Algorithms for Analyzing Intraspecific Sequence Variation

Srinath Sridhar

CMU-CS-07-168

December 19, 2007

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Guy E. Blelloch, co-chair
Russell Schwartz, co-chair
R. Ravi
Eran Halperin, International Computer Science Institute, Berkeley.

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant no.IIS-0612099. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity

Keywords: discrete algorithms, computational genomics, phylogenetic tree reconstruction, population substructure, single nucleotide polymorphisms (SNPs), copy number polymorphisms (CNPs)

Abstract

Analysis of human genetic variation has gained significant momentum due to the success of many large-scale sequencing projects. Within the last five years, about 4 million single nucleotide polymorphisms (SNPs) have been genotyped over hundreds of individuals belonging to several ethnic groups. Recently, researchers have made significant progress in detecting and cataloging copy number polymorphisms (CNPs) as well. These large-scale efforts have now made it possible to analyze genetic variation within a single species, mainly humans, at very fine-scales. In our work, we address a fundamental question: how can we use these genetic differences to make inferences about the recent history of a species and its genome?

In this work, we primarily focus on two methods to analyze SNP variation data within humans. We develop maximum parsimony phylogenetic tree reconstruction algorithms that are specifically catered to work on SNP data. Such a phylogeny should cluster closely related individuals (perhaps an ethnic group) together. Therefore, these techniques are widely used to answer questions of human migration. Recently, these methods have also been applied to find regions in the human genome that are rapidly evolving. A second method to understand genetic variation is to directly cluster the SNP data based on the property that individuals within a cluster would share similar genotypes (and phenotypes).

Mathematically, we work with two variants of the phylogeny reconstruction problem, both of which are NP-complete. The first variant is equivalent to finding a Steiner minimum tree on a hypercube and the second is a generalization of this problem. We solve the two variants in polynomial time when the size of the phylogeny (Steiner minimum tree) is 'small' (near-perfect). We also develop integer linear program based algorithms that find the optimal solution fast in practice. We provide extensive empirical analysis to demonstrate the methods' usefulness on large-scale genomic data. For the clustering part, we work with two related problems. We solve the first problem by finding the max-cut of a graph, a well-known NP-complete problem. We can show that if the graph is generated with the clusters 'planted', then our algorithm returns the max-cut in polynomial time given enough data. The second problem is more general in that individuals (vertices) can fractionally belong to either of the clusters. We solve this problem by attempting to maximize the likelihood function using an initial solution that we show to be analytically correct under some theoretical assumptions.

Acknowledgments

Firstly, many thanks to my advisors, Guy Blelloch and Russell Schwartz and my internal thesis committee member R. Ravi. It has been fantastic to work with them and I am especially thankful to them for always providing very good advice while giving me incredible academic independence. I can not thank Eran Halperin enough for working closely with me right from my first year, motivating several problems on this thesis, hosting me for Summer 2006 at Berkeley and for all his advice – both technical and otherwise. Thanks to Ira Hall, Michael Wigler, Lakshmi Muthuswamy and Jonathan Sebat (all of Cold Spring Harbor Labs) for hosting me, advising me and working with me on copy number variation during Summer 2005. Thanks to Kedar Dhamdhere, Ashutosh Garg and Varun Kacholia for hosting me during Summer 2007 at Google. *Heaps* of thanks to my undergraduate advisor Vijaya Ramachandran, for spending immense time and effort in working with me closely and for recommending me for graduate school.

Thanks to Sharon Burks, Susan Hrishenko, Catherine Copetas and Deborah Cavlovich for all their help over the years. Special thanks to Randy Pausch for his brilliant open-house lecture that got me extremely excited about graduate school in general and Carnegie Mellon specifically.

Thanks to all my friends and collaborators! Here are some who have helped me with my academic work: Umut Acar, Hubert Chan, Victor Chen (UT-Austin/MIT), Rezaul Alam Chowdhury (UT-Austin), Jonathan Derryberry, Jason Ernst, Ganeshkumar Ganapathy (UT-Austin), Daniel Golovin, Severin Hacker, Krishna Prasanna Jagannathan (MIT), Hetunandan Kamichetty, Jude Kendall (CSHL), Gad Kimmel (Berkeley), Arun Krishnaswamy, Fumei Lam (MIT/Brown), Balakrishnan Narayanaswamy, Seth Pettie (UT-Austin/UMich-Ann Arbor), Pradeep Ravikumar, Sriram Sankararaman (Berkeley), Vyas Sekar, Runting Elaine Shi, Mohan Sridharan (UT-Austin), Sridhar Srinivasan (UT-Austin/MSR), Ming-Chi Tsai, Virginia Vassilevska, Shobha Venkataraman, Chengwen Chris Wang, Noam Zeilberger and the Schwartz lab members.

to mom, dad and krishna,

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Background

Over the last five years, human genetics has been greatly influenced by the rapid pace at which genetic variation has been discovered, annotated and typed over hundreds of individuals belonging to many different ethnicities. The first major breakthrough came from the human genome project [21, 37], which provided a nearly complete genome sequence for the first time. Towards the end of the project, researchers had already started to investigate the extent of genome variation present within humans. The most frequent form of variation, *or polymorphism*, was attributed to single base changes. These are called *single nucleotide polymorphisms (SNPs)*. The next major step towards understanding the human genome came from the ambitious *International Haplotype Map (HapMap)* project [75] that was aimed at discovering and typing all the common SNPs in the human genome over hundreds of individuals. As the *HapMap* project was being completed, research groups shifted focus towards finding human genome variation caused due to large segmental deletions or duplications called *copy number polymorphisms (CNPs)*. These massive research efforts led by multi-national consortia have now made it possible to conduct disease-association tests that attempt to identify genetic causes for diseases. Over the last year, several research groups have found SNPs and CNPs linked to many diseases, such as type 2 diabetes, autism, rheumatoid arthritis etc (see for example [36, 84, 99]). In parallel, we have also been able to understand how the human genome is shaped by evolutionary forces over very fine scales (e.g. [88]).

The analysis of these large data-sets require sophisticated algorithms that are efficient, accurate and robust to noise. This thesis develops several such algorithms. The first part of the thesis introduces several new algorithms that reconstruct a *phylogenetic tree*, a family tree that describes how the underlying genomes are inter-related. The second part of the thesis develops clustering algorithms that finds ethnic ancestries for the input individuals.

1.1 Single Nucleotide Polymorphisms (SNPs)

It is well known that human DNA sequences are largely identical. Indeed, more than 99% of the DNA sequence will be the same between a pair of individuals. While the remainder might seem a small fraction, considering that the DNA sequence is about 3 billion base-

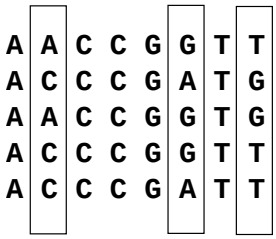


Figure 1.1: An illustration of SNPs. There are five individuals (one strand of one chromosome) and several sites of the DNA sequence, three of which show variation (SNPs).

pairs long, it still leaves us with millions of sites at which the human DNA shows variation. The most predominant and arguably the simplest variation is in the form of a single base change at a single position on the genome. For instance, some individuals might have a nucleotide A and others nucleotide G at the exact same site. Such variations are called single nucleotide polymorphisms (SNPs) and are illustrated in Figure 1.1.

SNPs hold incredible promise along at least two avenues of research. One is to identify, characterize and understand the evolutionary processes that produce such changes in the DNA sequences. The other of course is to determine genetic causes of disease and therefore speed up drug discovery as well. This promise has driven researchers to produce several major results in the last five years.

The first major project that aimed to catalog SNP variation was led by the International HapMap consortium [75], a collaboration between many research institutions world wide. Over a period of about 5 years, the organization analyzed about 120 individuals belonging to four ethnic groups at around 4 million SNP sites. Perlegen [68] also produced millions of SNPs belonging individuals spanning 3 ethnicities. These projects enabled researchers to understand variation in the human genome at scales never before possible.

A large number of problems arose from these projects as well and several algorithms that attempted to solve them. The algorithms ranged from mathematically elegant, to theoretically involved and of course, those that were found to work well in practice. This thesis contains such a diverse collection of algorithms as well.

Our work. In this work, we develop algorithms to characterize and analyze SNP variation present within the human genome. Such studies are important since SNPs can directly influence physical traits. One popular example is a single SNP on chromosome 2 that determines if a person is lactose tolerant or not. SNP variation can be studied either at the chromosome-level or at the population-level. At the chromosome-level, for example a researcher may wish to study how an ancestral lactose intolerant chromosome was converted to one that was lactose tolerant. To aid such analysis, computational tools are needed to infer the mutational history around any locus of the human genome. The first part of this thesis formalizes several problems, develops algorithms and applies them to answer such chromosome-level analysis problems. We can also analyze variation at the population-level. For instance most of the East Asians are lactose intolerant while most of the Europeans are lactose tolerant. While mutations could have happened in a common

ancestral population, segregation following the mutation can skew the distribution of the mutant genes, creating population-specific traits. The second part of the thesis develops efficient and accurate algorithms to tackle such problems. Since SNP variation at either level produces distinctive features, this thesis aims to obtain a more comprehensive analysis by tacking computational challenges encountered along both avenues.

1.2 Basic Definitions

We now provide some preliminary definitions related to single nucleotide polymorphisms (SNPs). A glossary of some basic definitions can be found in the Appendix in Section A.

Humans contain 23 pairs of chromosomes each of which is composed of two strands of DNA sequence. The strands are complimentary and therefore knowing one strand gives complete information about the other. When we use SNP sequences, we will therefore use information corresponding only to one of the two strands of a chromosome.

SNPs are bi-allelic meaning when a variation occurs there are only two forms observed. For instance in Figure 1.1, the second site contains only A and C. Therefore in the SNP datasets there would not be a SNP site i for which three different individuals have three different bases at i.

The main evolutionary process relating to this thesis is that of mutations, specifically, *point mutations*. These are mutations that change a DNA nucleotide from one base to another resulting in the creation of a SNP site. Since a single mutation can at most produce one new allele, to have three alleles at a site, we need at least 2 mutations, i.e. at least one *recurrent mutation*. Recurrent mutations are unlikely to occur, and this is the reason that current technology only looks for two (out of the four possible) bases.

An allele is called a *major allele* if it is more common in the population and *minor allele* otherwise.

As stated above, human chromosomes occur in pairs and 22 of the 23 chromosomes are *homologous pairs*. It is technologically prohibitive to assay the homologous chromosomes individually. Therefore, experiments produce results that are simply counts of an allele across a pair of homologous chromosomes. Without loss of generality, we can assume that the experiments produce minor allele counts.

At a SNP site, an individual can therefore contain one of three possibilities. Two minor alleles (*homozygous minor*), two major alleles (*homozygous major*) or one major and one minor allele (*heterozygous*). Such assays of allele counts from two chromosomes combined are called *genotypes*. Clearly though, in reality, the sequences come from each chromosome separately. Sequences from a single chromosome are referred to as *haplotypes*. Figure 1.2 illustrates the difference between genotypes and haplotypes. The figure also shows how an $\{A, C, G, T\}$ DNA sequence can be converted to a $\{0, 1\}$ sequence without changing the pair-wise Hamming distances. Notice that to do so, we exploit the fact that SNPs are bi-allelic.

