then we output the SNPs and involved nodes as an evidence of an out-group HGT. This algorithm also detects inheritance patterns that form a cycle by more than two nodes.

5.3 Experimental Results

We have implemented our algorithm in C/C++, denoted as HgtFinder. We have also implemented a simulator to generate simulated data and estimate the accuracy of HgtFinder. We also run HgtFinder on real data obtained from SNP analysis according to [16] of all available whole genomes (draft and finished) for 3 bacteria and 2 viruses.

5.3.1 Simulation

We use a model of random branching of lineages to simulate an evolutionary tree [36]. To simulate a tree of n strains, we start with a root and a branching event and its occurring time 0. When an event occurs, it splits a lineage into two, and brings two new branching events. For each new branching event, we draw a time interval from exponential distribution with a given branching rate, then add the time interval to the current time to make the occurring time of the new event. The time interval will also be the branch length of the corresponding edge. This process stops at the time the branching event which would generate the $(n + 1)$st strain is about to occur. The branch length of each edge which incidents to a leaf will be assigned as the time difference between the stop time and the branching time that generates the branch. Note that the summation of the branch length on the path from the root to all leaves will be the same.

After the evolutionary tree is generated, we then need to generate genome rearrangement events. In circular bacterial genomes, inversions tend to be symmetric to the origin of replication, i.e., the endpoints of the inversion are equally distant from the origin of replication [8, 13]. Dias et al. have published a program called SIB to simulate these symmetric inversions in bacterial chromosomes [10]. We use SIB to generate inversion events. SIB generates both symmetric and nonsymmetric inversions and the number of inversions on a branch is proportional to the branch length.

After the evolutionary tree and inversion events are generated, we then generate when and on which branches mutations and HGT should occur. For each edge, we generate a series of mutation events, and the time interval between a mutation and the next mutation is drawn from the exponential distribution with a given mutation rate. The series of mutations terminates when the time of the next mutation event is later than the time of the branching event that ends the edge. For each pair of edges, consider the time interval both edges appear. In the time interval, we generate a series of HGT events by the same way above we generate mutations, with a given HGT rate. After all events have been generated, we uniformly randomly generate the SNPs of the root. We then generate all SNPs of all nodes in the evolutionary tree with the given mutation/HGT events. The SNP position where each mutation takes place is assigned uniformly randomly. The position of each HGT is generated uniformly randomly on condition that it occurs within a homologous region, i.e., the SNP order/orientation should be the same in source and destination. Finally, on the leaf nodes, we generate sequencing errors and missing loci uniformly randomly with given error rate and missing rate, respectively.

There are many HGT/mutation events that cannot be detected easily, and some of them can never be detected. A mutation followed by another mutation or an HGT event on the same branch will get nullified and there is no way to detect it. The SNP sequence on source and destination of an HGT event may be identical or differ by only one SNP, then it has no effect or can be explained by a mutation, respectively. An HGT event may be followed by another HGT event on the same branch and get nullified. After simulated data is generated, we reduce events that we try to detect by discarding nullified events that can be determined with conditions listed above. By reducing nullified events, we can compute accuracy based on events that leave some evidence. However, we only detect nullifying effects that are all on the same branch. We do not detect nullifying effects that two or more branches are involved (e.g., a mutation followed by a branching event, then both following branches are affected by HGTs, nullifying the first mutation). There are still many scenarios in which HGT events cannot be detected: two or more HGT events may overlap and can be explained by a few mutations/errors, an inversion may separate an HGT into different blocks and we cannot detect it because we consider each block separately, etc. We do not eliminate these events when generating simulated data.

The weights of the events are set as $(w_e, w_m, w_x, w_{oo}, w_{oe}) = (2, 3, 5, 7, 1)$. By this setting, considering a segment that can be explained by either two (or more) mutations or one HGT from a node in the evolutionary tree, HgtFinder will choose one HGT. If a segment can be explained by either three (or more) errors or one HGT from a node in the evolutionary tree, HgtFinder will choose one HGT. For a segment that can be explained by one mutation or two errors, HgtFinder will not explain it by an HGT. Our first experiment shows that most HGT events are separated by inversions and cannot be detected. Therefore, in our second experiment, we do not generate inversions in order to focus on HGT events within a block. The default value of parameters are: average branch length = 20, 40 strains, 50 SNPs, mutation rate = 1% each SNP per branch length, HGT rate = 3% per branch length, error rate = 1% each SNP, and missing rate = 10%. Since most HGT events get partially nullified by other HGT events that overlap, an HGT event detected by HgtFinder is considered correct if it overlaps with an actual HGT event on the same branch. We denote *recall* as the number of correctly detected HGT events divided by the total number of actual HGT events, and *precision* as the number of correctly detected HGT events divided by the total number of predicted HGT events by HgtFinder. The average branch length is always fixed. In each set, we try 4 different values for a parameter, and all other parameters are fixed. For each parameter setup, we run the simulation 200 times, and compute the recall and precision.