In all the input data-sets that we will be considering for this thesis, the rows correspond to taxa (individuals) and the columns to the variations (characters). Therefore, for SNP variation, the input is represented as an $n \times m$ matrix. Each entry of the matrix can be

Ind 1:	ACGTAC ACGAGT	000000 000111	000111
Ind 2:	TGATAC ACGAGT	111000 000111	111111
Ind 3:	TGATAC TGATAC	111000 111000	222000
	(a)	(b)	(c)

Figure 1.2: (a) DNA sequence over SNP sites on homologous chromosomes for three individuals (b) binary matrix representation of the SNP data with '0' for the major allele and '1' for the minor allele (c) genotypes for the three individuals

called an allele or a state interchangeably. Note that using the above definitions, the input matrix is binary when the taxa are presented as haplotypes and is $\{0, 1, 2\}$ when the taxa are presented as genotypes.

1.3 Outline

In this work, we develop discrete algorithms for analyzing human SNP data-sets. In the first part of the thesis we focus on the problem of reconstructing the most parsimonious phylogenetic tree, i.e., the smallest 'family tree' that inter-connects all the input SNP sequences. In the second part we are interested in detecting *population substructure*, i.e. clustering individuals based on their ancestral ethnicity. We now provide more formal definitions and a detailed outline of the two parts.

1.3.1 Phylogenetic Tree Reconstruction

We have alluded to a phylogeny as a family tree that interconnects the input sequences. We now develop a formal framework. We begin with two equivalent definitions of a phylogeny and we find it convenient to use each at varying times depending on the problem that is required to be solved.

Definition 1 *A* phylogeny *T for a binary (haplotype) matrix I is a tree $T(V, E)$ and label function l with the following properties $R(I) \subseteq l(V(T))$ and for all $(u, v) \in E(T)$, $H(l(u), l(v)) = 1$ where H is the Hamming distance.*

Definition 2 *A* phylogeny *T for a binary (haplotype) matrix I is a tree $T(V, E)$ and label function l with the following properties: $R(I) \subseteq l(V(T))$ and $l(\{v \in V(T) | degree(v) \leq 2\}) \subseteq R(I)$. That is, every input taxon appears in T and every leaf or degree-2 vertex is an input taxon.*

Note that the above definitions of a phylogeny are specific to a haplotype input matrix. The following two definitions provide some useful terminology when discussing a phylogeny in general.

Definition 3 *A vertex v of phylogeny T is* terminal *if $l(v) \in R(I)$ and* Steiner *otherwise.*

Definition 4 *For a phylogeny T, $\texttt{length}(T) = \sum_{(u,v)\in E(T)} d(l(u), l(v))$, where d is the Hamming distance.*

Definition 5 *A phylogeny is called* most parsimonious (optimum) *phylogeny if its length is minimized.*

We will assume that all the input sites are varying, i.e., if the input consists of haplotypes, then all characters have both states $\{0, 1\}$ and if the input consists of genotypes then all characters either contain state 1 or contain both 0 and 2. Therefore the length of an optimum phylogeny is at least m. This leads to the following two definitions.

Definition 6 *For a phylogeny T on input I, $\texttt{penalty}(T) = \texttt{length}(T) - m$; $\texttt{penalty}(I) = \texttt{penalty}(T^{opt})$, where T^{opt} is any optimum phylogeny on I.*

Definition 7 *A phylogeny T is called* q-near-perfect *if $\texttt{penalty}(T) = q$ and* perfect *if $\texttt{penalty}(T) = 0$.*

Note that in an optimum phylogeny, no two vertices share the same label. Therefore, we can equivalently define an edge of a phylogeny as (t_1, t_2) where $t_i \in \{0, 1\}^m$. Since we will always be dealing with optimum phylogenies, we will drop the label function $l(v)$ and use v to refer to both a vertex and the taxon it represents in a phylogeny.

We now extend the above definitions to the case when the input consists of genotypes as opposed to haplotypes. The input is an $n \times m$ matrix G, where each row $g_i \in \{0, 1, 2\}^m$ represents a (genotype) taxon and each column represents a character. The output is a $2n \times m$ matrix H in which each row $h_i \in \{0, 1\}^m$ represents a haplotype. Furthermore corresponding to every taxon $g_i \in R(G)$, there are two taxa $h_{2i-1}, h_{2i} \in R(H)$ with the following properties (consistency):

- if $g_{i,c} \neq 2$ then $h_{2i-1,c} = h_{2i,c} = g_{i,c}$
- if $g_{i,c} = 2$ then $h_{2i-1,c} \neq h_{2i,c}$

We will refer to a pair of haplotypes as *consistent* with a genotype if the the above properties hold. A phylogeny for the genotype matrix G is simply a phylogeny on the haplotype matrix H that satisfies the consistency property. The definitions of q-near-perfect and perfect remain identical to the haplotype case. Therefore the problem of reconstructing the most parsimonious phylogeny for G involves finding H such that the length of the optimal tree on H is minimized. The problem is therefore a generalization of phylogeny reconstruction from haplotypes.

We now provide an informal introduction to the problem of *haplotype inference*. Note that, in reality, each row $g_i \in G$ is in fact produced by a pair of haplotypes. The problem of *haplotype inference* is to find the true pair of haplotype sequences for each input genotype sequence. Since there could be many pairs of haplotypes consistent with an input genotype, the problem can be posed as an optimization only under a model. We are now ready to provide an outline of the five related problems that we address in the first part.

In Chapter 2, we consider the problem of reconstructing the most parsimonious phylogenetic tree for $n \times m$ binary (haplotype) input matrix I. The algorithms are characterized by q, which is the penalty of the input matrix I. We initially develop a simple algorithm

that already improves the previous best known theoretical running time. We then present a more involved algorithm with a further improved run-time bound of $O(21^q + 8^q nm^2)$. This solves an important open problem in the area since it shows that the maximum parsimony phylogeny reconstruction problem is *fixed parameter tractable* in parameter q.

In Chapter 3, we continue to develop algorithms for finding the most parsimonious phylogenetic tree for haplotype matrix I. The main focus here is to obtain efficient, practical algorithms for the case when the penalty of the input matrix I could be large. We develop several theoretical pre-processing techniques that enable us to reduce the problem size without losing optimality. We then show an integer linear program formulation that runs very fast in practice, typically faster than even the most popular heuristic.

In Chapter 4, we consider the same problem as Chapter 2 except that the input is presented as genotypes. This considerably increases the difficulty of the problem. Researchers were able to make progress on this problem by developing polynomial time algorithms for the special case when $q = 0$ and later for $q = 1$, but the general problem remained open. We solve the general problem by developing an algorithm with run-time $nm^{O(q)}$ which is a polynomial bound for any constant q. In experiments we demonstrate the sizes of the input for which the algorithm is practically feasible. A very important application of this problem is that of *haplotype inference* i.e., determining the set of haplotypes given the genotypes using the maximum parsimony phylogeny model.

In Chapter 5, we consider the same problem as in Chapter 3 except that the input can be genotypes. We adapt the techniques presented previously to obtain a new algorithm for this problem. We show that the accuracy in terms of phylogeny sizes is significantly better than what could be obtained using currently leading software packages for haplotype inference.

In Chapter 6, we apply the techniques presented to construct phylogenies on a genome-wide-scale. We used the data published by HapMap [75] to construct around 4 million phylogenies around every locus of the genome. We found that the correlation of phylogeny sizes across different ethnic groups was very high. We also discovered that phylogenies over different structural elements of the genome possessed distinctive patterns.

1.3.2 Population Substructure

The second part of the thesis deals with clustering individuals based on their ancestral ethnicities using SNP variation data. This part contains two related chapters.

Chapter 7 considers the problem of unsupervised clustering, where we wish to cluster a given set of SNP sequences into two sub-populations with no prior knowledge of the distribution of the two sub-populations. This assumes that the input individuals belong to exactly one of two populations with no mixture. We now define the problem mathematically.

The input to the problem is an $n \times m$ genotype matrix I. We assume that there are two ancestral populations each characterized by the minor allele frequency in each of the SNPs. Thus, a population i is defined by an m-dimensional vector $\vec{p^i} = (p^i_1, \ldots, p^i_m)$, where p^i_j represents the minor allele frequency of population i in position j. We define the distance

between two sub-populations i, i' as:

$$d(i, i') = \sqrt{\sum_j (p_j^i - p_j^{i'})^2}$$

The problem asks for a classification $\theta : R(I) \to \{0, 1\}$, that assigns every individual to a particular sub-population. Let $\hat{\theta}$ be the correct classification. Our objective is to minimize the number of errors made by the algorithm, that is, we would like to minimize $|\{r \in R(A) \mid \theta(r) \neq \hat{\theta}(r)\}|$. We formulate the problem as a graph-cut instance and develop an algorithm that runs in polynomial time and is guaranteed to find the optimum given enough independent SNPs.

Chapter 8 considers a more general version of the problem. Given the genotype matrix I, we wish to classify each position of each sequence as belonging to one of two populations. That is, the problem asks for a classification $\theta : R(I) \times C(I) \to \{0, 0.5, 1\}$ that classifies each position as belonging to one of the two populations or as a mixture (exactly one allele from each population). We develop an algorithm that is based on spectral partitioning to obtain an initial solution, which can be proved to be correct under some assumptions. We then refine the solution by maximizing the likelihood function using an iterative heuristic. We show that in practice the new algorithm provides significantly better accuracy than prior methods.

Part II

Phylogenetic Tree Reconstruction

Chapter 2

Near-Perfect Phylogeny Reconstruction from Haplotypes

Phylogeny construction, or the inference of evolutionary trees from some form of population variation data, is one of the oldest and most intensively studied problems in computational biology [58, 100]. Yet it remains far from solved. The problem has become particularly acute for the special case of intraspecies phylogenetics, or tokogenetics, in which we wish to build evolutionary trees among individuals in a single species. In part, the persistence of the problem reflects its basic computational difficulty. The problem in most reasonable variants is formally NP hard [56] and thus has no known efficient solution. The continuing relevance of phylogeny inference algorithms also stems from the fact that the data sets to be solved have been getting increasingly large in both population sizes and numbers of variations examined. The genomic era has led to the identification of vast numbers of variant sites for human populations [27, 75], as well as various other complex eukaryotic organisms [38, 39, 101]. Large-scale re-sequencing efforts are now under way to use such sites to study population histories with precision never previously possible [22]. Even more vast data sets are available for microbial and viral genomes. As a result, methods that were adequate even a few years ago may no longer be suitable today.

Phylogeny reconstruction is typically performed either under the maximum likelihood model or the maximum parsimony model. In the maximum parsimony (MP) model of this problem one seeks the smallest tree to explain the evolution of a set of observed organisms. Parsimony is a particularly appropriate objective for trees representing short time scales, which makes it a good choice for inferring evolutionary relationships among individuals within a single species or a few closely related species. The intraspecific phylogeny problem has become especially important in studies of human genetics now that large-scale genotyping and the availability of complete human genome sequences have made it possible to identify millions of single nucleotide polymorphisms (SNPs) [102], sites at which a single DNA base takes on two common variants.

Minimizing the length of a phylogeny is the problem of finding the most parsimonious tree, a well known NP-complete problem [53]. Researchers have thus focused on either sophisticated heuristics or solving optimally for special cases (e.g. fixed parameter variants [2, 23, 77]). Previous attempts at such solutions for the general parsimony problem

have only produced theoretical results, yielding algorithms too complicated for practical implementation. A large number of related works have been published, but it is impossible to mention all of them here.

Introductory formal definitions for parsimony based phylogeny reconstruction can be found in the previous chapter. In this work, we focus on the the case when the set of character states is binary. In this setting, the input is often represented as an $n \times m$ binary matrix I. The n rows of the matrix (taxa) can be viewed as points on an m-cube. Therefore, the problem is equivalent to finding the Steiner minimum tree in a hypercube. In the binary state case, a phylogeny is called *perfect* if its length equals m. Gusfield showed that such phylogenies can be reconstructed in linear time [57].

If there exists no perfect phylogeny for input I, then one option is to slightly modify I so that a perfect phylogeny can be constructed for the resulting input. Upper bounds and negative results have been established for such problems. For instance, Day and Sankoff [24], showed that finding the maximum subset of characters containing a perfect phylogeny is NP-complete while Damaschke [23] showed fixed parameter tractability for the same problem. The problem of reconstructing the most parsimonious tree without modifying the input I seems significantly harder.

Fernandez-Baca and Lagergren recently considered the problem of reconstructing optimal near-perfect phylogenies [51], which assume that the size of the optimal phylogeny is at most q larger than that of a perfect phylogeny for the same input size. They developed an algorithm to find the most parsimonious tree in time $nm^{O(q)}2^{O(q^2s^2)}$, where s is the number of states per character, n is the number of taxa and m is the number of characters. This bound may be impractical for sizes of m to be expected from SNP data, even for moderate q. Given the importance of SNP data, it would therefore be valuable to develop methods able to handle large m for the special case of $s = 2$, a problem we call Binary Near Perfect Phylogenetic tree reconstruction (BNPP).

Outline of the Chapter. In this chapter, we present two algorithms to solve the BNPP problem as described below.

Algorithm 1: We first present theoretical and practical results on the optimal solution of the BNPP problem. We completely describe and analyze an intuitive algorithm for the BNPP problem that has running time $O((72\kappa)^q nm + nm^2)$, where κ is the number of characters that violate the *four gamete* condition, a test of perfectness of a data set explained formally later. But always $\kappa \leq m$ and even at equality this result significantly improves the prior running time. Furthermore, the complexity of the previous work would make practical implementation daunting; to our knowledge no implementation of it has ever been attempted. Our results thus describe the first practical phylogenetic tree reconstruction algorithm that finds guaranteed optimal solutions while being computationally feasible for data sets of biologically relevant complexity. A preliminary paper on this algorithm appeared as Sridhar et al. [106].

Algorithm 2: We then present a more involved algorithm that runs in time $O(21^q+8^q nm^2)$. Fernandez-Baca and Lagergren [51] in concluding remarks state that the most important

open problem in the area is to develop a parameterized algorithm or prove $W[t]$ hardness for the near-perfect phylogeny problem. We make progress on this open problem by showing for the first time that BNPP is fixed parameter tractable (FPT). To achieve this, we use a divide and conquer algorithm. Each divide step involves performing a 'guess' (or enumeration) with cost exponential in q. Finding the Steiner minimum tree on a q-cube dominates the run-time when the algorithm bottoms out. The present work substantially improves on the time bounds derived for a preliminary version of this algorithm, which was first presented in Blelloch et al. [9].

We further implement variants of both algorithms and demonstrate them on a selection of real mitochondrial, Y-chromosome and bacterial data sets. The results demonstrate that both algorithms substantially outperform their worst-case time-bounds, yielding optimal trees with high efficiency on real data sets typical of those for which such algorithms would be used in practice. The final version of the contents presented in this chapter is described in Sridhar et. al. [107].

2.1 Preliminaries

In defining formal models for parsimony-based phylogeny construction, we borrow definitions and notations from a couple of previous works [51, 100]. The input to a phylogeny problem is an $n \times m$ binary matrix I where rows $R(I)$ represent *input taxa* and are binary strings. The column numbers $C = \{1, \cdots, m\}$ are referred to as *characters*. In a *phylogenetic tree*, or *phylogeny*, each vertex v corresponds to a taxon (not necessarily in the input) and has an associated label $l(v) \in \{0,1\}^m$.

See definitions 1 and 2 for formal descriptions of a phylogeny. Also, definitions 3, 4, 6, 7 provide formal descriptions of a Steiner vertex, length of a phylogeny, penalty and a q-near-perfect phylogeny respectively. With the above definitions, we are now prepared to define our central computational problem.

The BNPP problem. Given an integer q and an $n \times m$ binary input matrix I, if $\texttt{penalty}(I) \leq q$, then return an optimum phylogeny T, else declare `NIL`. The problem is equivalent to finding the minimum Steiner tree on an m-cube if the optimum tree is at most q larger than the number of dimensions m or declaring `NIL` otherwise. The problem is fundamental and therefore expected to have diverse applications besides phylogenies.

Definition 8 *We define the following* notations*:*

- $r[i] \in \{0,1\}$*: the state in character* i *of taxon* r
- $\mu(e) : E(T) \rightarrow 2^C$*: the set of all characters corresponding to edge* $e = (u, v)$ *with the property for any* $i \in \mu(e)$*,* $u[i] \neq v[i]$*. Note that for the first definition of a phylogeny* $\mu(e) : E(T) \rightarrow C$*.*
- *for a set of taxa* M*, we use* T^*_M *to denote an optimum phylogeny on* M

We say that an edge e *mutates* character i if $i \in \mu(e)$. We will use the following well known definition and lemma on phylogenies.

Definition 9 *Given matrix I, the* set of gametes $G_{i,j}$ *for characters i, j is defined as:* $G_{i,j} = \{(r[i], r[j]) | r \in R(I)\}$. *Two characters i, j* share t *gametes in I i.f.f.* $|G_{i,j}| = t$.

In other words, the set of gametes $G_{i,j}$ is a projection on the i, j dimensions.

Lemma 2.1.1 [57] *An optimum phylogeny for input I is not perfect i.f.f. there exists two characters i, j that share (all) four gametes in I.*

Definition 10 (Conflict Graph [63]) *A* conflict graph G *for matrix I with character set C is defined as follows. Every vertex v of G corresponds to unique character $c(v) \in C$. An edge (u, v) is added to G i.f.f. $c(u), c(v)$ share all four gametes in I. Such a pair of characters are defined to be in* conflict.

Note that if the conflict graph G contains no edges, then a perfect phylogeny can be constructed for I. Gusfield [57] provided an efficient algorithm to reconstruct a perfect phylogeny in such cases.

Simplifications. We assume that the all zeros taxon is present in the input. If not, using our freedom of labeling, we convert the data into an equivalent input containing the all zeros taxon (see section 2.2 of Eskin et. al. [28] for details). We also remove any character that contains only one state. Such characters do not mutate in the whole phylogeny and are therefore useless in any phylogeny reconstruction. The BNPP problem asks for the reconstruction of an unrooted tree. For the sake of analysis, we will however assume that all the phylogenies are rooted at the all zeros taxon.

2.2 Simple Algorithm

This section describes a simple algorithm for the reconstruction of a binary near-perfect phylogenetic tree. Throughout this section, we will use the first definition of a phylogeny (Definition 1).

We begin by performing the following pre-processing step. For every pair of characters c', c'' if $|G_{c',c''}| = 2$, we (arbitrarily) remove character c''. After repeatedly performing the above step, we have the following.

Claim 2.2.1 *For every pair of characters c', c'', $|G_{c',c''}| \geq 3$.*

We will assume that the above claim holds on the input matrix for the rest of the chapter. Note that such characters c', c'' are identical (after possibly relabeling one character) and are usually referred to as non-informative. It is not hard to show that this preprocessing step does not change the correctness or running time of our algorithm.

The following additional definitions are required for the description and analysis of the simple algorithm:

Definition 11 *For any phylogeny T and set of characters $C' \subseteq C$:*

- *a* super node *is a maximal connected subtree T' of T s.t. for all edges $e \in T', \mu(e) \notin C'$*
- *the* skeleton *of T, $s(T, C')$, is the tree that results when all super nodes are contracted to a vertex. The vertex set of $s(T, C')$ is the set of super nodes. For all edges $e \in s(T, C')$, $\mu(e) \in C'$.*

Definition 12 *A* tag $t(u) \in \{0,1\}^m$ *of super node* u *in* $s(T, C')$ *has the property that* $t(u)[c'] = v[c']$ *for all* $c' \in C'$*, vertices* $v \in u$*;* $t[u][i] = 0$ *for all* $i \notin C'$*.*

Throughout this paper, we will assume without loss of generality that we are working with phylogenies and skeletons that are rooted at the all zeros taxon and tag respectively. Furthermore, the skeletons used in this work themselves form a perfect phylogeny in the sense that no character mutates more than once in the skeleton. Note that in such skeletons, tag $t(u)[i] = 1$ if and only if character i mutates exactly once in the path from the root to u. Figure 2.1(a) shows an example of a skeleton of a phylogeny. We will use the term *sub-phylogeny* to refer to a subtree of a phylogeny.

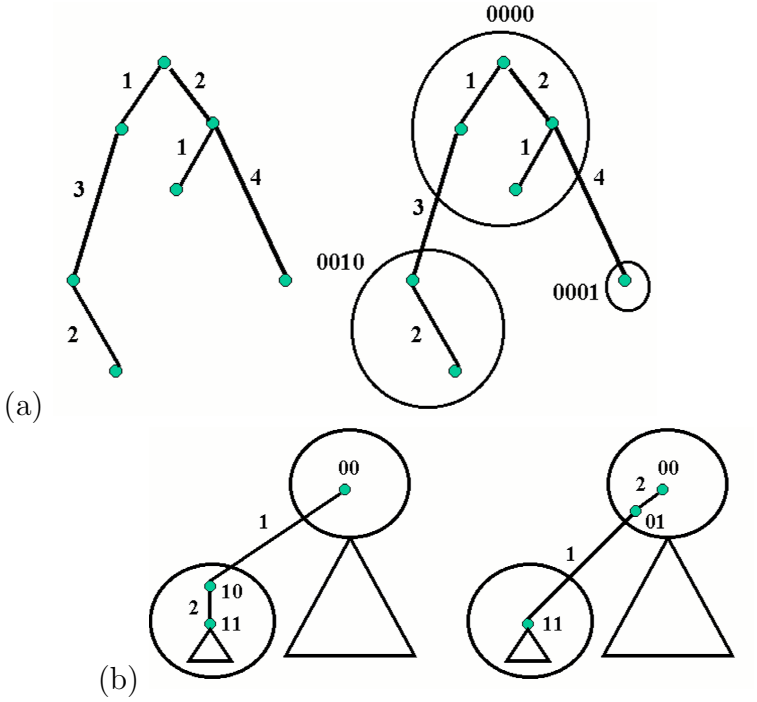


Figure 2.1: (a) Phylogeny T and skeleton $s(T, C')$, $C' = \{3, 4\}$. Edges are labeled with characters that mutate μ and super nodes with tags t. (b) Transform to remove a degree 2 Steiner root from a super node. Note: the size of the phylogeny is unchanged.