Table 5.1 shows the result of our simulation. A higher mutation rate brings more diversity, and it reduces the similarity between source and destination of an HGT event. More diversity makes HGT events easy to detect, and improves recall. However, a higher mutation rate also increases the probability that we have consecutive mutations, which HgtFinder will explain as HGT, thus slightly decreases the precision. A higher HGT rate brings more overlapped HGT events, and makes HGT events difficult to detect, thus decreases the recall. A higher HGT rate also increases the precision, because it makes it easy for a detected HGT event to overlap with an actual HGT event. A lower missing rate results in better recall, and has little effect on the precision. The number of SNPs, or the size of a block, and the error rate, do not have significant impact on the accuracy. The number of strains can affect the accuracy either way. More strains with a fixed average branch length bring more diversity and improve the accuracy. However, more strains also bring bigger phylogenetic trees, longer simulated time, and more overlapped HGT events, which lower

Mutation rate	0.5%	1%	3%	6%
Recall	31.90%	40.97%	49.07%	49.32%
Precision	79.90%	78.99%	77.33%	76.23%
HGT rate	1%	3%	6%	10%
Recall	61.58%	49.07%	38.48%	31.31%
Precision	55.32%	77.33%	85.72%	90.27%
Missing rate	1%	5%	10%	20%
Recall	51.74%	49.53%	49.07%	46.64%
Precision	77.36%	76.98%	77.33%	76.04%
Error rate	0.1%	0.5%	1%	3%
Recall	48.36%	49.35%	49.07%	48.18%
Precision	77.31%	76.66%	77.33%	76.69%
# SNPs	10	20	50	100
Recall	49.25%	48.94%	49.07%	48.51%
Precision	76.55%	76.99%	77.33%	77.04%
# strains	10	20	40	80
Recall	34.74%	48.29%	49.07%	44.61%
Precision	48.96%	63.15%	77.33%	85.81%

Table 5.1: The accuracy of HgtFinder under different parameters. The default values of parameters are: average branch length = 20, 40 strains, 50 SNPs, mutation rate = 1% each SNP per branch length, HGT rate = 3% per branch length, error rate = 1% each SNP, and missing rate = 10%.

the recall. Therefore, more strains affect recall both ways, but obviously bring a better precision.

5.3.2 Real Data

We run HgtFinder on *Bacillus anthracis*, *Burkholderia mallei*, *Burkholderia pseudomallei*, *vaccinia virus* and *variola virus*. There are 122204 SNPs in *Burkholderia pseudomallie* dataset, and it takes 2 minutes for HgtFinder to run on a single Intel Xeon 5660 CPU with 2.8 GHz. We would expect *Burkholderia mallei* and *Bacillus anthracis* to show little recombination, i.e., few HGT events, and *Burkholderia pseudomallei* to show large amounts of recombination based on extensive work by Tal Pearson, the Keim laboratory, and others [19, 27, 58, 59]. *Vaccinia virus* is also expected to show high rates of HGT due to a complex history due to broad host range, high passage in domesticated animals and chick embryos, spiking cultures with cowpox and variola, scarification practices of vaccination that reintroduced *vaccinia virus* to nature many times, and mixing of multiple vaccinia strains in vaccine preparations [54]. In contrast, *variola virus* is much more homogeneous than *vaccinia virus*, and its evolution is thought to be a result of natural selection via human-to-human transmission. As a result much lower levels of recombination have been found [15].

We plot the number of events detected by HgtFinder as a function of the length of the branches. Figures 5.2 through 5.6 show the results. Note that blocks may overlap heavily because of duplications, some SNPs may be computed many times, and some mutations may get counted many times. Therefore we use the increased number of SNPs as a reference

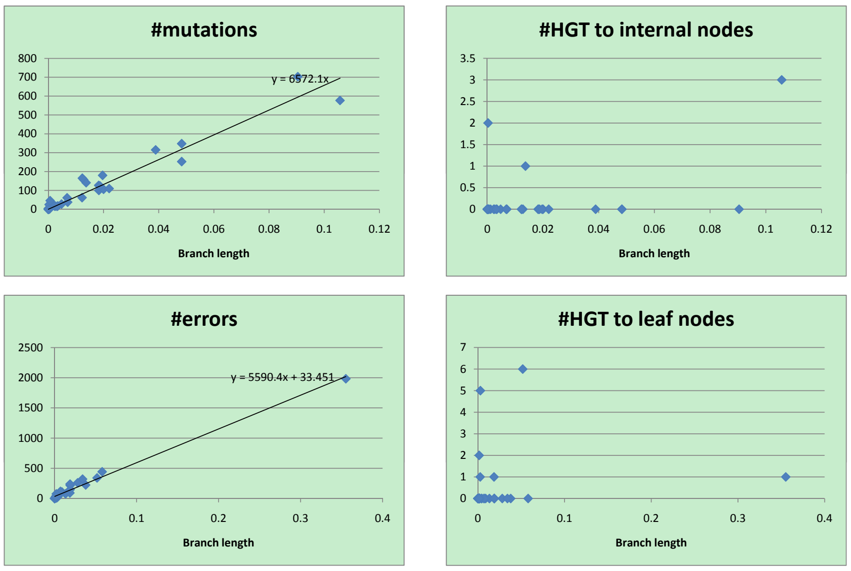


Figure 5.2: *Bacillus anthracis*, 34 strains, 8781 SNPs (increased).