Throughout the analysis, we fix an optimal phylogeny T^* and show that our algorithm finds it. We assume that both T_{opt} and its skeleton is rooted at the all zeros label and tag respectively. The high level idea of our algorithm is to first guess the characters that mutate more than once in T_{opt}. The algorithm then finds a perfect phylogeny on the remaining characters. Finally, it adds back the imperfect components by solving a Steiner tree problem. The algorithm is divided into two functions, `buildNPP` and `linkTrees`, whose pseudo-code is provided in Figures 2.2 and 2.3.

Function `buildNPP` starts by determining the set of characters $c(V_{nis})$ that corresponds to the non-isolated vertices of the conflict graph in Step 2. From set $c(V_{nis})$, the algorithm then selects by brute-force the set of characters M that mutate more than once in T_{opt}. Only characters corresponding to non-isolated vertices can mutate more than once in any

function buildNPP (binary matrix I, integer q)

1. let $G(V, E)$ be the conflict graph of I
2. let $V_{nis} \subseteq V$ be the set of non-isolated vertices
3. for all $M \in 2^{c(V_{nis})}$, $|M| \leq q$
   (a) construct rooted perfect phylogeny $\texttt{PP}(V_{PP}, E_{PP})$ on characters $C \setminus M$
   (b) define $\lambda : R \mapsto V_{PP}$ s.t. $\lambda(r) = u$ i.f.f. for all $i \in C \setminus M$, $r[i] = t(u)[i]$
   (c) $T_f :=$ **linkTrees** (PP)
   (d) if $\texttt{penalty}(T_f) \leq q$ then return T_f
4. return NIL

Figure 2.2: Pseudo-code to find the skeleton.

optimal phylogeny (a simple proof follows from Buneman graphs [100]). Since all characters of $C \setminus M$ mutate exactly once, the algorithm constructs a perfect phylogeny on this character set using Gusfield's linear time algorithm [57]. The perfect phylogeny is unique because of Claim 2.2.1. Note that PP is the skeleton $s(T_{opt}, C \setminus M)$. Since the tags of the skeleton are unique, the algorithm can now determine the super node where every taxon resides as defined by function λ in Step 3b. This rooted skeleton PP is then passed into function `linkTrees` to complete the phylogeny.

Function `linkTrees` takes a rooted skeleton Sk (sub-skeleton of PP) as argument and returns a tuple (r, c). The goal of function `linkTrees` is to convert skeleton Sk into a phylogeny for the taxa that reside in Sk by adding edges that mutate M. Notice that using function λ, we know the set of taxa that reside in skeleton Sk. The phylogeny for Sk is built bottom-up by first solving the phylogenies on the sub-skeleton rooted at children super nodes of Sk. Tuple (r, c) returned by function call to $\texttt{linkTrees}(Sk)$ represents the cost c of the optimal phylogeny when the label of the root vertex in the root super node of Sk is r. Let $S = \texttt{root}(Sk)$ represent the root super node of skeleton Sk. R_S is the set of input taxa that map to super node S under function λ. Let its children super nodes be $S_1, S_2, \ldots$. Assume that recursive calls to $\texttt{linkTrees}(S_i)$ return (r_i, c_i). Notice that the parents of the set of roots r_i all reside in super node S. The parents of r_i are denoted by p_i and are identical to r_i except in the character that mutates in the edge connecting S_i to S. Set τ is the union of p_i and R_S, and forms the set of vertices inferred to be in S. Set D is the set of characters on which the labels of τ differ i.e. for all $i \in D, \exists r_1, r_2 \in \tau, r_1[i] \neq r_2[i]$. In Step 9, we guess the root r_S of super node S. This guess is 'correct' if it is identical to the label of the root vertex of S in T_{opt}. Notice that we are only guessing $|D|$ bits of r_S. Corollary 2.2.3 of Lemma 2.2.2 along with optimality requires that the label of the root vertex of T_{opt} is identical to τ in all the characters $C \setminus D$:

Lemma 2.2.2 *There exists an optimal phylogeny T_{opt} that does not contain any degree 2 Steiner roots in any super node.*

function linkTrees (skeleton $Sk(V_s, E_s)$ **)**

1. let $S :=$ `root`(Sk)
2. let $R_S := \{s \in R | \lambda(s) = S\}$
3. for all children S_i of S
   (a) let Sk_i be subtree of Sk rooted at S_i
   (b) $(r_i, c_i) :=$ **linkTrees**(Sk_i)
4. let `cost` $:= \sum_i c_i$
5. for all i, let $l_i := \mu(S, c_i)$
6. for all i, define $p_i \in \{0,1\}^m$ s.t. $p_i[l_i] \neq r_i[l_i]$ and for all $j \neq l_i$, $p_i[j] = r_i[j]$
7. let $\tau := R_S \cup (\cup_i \{p_i\})$
8. let $D \subseteq C$ be the set of characters where taxa in τ differ
9. guess root taxon of S, $r_S \in \{0,1\}^m$ s.t. $\forall i \in C \setminus D, \forall u \in \tau,\ r_S[i] = u[i]$
10. let c_S be the size of the optimal Steiner tree of $\tau \cup \{r_S\}$
11. return $(r_S,$ `cost` $+ c_S)$

Figure 2.3: Pseudo-code to construct and link imperfect phylogenies

Proof: Figure 2.1(b) shows how to transform a phylogeny that violates the property into one that doesn't. Root `10` is degree 2 Steiner and is moved into parent supernode as `01`. Since `10` was Steiner, the transformed tree contains all input.

Corollary 2.2.3 *In T_{opt}, the LCA of the set τ is the root of super node S.*

In step 10, the algorithm finds the cost of the optimum Steiner tree for the terminal set of taxa $\tau \cup \{r_S\}$. We use Dreyfus-Wagner recursion [91] to compute this minimum Steiner tree. The function now returns r_S along with the cost of the phylogeny rooted in S which is obtained by adding the cost of the optimum Steiner tree in S to the cost of the phylogenies rooted at c_i. The following Lemma bounds the running time of our algorithm and completes the analysis:

Lemma 2.2.4 *The algorithm described above runs in time $O((18\kappa)^q nm + nm^2)$ and solves the BNPP problem with probability at least 2^{-2q}. The algorithm can be easily derandomized to run in time $O((72\kappa)^q nm + nm^2)$.*

Proof: The probability of a correct guess at Step 9 in function `linkTrees` is exactly $2^{-|D|}$. Notice that the Steiner tree in super node S has at least $|D|$ edges. Since `penalty`$(T_{opt}) \leq q$, we know that there are at most $2q$ edges that can be added in all of the recursive calls to `linkTrees`. Therefore, the probability that all guesses at Step 9 are correct is at least 2^{-2q}. The time to construct the optimum Steiner tree in step 10 is $O(3^{|\tau|}2^{|D|})$. Assuming that all guesses are correct, the total time spent in Step 10 over all recursive calls is $O(3^{2q}2^q)$. Therefore, the overall running time of the randomized algorithm

is $O((18\kappa)^q nm + nm^2)$. To implement the randomized algorithm, since we do not know if the guesses are correct, we can simply run the algorithm for the above time, and if we do not have a solution, then we restart. *Although presented as a randomized algorithm for ease of exposition, it is not hard to see that the algorithm can be derandomized by exploring all possible roots at Step 9.* The derandomized algorithm has total running time $O((72\kappa)^q nm + nm^2)$.

2.3 Fixed Parameter Tractable Algorithm

This section deals with the complete description and analysis of our fixed parameter tractable algorithm for the BNPP problem. Throughout this section we will use the second definition of a phylogeny (Definition 2). For ease of exposition, we first describe a randomized algorithm for the BNPP problem that runs in time $O(18^q + qnm^2)$ and returns an optimum phylogeny with probability at least 8^{-q}. We later show how to derandomize it. In sub-section 2.3.1, we first provide the complete pseudo-code and describe it. In sub-section 2.3.2, we prove the correctness of the algorithm. In sub-section 2.3.3, we upper-bound the running time for the randomized and derandomized algorithms and the probability that the randomized algorithm returns an optimum phylogeny. The above work follows that presented in a preliminary paper on the topic [9]. Finally, in sub-section 2.3.4, we show how to tighten the above bounds on the derandomized algorithm to achieve our final result of $O(21^q + 8^q nm^2)$ run time.

2.3.1 Description

We begin with a high-level description of our randomized algorithm. The algorithm iteratively finds a set of edges E that decomposes an optimum phylogeny T_I^* into at most q components. An optimum phylogeny for each component is then constructed using a simple method and returned along with edges E as an optimum phylogeny for I.

We can alternatively think of the algorithm as a recursive, divide and conquer procedure. Each recursive call to the algorithm attempts to reconstruct an optimum phylogeny for an input matrix M. The algorithm identifies a character c s.t. there exists an optimum phylogeny T_M^* in which c mutates exactly once. Therefore, there is exactly one edge $e \in T_M^*$ for which $c \in \mu(e)$. The algorithm, then guesses the vertices that are adjacent to e as r, p. The matrix M can now be partitioned into matrices $M0$ and $M1$ based on the state at character c. Clearly all the taxa in $M1$ reside on one side of e and all the taxa in $M0$ reside on the other side. The algorithm adds r to $M1$, p to $M0$ and recursively computes the optimum phylogeny for $M0$ and $M1$. An optimum phylogeny for M can be reconstructed as the union of *any* optimum phylogeny for $M0$ and $M1$ along with the edge (r, p). We require at most q recursive calls. When the recursion bottoms out, we use a simple method to solve for the optimum phylogeny.

We describe and analyze the iterative method which flattens the above recursion to simplify the analysis. For the sake of simplicity, we also define the following notations:

buildNPP(input matrix I)

1. let $L := \{I\}, E := \emptyset$
2. while $|\cup_{M_i \in L} N(M_i)| > q$
   (a) guess vertex v from $\cup_{M_i \in L} N(M_i)$, let $v \in N(M_j)$
   (b) let $M0 := M_j(c(v), 0)$ and $M1 := M_j(c(v), 1)$
   (c) guess taxa r and p
   (d) add r to $M1$, p to $M0$ and (r, p) to E
   (e) remove M_j from L, add $M0$ and $M1$ to L
3. for each $M_i \in L$ compute an optimum phylogeny T_i
4. return $E \cup (\cup_i T_i)$

Figure 2.4: Pseudo-code to solve the BNPP problem. For all $M_i \in L$, $N(M_i)$ is the set of non-isolated vertices in the conflict graph of M_i. Guess at Step 2a is correct i.f.f. there exists $T^*_{M_j}$ where $c(v)$ mutates exactly once. Guess at Step 2c is correct i.f.f. there exists $T^*_{M_j}$ where $c(v)$ mutates exactly once and edge $(r, p) \in T^*_{M_j}$ with $r[c(v)] = 1, p[c(v)] = 0$. Implementation details for Steps 2a, 2c and 3 are provided in Section 2.3.3.

- For the set of taxa M, $M(i, s)$ refers to the subset of taxa that contains state s at character i.
- For a phylogeny T and character i that mutates exactly once in T, $T(i, s)$ refers to the maximal subtree of T that contains state s on character i.