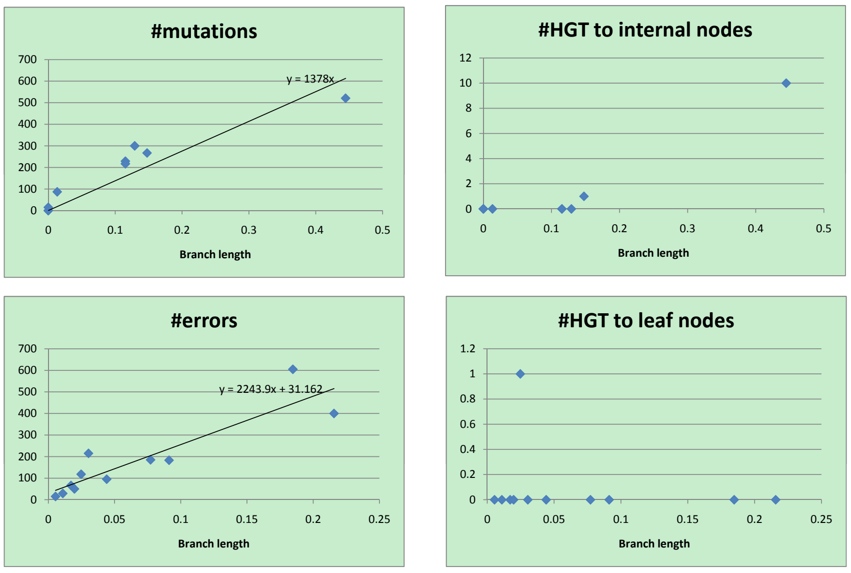


Figure 5.3: *Burkholderia mallei*, 11 strains, 3659 SNPs (increased).

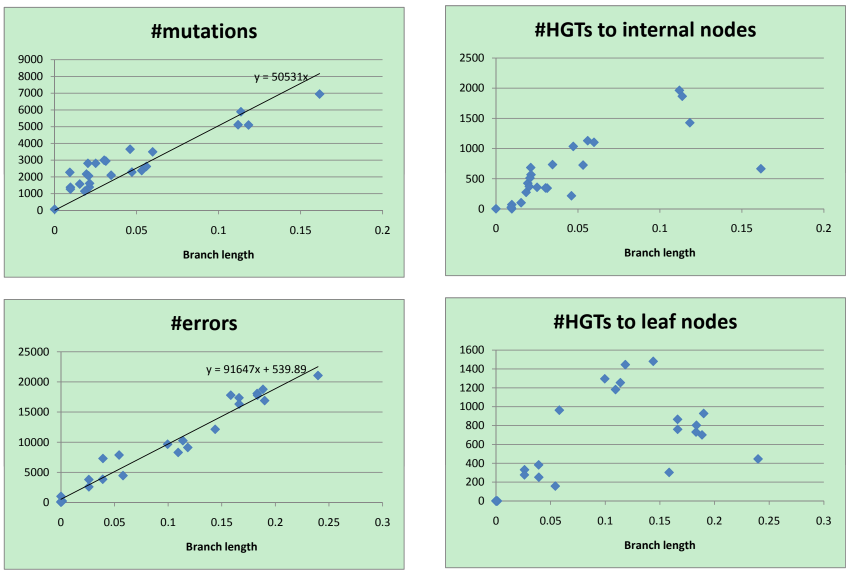


Figure 5.4: *Burkholderia pseudomallei*, 26 strains, 212174 SNPs (increased).

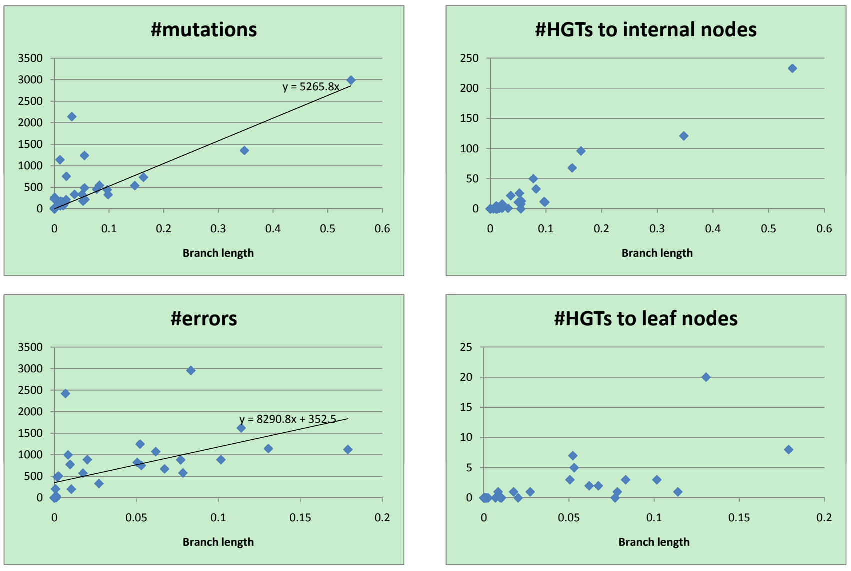


Figure 5.5: *Vaccinia virus*, 33 strains, 17562 SNPs (increased).