The pseudo-code for the above described algorithm is provided in Figure 2.4. The algorithm performs 'guesses' at Steps 2a and 2c. If all the guesses performed by the algorithm are 'correct' then it returns an optimum phylogeny. The guess at Step 2a is correct if and only if there exists $T^*_{M_j}$ where $c(v)$ mutates exactly once. The guess at Step 2c is correct if and only if there exists $T^*_{M_j}$ where $c(v)$ mutates exactly once and edge $(r, p) \in T^*_{M_j}$ with $r[c(v)] = 1, p[c(v)] = 0$. Implementation details for Steps 2a, 2c and 3 are provided in Section 2.3.3. An example illustrating the reconstruction is provided in Figure 2.5.

2.3.2 Correctness

We will now prove the correctness of the pseudo-code under the assumption that all the guesses performed by our algorithm are correct. Specifically, we will show that if $\texttt{penalty}(I) \leq q$ then function `buildNPP` returns an optimum phylogeny. The following lemma proves the correctness of our algorithm.

Lemma 2.3.1 *At any point in execution of the algorithm, an optimum phylogeny for I can be constructed as $E \cup (\cup_i T_i)$, where T_i is* any *optimum phylogeny for $M_i \in L$.*

Proof: We prove the lemma using induction. The lemma is clearly true at the beginning of the routine when $L = \{I\}, E = \emptyset$. As inductive hypothesis, assume that

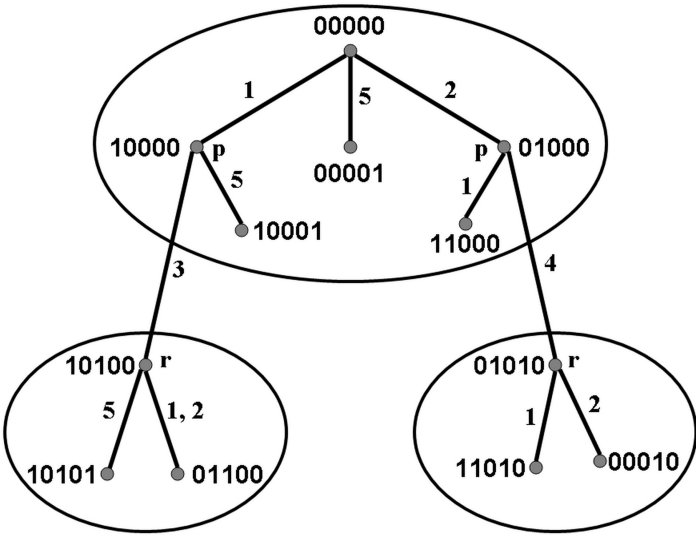


Figure 2.5: Example illustrating the reconstruction. Underlying phylogeny is T_I^*; taxa r and p (both could be Steiner) are guessed to create $E = \{(10000, 10100), (01000, 01010)\}$; E induces three components in T_I^*. When all taxa in T_I^* are considered, character 3 conflicts with $1, 2$ and 5 and character 4 conflicts with 1 and 2; two components are perfect (penalty 0) and one has penalty 2; $\texttt{penalty}(I) =_{def} \texttt{penalty}(T_I^*) = 7$.

the above property is true right before an execution of Step 2e. Consider any optimum phylogeny $T_{M_j}^*$ where $c(v)$ mutates exactly once and on the edge (r, p). Phylogeny $T_{M_j}^*$ can be decomposed into $T_{M_j}^*(c(v), 0) \cup T_{M_j}^*(c(v), 1) \cup (r, p)$ with length $l = \texttt{length}(T_{M_j}^*(c(v), 0)) + \texttt{length}(T_{M_j}^*(c(v), 1)) + d(r, p)$. Again, since $c(v)$ mutates exactly once in $T_{M_j}^*$, all the taxa in $M0$ and $M1$ are also in $T_{M_j}^*(c(v), 0)$ and $T_{M_j}^*(c(v), 1)$ respectively. Let T', T'' be *arbitrary* optimum phylogenies for $M0$ and $M1$ respectively. Since $p \in M0$ and $r \in M1$ we know that $T' \cup T'' \cup (r, p)$ is a phylogeny for M_j with cost $\texttt{length}(T') + \texttt{length}(T'') + d(r, p) \leq l$. By the inductive hypothesis, we know that an optimum phylogeny for I can be constructed using any optimum phylogeny for M_j. We have now shown that using any optimum phylogeny for $M0$ and $M1$ and adding edge (r, p) we can construct an optimum phylogeny for M_j. Therefore the proof follows by induction.

2.3.3 Initial Bounds

In this sub-section we bound the probability of correct guesses, analyze the running time and show how to derandomize the algorithm. We perform two guesses at Steps 2a and 2c. Lemmas 2.3.2 and 2.3.6 bound the probability that all the guesses performed at these steps are correct throughout the execution of the algorithm.

Lemma 2.3.2 *The probability that all guesses performed at Step 2a are correct is at least* 4^{-q}.

Proof: Implementation: The guess at Step 2a is implemented by selecting v uniformly at random from $\cup_i N(M_i)$.

To prove the lemma, we first show that the number of iterations of the `while` loop (step 2) is at most q. Consider any one iteration of the while loop. Since v is a non-isolated vertex of the conflict graph, $c(v)$ shares all four gametes with some other character c' in some M_j. Therefore, in every optimum phylogeny $T^*_{M_j}$ that mutates $c(v)$ exactly once, there exists a path P starting with edge e_1 and ending with e_3 both mutating c', and containing edge e_2 mutating $c(v)$. Furthermore, the path P contains no other mutations of $c(v)$ or c'. At the end of the current iteration, M_j is replaced with $M0$ and $M1$. Both subtrees of $T^*_{M_j}$ containing $M0$ and $M1$ contain (at least) one mutation of c' each. Therefore, $\texttt{penalty}(M0) + \texttt{penalty}(M1) < \texttt{penalty}(M_j)$. Since $\texttt{penalty}(I) \leq q$, there can be at most q iterations of the while loop.

We now bound the probability. Intuitively, if $|\cup_i N(M_i)|$ is very large, then the probability of a correct guess is large, since at most q out of $|\cup_i N(M_i)|$ characters can mutate multiple times in $T^*_{M_j}$. On the other hand if $|\cup_i N(M_i)| = q$ then we terminate the loop. Formally, at each iteration $|\cup_i N(M_i)|$ reduces by at least 1 (guessed vertex v is no longer in $\cup_i N(M_i)$). Therefore, in the worst case (to minimize the probability of correct guesses), we can have q iterations of the loop, with $q+1$ non-isolated vertices in the last iteration and $2q$ in the first iteration. The probability in such a case that all guesses are correct is at least

$$(\frac{q}{2q}) \times (\frac{q-1}{2q-1}) \times \ldots \times (\frac{1}{q+1}) = \frac{1}{\binom{2q}{q}} \geq 2^{-2q}$$

Application of Buneman Graphs

We now show that r, p can be found efficiently. To prove this we need some tools from the theory of Buneman graphs [100]. Let M be a set of taxa defined by character set C of size m. A Buneman graph F for M is a vertex induced subgraph of the m-cube. Graph F contains vertices v if and only if for every pair of characters $i, j \in C$, $(v[i], v[j]) \in G_{i,j}$. Recall that $G_{i,j}$ is the set of gametes (or projection of M on dimensions i, j). Each edge of the Buneman graph is labeled with the character at which the adjacent vertices differ.

We will use the Buneman graph to show how to incrementally extend a set of taxa M by adding characters that share exactly two gametes with some existing character. As before, we can assume without loss of generality that the all zeros taxon is present in M. Therefore if a pair of characters share exactly two gametes then they are identical. Assume that we want to add character i to M and $i' \in M$ is identical to i. We extend M to M' by first adding the states on character i' for all taxa. For the rest of the discussion let $G_{i,j}$ be the set of gametes shared between characters i, j in matrix M'. We extend M' to M'' by adding a taxon t s.t. $t[i] = 0, t[i'] = 1$ and for all other characters j, if $(0,1) \notin G_{j,i}$ then $t[j] = 1$ else $t[j] = 0$. Since we introduced a new gamete on i, i', no pair of characters share exactly two gametes in M''. Therefore a Buneman graph G'' for M'' can be constructed as before. A Buneman graph is a median graph [100] and clearly a subgraph of the $m+1$-cube, where $m+1$ is the number of characters in M''. Every taxon in M' is present in G''

by construction. Using the two properties, we have the following lemma.

Lemma 2.3.3 *Every optimum phylogeny for the taxa in M' defined over the $m+1$ characters is contained in G''(See Section 5.5, [100] for more details).*

We now show the following important property on the extended matrix M''.

Lemma 2.3.4 *If a pair of characters c, c' conflict in M'' then they conflict in M'.*

Proof: For the sake of contradiction, assume not. Clearly i, i' share exactly three gametes in M''. Now consider any character j and assume that j, i shared exactly three gametes in M'. For the newly introduced taxon t, $t[i] = 0$. If $t[j] = 1$, then j, i cannot share $(0,1)$ gamete in M'' and therefore they do not conflict. If $t[j] = 0$, then the newly introduced taxon creates the $(0,0)$ gamete which should be present in all pairs of characters. Now consider the pair of characters (j, i'). If $t[j] = 1$, then in any taxon t' of M', if $t'[j] = 1$ then $t'[i] = 1$ and therefore $t[i] = 1$ (since i, i' are identical on all taxa except t) and therefore $(1,1)$ cannot be a newly introduced gamete. If $t[j] = 0$, then there exists some taxon t' for which $t'[j] = 0$ and $t'[i] = 1$ and therefore $t'[i'] = 1$ and again $(0,1)$ cannot be a newly introduced gamete. Finally consider any pair of characters j, j'. If taxon t introduces gamete $(0,1)$, then there exists some taxon t' with $t'[j] = 0$ and $t'[i] = 1$ If $t'[j'] = 1$, then $(0,1)$ cannot be a new gamete. If $t'[j'] = 0$, then $t[j'] = 0$ and not 1. The case when $(1,0)$ is introduced by t is symmetric. Finally if t introduces $(1,1)$ then consider any taxon t' with $t'[i] = 1$. It has to be the case that $t'[j] = t'[j'] = 1$, and therefore $(1,1)$ cannot be a newly introduced gamete.

We now have the following lemma.

Lemma 2.3.5 *In every optimum phylogeny T^*_M, the conflict graph on the set of taxa in T^*_M (Steiner vertices included) is the same as the conflict graph on M.*

Proof: We say that a subgraph F' of F is the same as an edge labeled tree T if F' is a tree and T can be obtained from F' by suppressing degree-two vertices. A phylogeny T is contained in a graph F if there exists an edge-labeled subgraph F' that is the same as the edge labeled (by function μ) phylogeny T. We know from Lemma 2.3.3 that all optimum phylogenies T^*_M for M is contained in the (extended) Buneman graph of M. Lemma 2.3.4 shows that the conflict graph on M'' (and therefore on the extended Buneman graph of M'') is the same as the conflict graph of M.

Lemma 2.3.6 *The probability that all guesses performed at Step 2c are correct is at least 2^{-q}.*

Proof: Implementation: We first show how to perform the guess efficiently. For every character i, we perform the following steps in order.