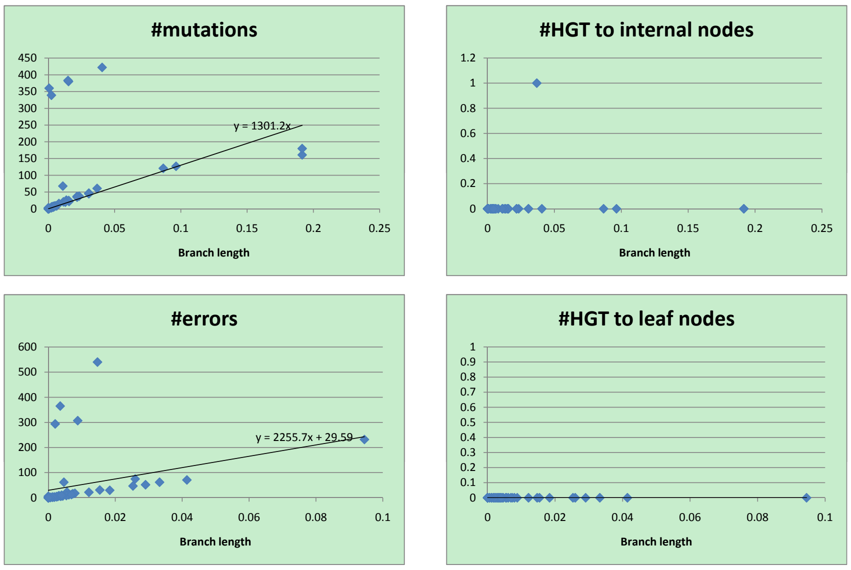


Figure 5.6: *Variola virus*, 49 strains, 2926 SNPs (increased).

of mutation counts. The branch length is calculated by the rearrangement distance, which we expect to be proportional to the evolutionary time [43]. We also expect the number of mutations to be proportional to the evolutionary time, thus proportional to the branch length, which is consistent with the plots. For leaf nodes, mutations are considered as errors. The number of errors should only be proportional to the number of SNPs. If we draw a linear trendline $y = ax + b$ where x is the branch length and y the number of errors and mutations, then the intercept b should represent the number of errors. Given the intercepts are small in our plots, most "errors" on the leaf nodes should be mutations. The difference of slopes between mutations and errors in the plots could be because the accuracy of branch length estimation is difference between internal nodes and leaf nodes. A few branches are outliers, however, showing more mutations than expected based on the branch length, which could be explained by the following. In *Burkholderia mallei* dataset, there are many blocks that overlap extremely heavily, and mutations in the overlapping regions get counted repeatedly. In the other 4 datasets, there are some regions that get partitioned into many single-SNP blocks by HgtFinder, and some HGT events fall into these regions. HgtFinder explains these HGT events by many mutations or errors, and it leads to some plots with extreme amounts of mutations and errors.

For homogeneous species like *Bacillus anthracis*, *Burkholderia mallei* and *variola virus*, there appears to be no relationship between HGTs and branch length, since so few HGTs have occurred. Even for the more heterogeneous *Burkholderia pseudomallei* and *vaccinia virus*, HGTs seem to have much weaker relationship to branch length than do mutations or errors. HGT may have less to do with evolutionary time (branch length) and more to do with ecological opportunity. Factors like co-infection or co-habitation in the environment with multiple strains or species could lead to more opportunities for HGT, as could the prevalence of genetic mechanisms for HGT like transposons or other mobile elements. There are more HGTs to internal nodes than to leaf nodes. We believe it is because the weight of an error is smaller than that of a mutation, so HgtFinder tends to assign errors on leaf nodes but HGTs on internal nodes.

We plot the number of mutation/HGT/error events of *Burkholderia pseudomallei* dataset in Figure 5.7 by Dendroscope [28] and outline the number of HGT events from the out-groups of each strain in Figure 5.8. There are 420 out of 29515 HGT events are considered from the out-groups and 260 of them are on the leaf nodes. The regions of HGT events from the out-groups in *Burkholderia pseudomallei* have good blast hits to transposon, phage, and plasmid sequences, prime candidates for HGT. Other good hits are to soil and water inhabiting microbes like Rhizobium, Pseudomonas, and other Burkholderia species, consistent with HGT occurring in soil and aquatic environments.

5.4 Conclusion

Using a definition of SNPs that facilitates scalable, whole genome analysis of dozens of strains, we designed and implemented an algorithm to do whole genome HGT detection. Our experimental results on simulated data show that there are many HGT events that cannot be detected, but the HGT events detected by our program are mostly true events. The experimental results on real sequence data also show that the number of HGTs we predict for 5 bacteria and viruses is consistent with expectations based on the literature.