1. if all taxa in $M0$ contain the same state s in i, then fix $r[i] = s$
2. if all taxa in $M1$ contain the same state s in i, then fix $r[i] = s$
3. if $r[i]$ is unfixed then guess $r[i]$ uniformly at random from $\{0,1\}$

Assuming that the guess at Step 2a (Figure 2.4) is correct, we know that there exists an optimum phylogeny $T^*_{M_j}$ on M_j where $c(v)$ mutates exactly once. Let $e \in T^*_{M_j}$ s.t. $c(v) \in \mu(e)$. Let r' be an end point of e s.t. $r'[c(v)] = 1$ and p' be the other end point. If the first two conditions hold with the same state s, then character i does not

mutate in M_j. In such a case, we know that $r'[i] = s$, since $T^*_{M_j}$ is optimal and the above method ensures that $r[i] = s$. Notice that if both conditions are satisfied simultaneously with different values of s then i and $c(v)$ share exactly two gametes in M_j and therefore $i, c(v) \in \mu(e)$. Hence, $r'[i] = r[i]$. We now consider the remaining cases when exactly one of the above conditions hold. We show that if $r[i]$ is fixed to s then $r'[i] = s$. Note that in such a case at least one of $M0, M1$ contain both the states on i and $i, c(v)$ share at least 3 gametes in M_j. The proof can be split into two symmetric cases based on whether r is fixed on condition 1 or 2. One case is presented below:

Taxon $r[i]$ is fixed based on condition 1: In this case, all the taxa in $M0$ contain the same state s on i. Therefore, the taxa in $M1$ should contain both states on i. Hence i mutates in $T^*_{M_j}(c(v), 1)$. For the sake of contradiction, assume that $r'[i] \neq s$. If $i \notin \mu(e)$ then $p'[i] \neq s$. However, all the taxa in $M0$ contain state s. This implies that i mutates in $T^*_{M_j}(c(v), 0)$ as well. Therefore i and $c(v)$ share all four gametes on $T^*_{M_j}$. However i and $c(v)$ share at most 3 gametes in M_j - one in $M0$ and at most two in $M1$. This leads to a contradiction to Lemma 2.3.5. Once r is guessed correctly, p can be computed since it is is identical to r in all characters except $c(v)$ and those that share two gametes with $c(v)$ in M_j. We make a note here that we are assuming that e does not mutate any character that does not share two gametes with $c(v)$ in M_j. This creates a small problem that although the length of the tree constructed is optimal, r and p could be degree-two Steiner vertices. If after constructing the optimum phylogenies for $M0$ and $M1$, we realize that this is the case, then we simply add the mutation adjacent to r and p to the edge (r, p) and return the resulting phylogeny where both r and p are not degree-two Steiner vertices.

The above implementation therefore requires only guessing states corresponding to the remaining unfixed characters of r. If a character i violates the first two conditions, then i mutates once in $T^*_{M_j}(i, 0)$ and once in $T^*_{M_j}(i, 1)$. If $r[i]$ has not been fixed, then we can associate a pair of mutations of the same character i with it. At the end of the current iteration M_j is replaced with $M0$ and $M1$ and each contains exactly one of the two associated mutations. Therefore if q' characters are unfixed then $\texttt{penalty}(M0) + \texttt{penalty}(M1) \leq \texttt{penalty}(M_j) - q'$. Since $\texttt{penalty}(I) \leq q$, throughout the execution of the algorithm there are q unfixed states. Therefore the probability of all the guesses being correct is 2^{-q}.

This completes our analysis for upper bounding the probability that the algorithm returns an optimum phylogeny. We now analyze the running time. We use the following lemma to show that we can efficiently construct optimum phylogenies at Step 3 in the pseudo-code:

Lemma 2.3.7 *For a set of taxa M, if the number of non-isolated vertices of the associated conflict graph is t, then an optimum phylogeny T^*_M can be constructed in time $O(3^s 6^t + nm^2)$, where $s = \texttt{penalty}(M)$.*

Proof: We use the approach described by Gusfield and Bansal (see Section 7 of [62]) that relies on the Decomposition Optimality Theorem for recurrent mutations. We first construct the conflict graph and identify the non-trivial connected components of it in

time $O(nm^2)$. Let κ_i be the set of characters associated with component i. We compute the Steiner minimum tree T_i for character set κ_i. The remaining conflict-free characters in $C \setminus \cup_i \kappa_i$ can be added by contracting each T_i to vertices and solving the perfect phylogeny problem using Gusfield's linear time algorithm [57].

Since $\texttt{penalty}(M) = s$, there are at most $s + t + 1$ distinct bit strings defined over character set $\cup_i \kappa_i$. The Steiner space is bounded by 2^t, since $|\cup_i \kappa_i| = t$. Using the Dreyfus-Wagner recursion [91] the total run-time for solving all Steiner tree instances is $O(3^{s+t}2^t)$.

Lemma 2.3.8 *The algorithm described solves the BNPP problem in time $O(18^q + qnm^2)$ with probability at least 8^{-q}.*

Proof: For a set of taxa $M_i \in L$ (Step 3, Figure 2.4), using Lemma 2.3.7 an optimum phylogeny can be constructed in time $O(3^{s_i}6^{t_i} + nm^2)$ where $s_i = \texttt{penalty}(M_i)$ and t_i is the number of non-isolated vertices in the conflict graph of M_i. We know that $\sum_i s_i \leq q$ (since $\texttt{penalty}(I) \leq q$) and $\sum_i t_i \leq q$ (stopping condition of the `while` loop). Therefore, the total time to reconstruct optimum phylogenies for all $M_i \in L$ is bounded by $O(18^q + nm^2)$. The running time for the while loop is bounded by $O(qnm^2)$. Therefore the total running time of the algorithm is $O(18^q + qnm^2)$. Combining Lemmas 2.3.2 and 2.3.6, the total probability that all guesses performed by the algorithm is correct is at least 8^{-q}.

Lemma 2.3.9 *The algorithm described above can be derandomized to run in time $O(72^q + 8^q nm^2)$.*

Proof: It is easy to see that Step 2c can be derandomized by exploring all possible states for the unfixed characters. Since there are at most q unfixed characters throughout the execution, there are 2^q possibilities for the states.

However, Step 2a cannot be derandomized naively. We use the technique of bounded search tree [26] to derandomize it efficiently. We select an arbitrary vertex v from $\cup_i N(M_i)$. We explore both the possibilities on whether v mutates once or multiple times. We can associate a search (binary) tree with the execution of the algorithm, where each node of the tree represents a selection v from $\cup_i N(M_i)$. One child edge represents the execution of the algorithm assuming v mutates once and the other assuming v mutates multiple times. In the execution where v mutates multiple times, we select a different vertex from $\cup_i N(M_i)$ and again explore both paths. The height of this search tree can be bounded by $2q$ because at most q characters can mutate multiple times. The path of height $2q$ in the search tree is an interleaving of q characters that mutate once and q characters that mutate multiple times. Therefore, the size of the search tree is bounded by 4^q.

Combining the two results, the algorithm can be derandomized by solving at most 8^q different instances of Step 3 while traversing the while loop 8^q times for a total running time of $O(144^q + 8^q nm^2)$. This is, however, an over-estimate. Consider any iteration of the while loop when M_j is replaced with $M0$ and $M1$. If a state in character c is unfixed and therefore guessed, we know that there are two associated mutations of character c in both $M0$ and $M1$. Therefore at iteration i, if q'_i states are unfixed, then $\texttt{penalty}(M0) + \texttt{penalty}(M1) \leq \texttt{penalty}(M_j) - q'_i$. At the end of the iteration we can reduce the value of q used in Step 2 by q'_i, since the penalty has reduced by q'_i. Intuitively this implies that if we perform a total of q' guesses (or enumerations) at Step 2c, then at Step 3 we only need to solve Steiner trees

on $q - q'$ characters. The additional cost $2^{q'}$ that we incur results in reducing the running time of Step 3 to $O(18^{q-q'} + qnm^2)$. Therefore the total running time is $O(72^q + 8^q nm^2)$.

2.3.4 Improving the Run Time Bounds

In Lemma 2.3.9, we showed that the guesses performed at Step 2c of the pseudo-code in Figure 2.4 do not affect the overall running time. We can also establish a trade-off along similar lines for Step 2a that can reduce the theoretical run-time bounds. We now analyze the details of such a trade-off in the following lemma:

Lemma 2.3.10 *The algorithm presented above runs in time* $O(21^q + 8^q nm^2)$.

Proof: For the sake of this analysis, we can declare each character to be in either a 'marked' state or an 'unmarked' state. At the beginning of the algorithm, all the characters are 'unmarked'. As the algorithm proceeds, we will mark characters to indicate that the algorithm has identified them as mutating more than once in T^*

We will then examine two parameters, ρ and γ, which specify the progress made by the derandomized algorithm in either identifying multiply mutating characters or reducing the problem to sub-problems of lower total penalty. Consider the set of characters S such that for all $c \in S$, character c is unmarked and there exists matrix M_i such that c mutates more than once in $T^*_{M_i}$. We define parameter ρ to be $|S|$. Parameter ρ, intuitively, refers to the number of characters mutating more than once (within trees $T^*_{M_i}$) that have not been identified yet. Parameter γ denotes the sum of the penalties of the remaining matrices M_i, $\gamma = \sum_i \text{penalty}(M_i)$.

Consider Step 2a of Figure 2.4, when the algorithm selects character $c(v)$. After selecting $c(v)$, the algorithm proceeds to explore both cases when $c(v)$ either mutates once or multiple times in $T^*_{M_j}$. In the first case, $\texttt{penalty}(T^*_{M_j})$ decreases by at least 1. Therefore, γ decreases by at least 1. In the second case, the algorithm has successfully identified a multiple mutant. We now proceed to mark character $c(v)$, which reduces ρ by 1 and leaves γ unchanged.

If the main loop at Step 3 terminates, then the algorithm finds optimal Steiner trees using the Dreyfus-Wagner recursion and the run-time is bounded by 18^γ using Lemma 2.3.7 as before. Therefore, the running time of this portion of our algorithm can be expressed as:

$$T(\gamma, \rho) \leq \max\{18^\gamma, T(\gamma - 1, \rho) + T(\gamma, \rho - 1) + 1\}$$

Function $T(\gamma, \rho)$ can be upper-bounded by $18^{\gamma+1}(19/17)^{\rho+1}$. We can verify this by induction. The right-side of the above equation is:

$$\begin{aligned}
&\max\{18^\gamma, 18^\gamma(19/17)^{\rho+1} + 18^{\gamma+1}(19/17)^\rho + 1\} \\
&= \max\{18^\gamma, 18^\gamma(19/17)^\rho(19/17 + 18) + 1\} \\
&\leq \max\{18^\gamma, 18^\gamma(19/17)^\rho(19/17 + 19)\} \\
&= 18^{\gamma+1}(19/17)^{\rho+1}
\end{aligned}$$

Since we know that $\gamma \leq q$ and $\rho \leq q$, we can bound $T(q,q) = O(20.12^q)$. Therefore, we can improve the run-time bound for the complete algorithm to $O(20.12^q + 8^q nm^2)$.

We note that further improvements may be achievable in practice for moderate q by pre-processing possible Steiner tree instances. If all Steiner tree problem instances on the q-cube are solved in a pre-processing step, then our running time would depend only on the number of iterations of the while loop, which is $O(8^q nm^2)$. Such pre-processing would be impossible to perform with previous methods. Alternate algorithms for solving Steiner trees may be faster in practice as well.