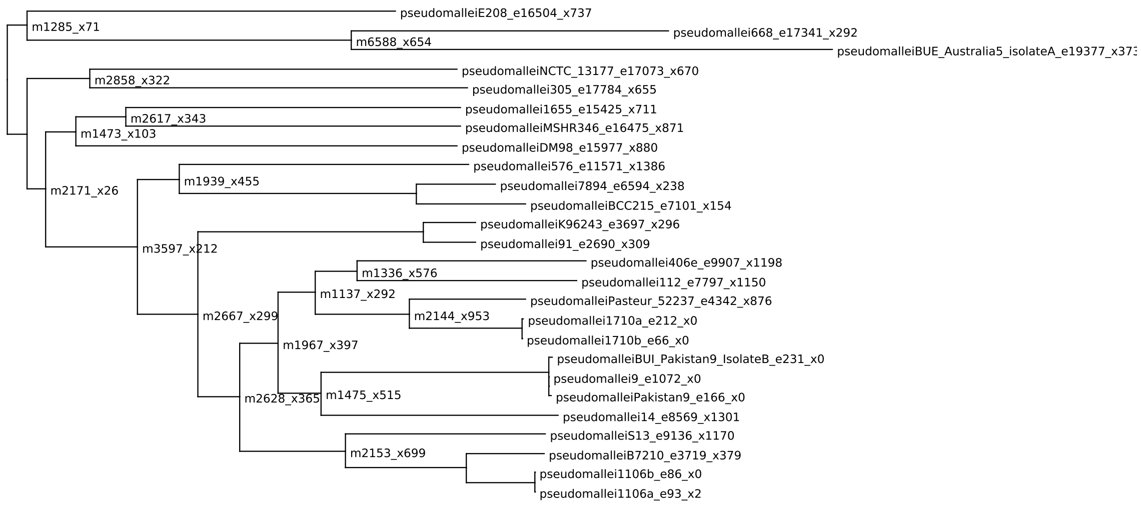

pseudomalleiE208_e16504_x737
m1285_x71
pseudomallei668_e17341_x292
m6588_x654
pseudomalleiBUE_Australia5_isolateA_e19377_x37
pseudomalleiNCTC_13177_e17073_x670
m2858_x322
pseudomallei305_e17784_x655
pseudomallei1655_e15425_x711
m2617_x343
pseudomalleiMSHR346_e16475_x871
m1473_x103
pseudomalleiDM98_e15977_x880
pseudomallei576_e11571_x1386
m1939_x455
pseudomallei7894_e6594_x238
m2171_x26
pseudomalleiBCC215_e7101_x154
pseudomalleiK96243_e3697_x296
m3597_x212
pseudomallei91_e2690_x309
pseudomallei406e_e9907_x1198
m1336_x576
pseudomallei112_e7797_x1150
m1137_x292
pseudomalleiPasteur_52237_e4342_x876
m2667_x299
m2144_x953
pseudomallei1710a_e212_x0
pseudomallei1710b_e66_x0
m1967_x397
pseudomalleiBUI_Pakistan9_IsolateB_e231_x0
pseudomallei9_e1072_x0
m2628_x365
m1475_x515
pseudomalleiPakistan9_e166_x0
pseudomallei14_e8569_x1301
pseudomalleiS13_e9136_x1170
m2153_x699
pseudomalleiB7210_e3719_x379
pseudomallei1106b_e86_x0
pseudomallei1106a_e93_x2

Figure 5.7: Evolutionary tree of *Burkholderia pseudomallei*. The internal nodes are labeled by the number of events, m for mutation, e for error, and x for HGT. The leaf nodes are labeled by the strain name follower by the number of events.

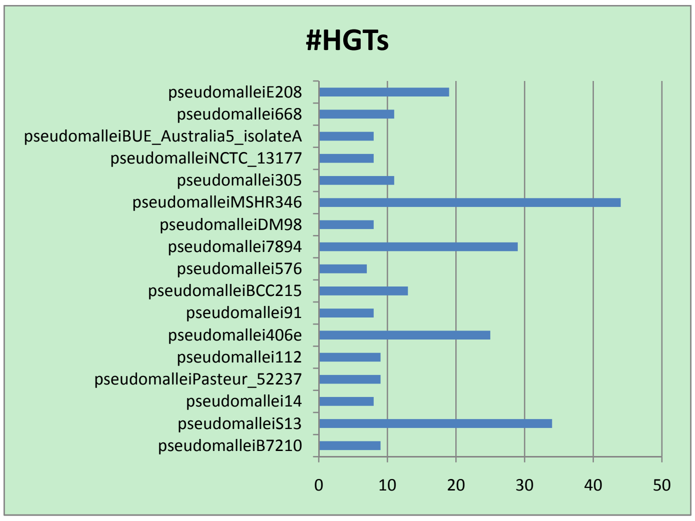


Figure 5.8: The number of HGTs from out-groups to leaf nodes in *Burkholderia pseudomallei* dataset.

Chapter 6

Future Work

For the tag SNP selection problem, it remains interesting if we can use less space in our tag SNP selection problem, or if we can compute fewer triplets/quartets. This actually has been improved by Liu et al. [44]. For the haplotype inference problem, we still wonder if we can lower the variance of the running time. Randomized algorithms are possible candidates. The recovery phase might be improved by generating smaller number of configurations within nuclear families. For the detection of horizontal gene transfer problem, we wonder if we can handle duplications and overlapped blocks better. We also wonder if we can avoid HGT regions being partitioned into many single-SNP blocks, which might hinder the detection of some HGTs.

Bibliography

[1] Gonçalo R. Abecasis, Stacey S. Cherny, and Lon R. Cardon. The impact of genotyping error on family-based analysis of quantitative traits. *European Journal of Human Genetics*, 9:130–134, 2001.

[2] Gonçalo R. Abecasis, Stacey S. Cherny, William O. Cookson, and Lon R. Cardon. Merlin — rapid analysis of dense genetic maps using sparse gene flow trees. *Nature Genetics*, 30(1):97–101, January 2002.

[3] Cornelis A. Albers, Tom Heskes, and Hilbert J. Kappen. Haplotype inference in general pedigrees using the cluster variation method. *Genetics*, 177(2):1101–1116, October 2007.

[4] Tatiana N. Badaeva, Daria N. Malysheva, Vitaly I. Korchagin, and Alexei P. Ryskov. Genetic variation and *De Novo* mutations in the parthenogenetic caucasian rock lizard *Darevskia unisexualis*. *PLoS ONE*, 3(7):e2730, July 2008.

[5] Eyal Baruch, Joel Ira Weller, Miri Cohen-Zinder, Micha Ron, and Eyal Seroussi. Efficient inference of haplotypes from genotypes on a large animal pedigree. *Genetics*, 172(3):1757–1765, March 2006.