2.4 Experiments

We tested both algorithms using a selection of non-recombining DNA sequences. These include mitochondrial DNA samples from two human populations [113] and a chimpanzee population [111], Y chromosome samples from human [75] and chimpanzee populations [111], and a bacterial DNA sample [80]. Such non-recombining data sources provide a good test of the algorithms' ability to perform inferences in situations where recurrent mutation is the probable source of any deviation from the perfect phylogeny assumption.

We implemented variants of both algorithms. The simple algorithm was derandomized and used along with a standard implementation of the Dreyfus-Wagner routine. For the FPT algorithm, we implemented the randomized variant described above using an optimized Dreyfus-Wagner routine. The randomized algorithm takes two parameters, q and p, where q is the imperfectness and p is the maximum probability that the algorithm has failed to find an optimal solution of imperfectness q. On each random trial, the algorithm tallies the probability of failure of each random guess, allowing it to calculate an upper bound on the probability that that trial failed to find an optimal solution. It repeats random trials until the accumulated failure probability across all trials is below the threshold p. An error threshold of 1% was used for the present study.

The results are summarized in Table 2.1. Successive columns of the table list the source of the data, the input size, the optimal penalty q, the parsimony score of the resulting tree, the run times of both of our algorithms in seconds, and the number of trials the randomized FPT algorithm needed to reach a 1% error bound. All run times reported are based on execution on a 2.4 GHz Intel P4 computer with 1 Gb of RAM. One data point, the human mtDNA sample from the Buddhist population, was omitted from the results of the simple algorithm because it failed to terminate after 20 minutes of execution. All other instances were solved optimally by the simple algorithm and all were solved by the randomized FPT algorithm. The randomized variant of the FPT algorithm in all but one case significantly outperformed the derandomized simple algorithm in run time. This result that may reflect the superior asymptotic performance of the FPT algorithm in general, the performance advantage of the randomized versus the deterministic variants, and the advantage of a more highly optimized Dreyfus-Wagner subroutine. The randomized algorithm also generally needed far fewer trials to reach a high probability of success than would be expected from the theoretical error bounds, suggesting that those bounds are quite pessimistic for realistic data sets. Both implementations, however, appear efficient for biologically realistic data

sets with moderate imperfection.

We can compare the quality of our solutions to those produced on the same data sets by other methods. Our methods produced trees of identical parsimony score to those derived by the `pars` program from the `PHYLIP` package [50]. However while we can guarantee optimality of the returned results, `pars` does not provide any guarantee on the quality of the tree. (Note that our preliminary paper [106] incorrectly stated that `pars` produced an inferior tree on the chimpanzee mtDNA data set.) Our methods also yielded identical output for the chimpanzee Y chromosome data to a branch-and-bound method used in the paper in which that data was published [111].

Description	Rows × Cols	q	Pars Score	Run time		trials
				Simple (secs)	FPT (secs)	
mtDNA, genus Pan [111]	24×1041	2	63	0.59	0.14	25
chr Y, genus Pan [111]	15×98	1	99	0.33	0.02	12
Bacterial DNA sequence [80]	17×1510	7	96	0.47	4.61	262
HapMap chr Y, 4 ethnic groups [75]	150×49	1	16	0.3	0.02	16
mtDNA, Humans (Muslims) [113]	13×48	3	30	0.61	0.28	117
mtDNA, Humans (Buddhists) [113]	26×48	7	43	—-	18.44	1026

Table 2.1: Empirical Results on a collection of Real SNP Variation Data Sets

2.5 Conclusions

We have presented two new algorithms for inferring optimal near-perfect binary phylogenies. The algorithms substantially improve on the running times of any previous methods for the BNPP problem. This problem is of considerable practical interest for phylogeny reconstruction from SNP data. Furthermore, our algorithms are easily implemented, unlike previous theoretical algorithms for this problem. The algorithms can also provide guaranteed optimal solutions in their derandomized variants, unlike popular fast heuristics for phylogeny construction. Experiments on several non-recombining variation data sets have further shown the methods to be generally extremely fast on real-world data sets typical of those for which one would apply the BNPP problem in practice. Our algorithms perform in practice substantially better than would be expected from their worst case run time bounds, with both proving practical for at least some problems with q as high as seven. The FPT algorithm in its randomized variant shows generally superior practical performance to the simple algorithm. In addition, the randomized algorithm appears to find optimal solutions for these data sets in far fewer trials than would be predicted from the worst-case theoretical bounds. Even the deterministic variant of the simple algorithm, though, finds optimal solutions in under one second for all but one example. The algorithms presented here thus represent the first practical methods for provably optimal near-perfect phylogeny inference from biallelic variation data.

Chapter 3

Maximum Parsimony Phylogeny Reconstruction from Haplotypes

In this chapter, we continue to focus on the inference of intraspecies phylogenies on binary genetic variation data, which is of particular practical importance because of the large amount of binary SNP data now available. The binary intraspecies phylogeny problem has traditionally been modeled by the minimum Steiner tree problem on binary sequences, a classic NP hard problem [56]. Some special cases of the problem are efficiently solvable, most notably the case of *perfect phylogenies*, in which each variant site mutates only once within the optimal tree [2, 57, 77]. However, real data will not, in general, conform to the perfect phylogeny assumption. The standard in practice is the use of sophisticated heuristics that will always produce a tree but cannot guarantee optimality (e.g. [5, 50, 97]). As outlined in the previous chapter, some theoretical advances have recently been made in the efficient solution of *near-perfect phylogenies*, those that deviate only by a fixed amount from the assumption of perfection [9, 51, 105, 106]. These methods can provide provably efficient solutions in many instances, but still struggle with some moderate-size data sets in practice. As a result, some recent attention has turned to integer linear programming (ILP) methods [60]. ILPs provide provably optimal solutions and while they do not provide guaranteed run-time bounds, they may have practical run times far better than those of the provably efficient methods.

In this chapter, we develop two ILP formulations to solve the most parsimonious phylogenetic tree problem on binary sequences. These methods find provably optimal trees from real binary sequence data, much like the prior theoretical methods and unlike the prevailing heuristic methods. Practical run time is, however, substantially lower than that of the existing provably efficient theoretical methods, allowing us to tackle larger and more difficult datasets. Below, we formalize the problem solved, present our methods, and establish their practical value on a selection of real variation data sets. We further document a web server providing open access to this ILP method and serving as a front-end to a database of local phylogenies inferred throughout the autosomal human genome. This work provides a platform for more extensive empirical studies of variation patterns on genomic scales than were previously possible and may also help lay the groundwork for more sophisticated optimization methods that are likely to be needed in the future.

3.1 Preliminaries

We will assume that the input to the problem is a haplotype matrix H where each row corresponds to a haploid sequence of a taxon and each column corresponds to a Single Nucleotide Polymorphism (SNP) site. The input H can therefore be viewed as an $n \times m$ binary matrix. For this chapter, we will use Definition 1 of a phylogeny. Definitions 3, 4, 5 are also useful for this chapter.

As mentioned in the previous chapter the phylogeny reconstruction problem is closely related to the *Steiner Tree Problem*, a well studied problem in combinatorial optimization (for a survey and applications, see [16, 74]). We provide a more formal description of this relationship below, since it is very important for this chapter. Given a graph $G = (V, E)$ and a set of *terminals* in V, the problem is to find the smallest subgraph of G such that there is a path between any pair of terminals. Let graph G be the m-cube defined on vertices $V = \{0, 1\}^m$ and edges $E = \{(u, v) \in V \times V : \sum_i |u_i - v_i| = 1\}$. The vertices are binary strings of length m and an edge connects two vertices if and only if their Hamming distance is 1. Let $V_T \subseteq V$ be the set of species corresponding to the rows of input matrix H. The maximum parsimony problem is then equivalent to the minimum Steiner tree problem on underlying graph G with terminal vertices V_T. Even in this restricted setting, the Steiner tree problem has been shown to be NP-complete [53]. However, the phylogeny reconstruction problem when the optimal phylogeny is q-near-perfect can be solved in time polynomial in n and m when $q = O(\log(\text{poly}(n, m)))$ [106]. If q is very large, though, such algorithms do not perform well. Moreover, these algorithms use a sub-routine that solves the Steiner tree problem on m-cubes when the dimensions are small. Therefore, improving the existing solutions for the general problem will also improve the running time for the restricted cases.

3.2 Preprocessing

We now describe a set of preprocessing steps that can substantially reduce the size of the input data without affecting the final output.

3.2.1 Reducing the set of possible Steiner vertices

The complexity of solving the Steiner tree problem in general graphs is a consequence of the exponentially many possible subsets that can be chosen as the final set of Steiner vertices in the most parsimonious phylogeny. Therefore, an important component of any computational solution to the Steiner tree problem is to eliminate vertices that cannot be present in any optimal tree. We describe an approach that has been used to eliminate such vertices when the underlying graph is the m-cube.

For input graph H and column c of H, the *split* $c(0)|c(1)$ defined by c is a partition of the taxa into two sets, where $c(0)$ is the set of taxa with value 0 in column c and $c(1)$ is the set of taxa with value 1 in column c. This forms a partition of the taxa since $c(0) \cup c(1)$ is the set of all taxa and $c(0) \cap c(1)$ is empty. Each of $c(0)$ and $c(1)$ is called a *block* of c.

function `findBuneman`(V_T)

1. let $\lambda \leftarrow V_T$; let $v \in \lambda$
2. `bunemanNeighbor`(λ, v)

function `bunemanNeighbor`(λ, v)

1. for all $j \in \{1, \ldots, m\}$
   (a) let $v' \leftarrow v; v'_j \leftarrow c_j(1 - i_j)$
   (b) if v' is Buneman and $v' \notin \lambda$ then
      i. $\lambda \leftarrow \lambda \cup \{v'\}$
      ii. `bunemanNeighbor`(λ, v')

Figure 3.1: Finding the Buneman graph in polynomial time

Buneman used the blocks of binary taxa to introduce a graph, now called the *Buneman graph* $\mathcal{B}(H)$, which captures structural properties of the optimal phylogeny [13]. We will explain the generalization of this graph due to Barthélemy [6]. Each vertex of the Buneman graph is an m-tuple of blocks $[c_1(i_1), c_2(i_2), \ldots c_m(i_m)]$ ($i_j = 0$ or 1 for each $1 \leq j \leq m$), with one block for each column and such that each pair of blocks has nonempty intersection ($c_j(i_j) \cap c_k(i_k) \neq \emptyset$ for all $1 \leq j, k \leq m$). There is an edge between two vertices in $\mathcal{B}(H)$ if and only if they differ in exactly one block. Buneman graphs are very useful because of the following theorem.

Theorem 3.2.1 *[5, 100] For input matrix H, let T^*_H denote the optimal phylogeny on H and let $\mathcal{B}(H)$ denote the Buneman graph on H. If matrix H has binary values, then every optimal phylogeny T^*_H is a subgraph of $\mathcal{B}(H)$.*

Using the above theorem, our problem is now reduced to constructing the Buneman graph on input H and solving our problem on underlying graph $\mathcal{B}(H)$. Ideally we would like to find the Buneman graph in time $O(\text{poly}(k))$ where k is the number of vertices in the Buneman graph. Note that this is output-sensitive. We first state the following theorem, which we will use to show the Buneman graph can be generated efficiently.