[6] Christopher S. Carlson, Michael A. Eberle, Mark J. Rieder, Qian Yi, Leonid Kruglyak, and Deborah A. Nickerson. Selecting a maximally informative set of single-nucleotide polymorphisms for association analyses using linkage disequilibrium. *The American Journal of Human Genetics*, 74(1):106–120, 2004.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, second edition.* MIT press, 2001.

[8] Aaron E. Darling, István Miklós, and Mark A. Ragan. Dynamics of genome rearrangement in bacterial populations. *PLoS Genetics*, 4(7):e1000128, 2008.

[9] P. De Bakker et al. Transferability of tag SNPs in genetic association studies in multiple populations. *Nature Genetics*, 38(11):1298–1303, 2006.

[10] Ulisses Dias, Zanoni Dias, and João C. Setubal. A simulation tool for the study of symmetric inversions in baterial genomes. In E. Tannier, editor, *RECOMB Comparative Genomics*, LNBI 6398, pages 240–251. Springer, 2010.

[11] Julie A. Douglas, Michael Boehnke, and Kenneth Lange. A multipoint method for detecting genotyping errors and mutations in sibling-pair linkage data. *The American Journal of Human Genetics*, 66:1287–1297, 2000.

[12] Julie A. Douglas, Andrew D. Skol, and Michael Boehnke. Probability of detection of genotyping errors and mutations as inheritance inconsistencies in nuclear-family data. *The American Journal of Human Genetics*, 70(2):487–495, 2002.

[13] Jonathan A. Eisen, John F. Heidelberg, Owen White, and Steven L. Salzberg. Evidence for symmetricchromosomal inversions around the replication origin in bacteria. *Genome Biology*, 1(6):research 0011.1–0011.9, 2000.

[14] Hans Ellegren. Microsatellite mutations in the germline: Implications for evolutionary inference. *Trends in Genetics*, 16(12):551–558, December 2000.

[15] Joseph J. Esposito et al. Genome sequence diversity and clues to the evolution of Variola (smallpox) virus. *Science*, 313:807–812, 2006.

[16] Shea N. Gardner and Tom Slezak. Scalable snp analyses of 100+ bacterial or viral genomes. *Journal of Forensic Research*, 1(107), 2010.

[17] Dan Gusfield. Inference of haplotypes from samples of diploid populations: Complexity and algorithms. *Journal of Computational Biology*, 8(3):305–323, June 2001.

[18] B. V. Halldórsson et al. Optimal haplotype block-free selection of tagging SNPs for genome-wide association studies. *Genome Research*, 14:1633–1640, 2004.

[19] William P. Hanage, Christophe Fraser, and Brian G. Spratt. The impact of homologous recombination on the generation of diversity in bacteria. *Journal of Theoretical Biology*, 239:210–219, 2006.

[20] K. Hao, X. Di, and S. Cawley. LdCompare: rapid computation of single- and multiple-marker r^2 and genetic coverage. *Bioinformatics*, 23(2):252–254, 2007.

[21] Ke Hao. Genome-wide selection of tag SNPs using multiple-marker correlation. *Bioinformatics*, 23(23):3178–3184, 2007.

[22] Ke Hao, Cheng Li, Carsten Rosenow, and Wing Hung Wong. Estimation of genotype error rate using samples with pedigree information – an application on the GeneChip Mapping 10K array. *Genomics*, 84:623–630, 2004.

[23] Tobias Hill, Karl JV. Nordström, Mikael Thollesson, Tommy M. Säfström, Andreas KE. Vernersson, Robert Fredriksson, and Helgi B. Schiöth. SPRIT: Identifying horizontal gene transfer in rooted phylogenetic trees. *BMC Evolutionary Biology*, 10(42), 2010.

[24] William G. Hill. Estimation of linkage disequilibrium in randomly mating populations. *Heredity*, 33(2):229–239, 1974.

[25] William G. Hill. Tests for association of gene frequencies at several loci in random mating diploid populations. *Biometrics*, 31(4):881–888, 1975.

[26] D. Hinds et al. Whole-genome patterns of common DNA variation in three human populations. *Science*, 307(5712):1072–1079, 2005.

[27] Matthew T. G. Holden. Genomic plasticity of the causative agent of melioidosis, Burkholderia pseudomallei. *PNAS*, 101(39):14240–14245, 2004.

[28] Daniel H. Huson, Daniel C. Richter, Christian Rausch, Tobias Dezulian, Markus Franz, and Regula Rupp. Dendroscope: An interactive viewer for large phylogenetic trees. *BMC Bioinformatics*, 8(1):460, 2007.

[29] G. Johnson et al. Haplotype tagging for the identification of common disease genes. *Nature Genetics*, 29:233–237, 2001.

[30] Neil C. Jones and Pavel A. Pevzner. *An Introduction to Bioinformatics Algorithms*. MIT Press, 2000.

[31] Steven T. Kalinowski and Philip W. Hedrick. Estimation of linkage disequilibrium for loci with multiple alleles: basic approach and an application using data from bighorn sheep. *Heredity*, 87:698–708, 2001.

[32] Patrick J. Keeling and Jeffery D. Palmer. Horizontal gene transfer in eukaryotic evolution. *Nature Reviews Genetics*, 9(8):605–618, 2008.

[33] Motoo Kimura and James F. Crow. The number of alleles that can be maintained in a finite population. *Genetics*, 49:725–738, 1964.

[34] Katherin M. Kirk and Lon R. Cardon. The impact of genotyping error on haplotype reconstruction and frequency estimation. *European Journal of Human Genetics*, 10:616–622, 2002.

[35] Sergei L. Kosakovsky Pond, David Posada, Eric Stawiski, Colombe Chappey, Art F.Y. Poon, Gareth Hughes, Esther Fearnhill, Mike B. Gravenor, Andrew J. Leigh Brown, and Simon D.W. Frost. An evolutionary model-based algorithm for accurate phylogenetic breakpoint mapping and subtype prediction in HIV-1. *PLoS Comput Biol*, 5(11):e1000581, 11 2009.

[36] Mary K. Kuhner and Joseph Felsenstein. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Molecular Biology and Evolution*, 11(3):459–468, 1994.

[37] Eric S. Lander and Philip Green. Construction of multilocus genetic linkage maps in humans. In *Proceedings of the National Academy of Science of the United States of America*, volume 84 of *Genetics*, pages 2363–2367, April 1987.

[38] Jérémie JP. Lebrec, Hein Putter, Jeanine J. Houwing-Duistermaat, and Hans C. van Houwelingen. Influence of genotyping error in linkage mapping for complex traits — an analytic study. *BMC Genetics*, 9(57), 2008.

[39] Emmanuelle Lerat, Vincent Daubin, Howard Ochman, and Nancy A. Moran. Evolutionary origins of genomic repertoires in bacteria. *PLoS*, 3(5):e130, 2005.

[40] Jing Li and Tao Jiang. Efficient inference of haplotypes from genotypes on a pedigree. *Journal of Computational Biology*, 1(1):41–69, 2003.

[41] Jing Li and Tao Jiang. Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by integer linear programming. *Journal of Computational Biology*, 12(6):719–739, July 2005.

[42] Jing Li and Tao Jiang. A survey on haplotype algorithms for tightly linked markers. *Journal of Bioinformatics and Computational Biology*, 6(1):241–259, 2008.

[43] Yu Li, Darin S. Carrol, Shea N. Gardner, Matthew C. Walsh, Elizabeth A. Vitalis, and Inger K. Damon. On the origin of smallpox: Correlating variola phylogenics with historical smallpox records. *PNAS*, 104:15787–15792, 2007.

[44] Guimei Liu, Yue Wang, and Limsoon Wong. FastTagger: an efficient algorithm for genome-wide tag SNP selection using multi-marker linkage disequilibrium. *BMC Bioinformatics*, 11(66), 2010.

[45] Lan Liu and Tao Jiang. Linear-time reconstruction of zero-recombinant mendelian inheritance on pedigrees without mating loops. *Genome Informatics*, 19:95–106, 2007.

[46] Lan Liu and Tao Jiang. A linear-time algorithm for reconstructing zero-recombinant haplotype configuration on pedigrees without mating loops. *Journal of Combinatorial Optimization*, 19:217–240, 2008.

[47] Lan Liu, Yonghui Wu, Stefano Lonardi, and Tao Jiang. Efficient algorithms for genome-wide tagSNP selection across populations via linkage disequilibrium criterion. In *6th Annual International Conference on Computational Systems Bioinformatics*, pages 67–78, 2007.

[48] R. Mägi, L. Kaplinski, and M. Remm. The whole genome tagSNP selection and transferability among hapmap populations. *Pacific Symposium on Biocomputing*, 11:535–543, 2006.

[49] Jonathan Marchini1, David Cutler, Nick Patterson, Matthew Stephens, Eleazar Eskin, Eran Halperin, Shin Lin, Zhaohui S. Qin, Heather M. Munro, Gonçalo R. Abecasis, and Peter Donnelly. A comparison of phasing algorithms for trios and unrelated individuals. *The American Journal of Human Genetics*, 78:437–450, March 2006.

[50] Valentina Moskvina, Nick Craddock, Peter Holmans, Michael J. Owen, and Michael c. O'Donovan. Effects of differential genotyping error rate on the type I error probability of case-control studies. *Human Heredity*, 61:55–64, 2006.

[51] Tianhua Niu, Zhaohui S. Qin, Xiping Xu, and Jun S. Liu. Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms. *The American Journal of Human Genetics*, 70(1):157–169, 2002.

[52] Jeffrey R. O'Connell and Daniel E. Weeks. Pedcheck: A program for identification of genotype incompatibilities in linkage analysis. *The American Journal of Human Genetics*, 63:259–266, 1998.

[53] Timothy M. Olson, Thao P. Doan, Nina Y. Kishimoto, Frank G. Whitby, Michael J. Ackerman, and Lameh Fananapazir. Inherited and *de novo* mutations in the cardiac actin gene cause hypertrophic cardiomyopathy. *Journal of Molecular and Cellular Cardiology*, 32(9):1687–1694, September 2000.

[54] John D. Osborne et al. Genomic differences of Vaccinia virus clones from Dryvax smallpox vaccine: The Dryvax-like ACAM2000 and the mouse neurovirulent Clone-3. *Vaccine*, 25:8807–8832, 2007.

[55] Cristos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[56] H. J. Park, G. Jin, and L. Nakhleh. Algorithmic strategies for estimating the amount of reticulation from a collection of gene trees. In *9th Annual International conference on Computational Systems Bioinformatics*, pages 114–123. Life Science Society, 2010.

[57] N. Patil et al. Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. *Science*, 294(5547):1719–1723, 2001.

[58] Talima Pearson et al. Phylogeographic reconstruction of a bacterial species with high levels of lateral gene transfer. *BMC Biology*, 7(78), 2009.

[59] Talima Pearson, Richard T. Okinaka, Jeffrey T. Foster, and Paul Keim. Phylogenetic understanding of clonal populations in an era of whole genome sequencing. *Infection, Genetics and Evolution*, 9(5):1010–1019, 2009.

[60] Dajun Qian and Lars Beckmann. Minimum-recombinant haplotyping in pedigrees. *The American Journal of Human Genetics*, 70(6):1434–1445, 2002.

[61] Z. Qin, S. Gopalakrishnan, and G. Abecasis. An effient comprehensive search algorithm for tagSNP selection using linkage disequilibrium criteria. *Bioinformatics*, 22(2):220–225, 2006.

[62] Antonis Rokas, Barry L. Williams, Nicole King, and Sean B. Carroll. Genome-scale approaches to resolving incongruence in molecular phylogenies. *Nature*, 425(23):798–804, 2003.

[63] John Maynard Smith, Christopher G. Dowson, and Brian G. Spratt. Localized sex in bacteria. *Nature*, 349(3):29–31, 1991.

[64] Eric Sobel and Kenneth Lange. Descent graphs in pedigree analysis: Applications to haplotyping, location scores, and marker sharing statistics. *Human Genetics*, 58:1323–1337, 1996.

[65] Eric Sobel, Kenneth Lange, Jeffrey R. O'Connell, and Daniel E. Weeks. Haplotyping algorithms. In Terry Speed and Michael S. Waterman, editors, *Genetic Mapping and DNA Sequencing*, volume 81 of *IMA Volumes in Mathematics and its Applications*, pages 89–110. Springer-Verlag, 1996.

[66] Eric Sobel, Jeanette C. Papp, and Kenneth Lange. Detection and integration of genotyping errors in statistical genetics. *The American Journal of Human Genetics*, 70(2):496–508, 2002.

[67] Alexandros Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.

[68] Matthew Stephens, Nicholas J. Smith, and Peter Donnelly. A new statistical method for haplotype reconstruction from population data. *The American Journal of Human Genetics*, 68(4):978–989, 2001.

[69] Matthew Stephens, Nicholas J. Smith, and Peter Donnelly. A new statistical method for haplotype reconstruction from population data. *The American Journal of Huma Genetics*, 68:978–989, April 2001.

[70] D. Stram et al. Choosing haplotype tagging snps based on unphased genotype data using a preliminary sample of unrelated subjects with an example from the multiethnic cohort study. *Human Heredity*, 55(1):27–36, 2003.

[71] Pradip Tapadar, Saurabh Ghosh, and Partha P. Majumder. Haplotyping in pedigrees via a genetic algorithm. *Human Heredity*, 50(1):43–56, 2000.

[72] The Internaltional HapMap Consortium. The international HapMap project. *Nature*, 426:789–796, December 2003.

[73] Alun Thomas and Chris Canning. Simulating realistic zero loop pedigrees using a bipartite prüfer code and graphical modelling. *Mathematical Medicine and Biology*, 21(4):335–345, 2004.

[74] Shuang Wang, Kenneth K. Kidd, and Hongyu Zhao. On the use of DNA pooling to estimate haplotype frequencies. *Genetic Epidemiology*, 24(1):74–82, 2003.

[75] Wei-Bung Wang and Tao Jiang. A new model of multi-marker correlation for genome-wide tag SNP selection. *Genome Informatics*, 21:27–41, 2008.

[76] Wei-Bung Wang and Tao Jiang. Efficient inference of haplotypes from genotypes on a pedigree with mutations and missing alleles. In G. Kucherov and E. Ukkonen, editors, *Combinatorial Pattern Matching*, LNCS 5577, pages 353–367. Springer-Verlag Berlin Heidelberg, 2009.

[77] Wei-Bung Wang and Tao Jiang. Inferring haplotypes from genotypes on a pedigree with mutations, genotyping errors and missing alleles. *Journal of Bioinformatics and Computational Biology*, 9(2):339–365, 2010.

[78] Oscar Westesson and Ian Holmes. Accurate detection of recombinant breakpoints in whole-genome alignments. *PLoS Comput Biol*, 5(3):e1000318, 03 2009.

[79] Michael Wirtenberger, Kari Hemminki, Bowang Chen, and Barbara Burwinkel. SNP microarray analysis for genome-wide detection of crossover regions. *Human Genetics*, 117:389–397, 2005.

[80] Jing Xiao, Lan Liu, Lirong Xia, and Tao Jiang. Fast elimination of redundant linear equations and reconstruction of recombination-free mendelian inheritance on a pedigree. In *18th Annual ACM-SIAM Symposium on Descrete Algorithms*, pages 655–664, 2007.

[81] Yaning Yang, Jingshan Zhang, Josephine Hoh, Fumihiko Matsuda, Peng Xu, Mark Lathrop, and Jurg Ott. Efficiency of single-nucleotide polymorphism haplotype estimation from pooled DNA. In *Proceedings of the National Academy of Science of the United States of America*, volume 100, pages 7225–7230, 2002.

[82] Kui Zhang, Fengzhu Sun, and Hongyu Zhao. HAPLORE: A program for haplotype reconstruction in general pedigrees without recombination. *Bioinformatics*, 21(1):90–103, 2005.

[83] Kun Zhang and Li Jin. HaploBlockFinder: Haplotype block analyses. *Bioinformatics*, 19(10):1300–1301, 2003.

[84] Guohua Zou, Deyun Pan, and Hongyu Zhao. Genotyping error detection through tightly linked markers. *Genetics*, 164:1161–1173, 2003.