Theorem 3.2.2 *[100] The Buneman graph $\mathcal{B}(H)$ is connected for any input matrix H in which all columns contain both states 0, 1 and all pairs of columns are distinct.*

To generate the graph $\mathcal{B}(H)$, let $i_1, i_2, \ldots i_m$ be the first taxon in H. Then $v = [c_1(i_1), c_2(i_2), \ldots c_m(i_m)]$ is a vertex of $\mathcal{B}(H)$. Now, there are several ways to generate the graph $\mathcal{B}(H)$. The pseudo-code in Figure 3.1 begins with V_T the set of vertices of the $\mathcal{B}(H)$ corresponding to H. The algorithm then iteratively selects a vertex v and enumerates all the neighbors. For each vertex, the algorithm checks if it obeys the conditions of the Buneman graph, if so it is added to λ and we recurse.

Lemma 3.2.3 *The algorithm in Figure 3.1 finds the Buneman graph $\mathcal{B}(H)$ for the given input in time $O(km)$ where k is the number of vertices in $\mathcal{B}(H)$.*

Proof: The algorithm begins with a vertex $v \in \mathcal{B}(H)$ and determines $\mathcal{B}(H)$ in the depth-first search order. By Theorem 3.2.2, the algorithm will visit all vertices in $\mathcal{B}(H)$. Step 1a iterates over all m possible neighbors of vertex v in the m-cube which takes time

$O(m)$. For each vertex $v \in \mathcal{B}(H)$ function `bunemanNeighbor` is called using v exactly once. Therefore if there are k vertices in $\mathcal{B}(H)$, then the time spent to discover all of $\mathcal{B}(H)$ is $O(km)$. Note that instead of using depth-first search, we could use breadth-first search or any other traversal order.

3.2.2 Decomposition into smaller problems

In addition to allowing us to reduce the set of possible Steiner vertices, we show how Theorem 5.1.2 also allows us to decompose the problem into independent subproblems.

Definition 13 *[4] A pair of columns i, j* conflict *if the matrix H restricted to i, j contains all four gametes $(0,0), (0,1), (1,0)$ and $(1,1)$. Equivalently, the columns* conflict *if the projection of H onto dimensions i, j contains all four points of the square.*

For input I, the structure of the conflicts of I provides important information for building optimal phylogenies for I. For example, it is well known that a perfect phylogeny exists if and only if no pair of columns conflict [57, 100]. In order to represent the conflicts of H, we construct the *conflict graph* $\mathcal{G}$, where the vertices of $\mathcal{G}$ are columns of H and the edges of $\mathcal{G}$ correspond to pairs of conflicting columns [62]. The following theorem has been stated previously without proof [62]. For the sake of completeness, we provide an explicit proof using Theorem 5.1.2 and ideas from Gusfield and Bansal [62]. We denote the matrix H restricted to set of columns C as $C(H)$.

Theorem 3.2.4 *Let χ denote the set of non-trivial connected components of conflict graph $\mathcal{G}$ and let V_{isol} denote the set of isolated vertices of $\mathcal{G}$. Then any optimal Steiner tree on H is a union of optimal Steiner trees on the separate components of $\mathcal{G}$ and $\texttt{length}(T^*_H) = |V_{isol}| + \sum_{C \in \chi} \texttt{length}(T^*_{C(H)})$.*

Proof:

We use the fact that the optimal phylogeny is contained in the Buneman graph and show that the connected components impose restrictions on the set of possible edges in the Buneman graph. For two columns c and c', the block $c(i)$ is the *dominated block* of c with respect to the pair (c, c') if block $c(i)$ is contained in some block of c' (i.e., $c(i) \subset c'(0)$ or $c(i) \subset c'(1)$). Similarly, block $c(i)$ is the *dominating block* of c with respect to the pair (c, c') if $c(i)$ contains some block of c'.

Let C be a component in $\chi \cup V_{isol}$. If C is the only component in $\mathcal{G}$, the theorem follows immediately. Otherwise, we can reorder the columns so that C consists of the first k columns, i.e., $c_1, c_2, \ldots c_k \in C$ and $c_{k+1}, \ldots c_m \notin C$. Recall that for any edge in the Buneman graph $\mathcal{B}(H)$, its endpoints correspond to two m-tuples of blocks which differ in exactly one column; label this edge by the column for which its endpoints differ. For any collection of columns $\alpha_1, \alpha_2, \ldots \alpha_l$, let $T^*_H[\alpha_1, \alpha_2, \ldots \alpha_l]$ denote the subgraph of T^*_H induced by the set of edges labeled by $\alpha_1, \alpha_2, \ldots \alpha_l$. We will characterize all edges in the Buneman graph labeled by columns in C using the following lemma from Gusfield and Bansal [62].

Lemma 3.2.5 *[62] For a column c_i with $i > k$, c_i does not conflict with any column in connected component C, and therefore, exactly one of $c_i(0)$ or $c_i(1)$ is the dominating block in c_i with respect to* every *column in C.*

Let $c_i(l_i)$ $(i > k)$ denote the set of dominating blocks of c_i with respect to C. (It follows that $c_i(1 - l_i)$ is the dominated block in c_i with respect to every column in C).

Any vertex in the Buneman graph is an m-tuple of blocks which have pairwise nonempty intersection. Therefore, an edge e labeled by a column in C, say c_1, must have endpoints in which the blocks of column $c_{k+1}, c_{k+2}, \ldots c_m$, intersect both $c_1(0)$ and $c_1(1)$. This implies the blocks of $c_{k+1}, c_{k+2}, \ldots c_m$ are forced to be the dominating blocks with respect to component C, i.e., the last $m - k$ coordinates of the endpoints of e must be $c_{k+1}(l_{k+1}), c_{k+2}(l_{k+2}) \ldots c_m(l_m)$. Let $\mathcal{B}(C)$ be the subgraph of $\mathcal{B}(H)$ generated by the vertices whose last $m - k$ columns have this form. Then any edge labeled by a column in C has both endpoints in $\mathcal{B}(C)$.

Lemma 3.2.6 *$T_H^*[C] = T_H^*[c_1, c_2, \ldots c_k]$ is an optimal Steiner tree on $\mathcal{B}(C)$.*

Proof: We say that vertex $v \in \mathcal{B}(C)$ is a C-projected terminal vertex if there exists $h \in H$ with the same states as v in columns of C. We first show that any two terminals in $\mathcal{B}(C)$ that are C-projected vertices are connected by a path in $T_H^*[c_1, c_2, \ldots c_k]$. Suppose otherwise and let v_1 and v_2 be two distinct vertices in $\mathcal{B}(C)$ which are not connected by such a path. By definition of T_H^*, there is a path P in T_H^* connecting v_1 to v_2; we can assume that v_1 and v_2 are chosen so that the length of path P is minimized. Let $d_1, d_2, \ldots d_l$ denote the edge labels of P (by assumption, at least one of $d_1, d_2, \ldots d_l$ is not in $\{c_1, c_2, \ldots c_k\}$). If for some i, we have $d_i \in \{c_1, c_2, \ldots c_k\}$, then the endpoints u and w of d_i are in $\mathcal{B}(C)$, and either v_1, u or w, v_2 is a pair that is not connected in $T_H^*[c_1, c_2, \ldots c_k]$, a contradiction to the choice of vertices v_1, v_2.

Therefore, all edge labels d_i are in the set $\{c_{k+1}, c_{k+2}, \ldots c_m\}$. However, since v_1 and v_2 are in $\mathcal{B}(C)$, the final $m - k$ components of these two vertices are $c_{k+1}(l_{k+1})$, $c_{k+2}(l_{k+2})$ $\ldots$ $c_m(l_m)$ by definition. Finally, since there are no edges in P labeled by $c_1, c_2, \ldots c_k$, it follows that v_1 and v_2 are equal in all components, a contradiction.

Therefore, $T_H^*[c_1, c_2, \ldots c_k]$ is a Steiner tree on $\mathcal{B}(C)$ where the set of terminal vertices are the C-projected terminal vertices. Therefore if T_H^* is not optimal, then by removing $T_H^*[c_1, c_2, \ldots c_k]$ from T_H^* and replacing it by a tree of smaller cost, we obtain a Steiner tree for H with smaller cost than T_H^*, a contradiction.

The terminal vertices of $C(H)$ correspond to C-projected terminal vertices of $\mathcal{B}(H)$. Therefore, the above shows that for every connected component C, $T_{C(H)}^*$ is a subgraph of T_H^*. Therefore,

$$\texttt{length}(T_H^*) = \sum_{C \in \chi \cup V_{isol}} \texttt{length}(T_{C(H)}^*) = |V_{isol}| + \sum_{C \in \chi} \texttt{length}(T_{C(H)}^*)$$

This completes the proof of Theorem 3.2.4.

Our decomposition preprocessing step proceeds as follows. We first construct the conflict graph $\mathcal{G}$ for input matrix H and identify the set of connected components of $\mathcal{G}$. We ignore the columns corresponding to the isolated vertices V_{isol} since they each contribute exactly one edge to the final phylogeny. Then the columns corresponding to each connected component c of χ can be used independently to solve for the most parsimonious phylogeny. Our problem is now reduced to input matrices H consisting of a single non-trivial connected component.

3.2.3 Merging Rows and Columns

We now transform the input matrix H to possibly reduce its size. We can remove rows of H until all the rows are distinct since this does not change the phylogeny. Furthermore, we can remove all the columns of H that do not contain both states 0 and 1 since such columns will not affect the size or the topology of the phylogeny. Finally, we will assign weights w_i to column i; w_i is initialized to 1 for all i. We iteratively perform the following operation: identify columns i and j that are identical (up to relabeling 0, 1), set $w_i := w_i + w_j$ and remove column j from the matrix. Notice that in the final matrix H, all pair-wise rows are distinct, all pair-wise columns are distinct (even after relabeling 0, 1), every column contains both 0, 1 and all the columns have weights $w_i \geq 1$. From now, the input to the problem consists of the matrix H along with vector w containing the weights for the columns of H. We can now redefine the length of a phylogeny using a weighted Hamming distance as follows.

Definition 14 *The* `length` *of phylogeny* $T(V, E)$ *is* $\texttt{length}(T) = \sum_{(u,v)\in E} \sum_{i\in D(u,v)} w_i$*, where* $D(u, v)$ *is the set of indices where* u, v *differ.*

It is straight-forward to prove the correctness of the pre-processing step.

Lemma 3.2.7 *The length of the optimal phylogeny on the pre-processed input is the same as that of the original input.*

3.3 ILP Formulation

A common approach for studying the minimum Steiner tree problem is to use integer and linear programming methods. For convenience, we will consider the more general problem of finding a minimum Steiner tree for directed weighted graphs G (we represent an undirected graph as a directed graph by replacing each edge by two directed edges). The input to the minimum directed Steiner tree problem is a directed graph, a set of terminals T and a specified root vertex $r \in T$. The minimum Steiner tree is the minimum cost subgraph containing a directed path from r to every other terminal in T.

For a subgraph S of graph G, we associate a vector $x^S \in \mathbb{R}^E$, where edge variable x^S_e takes value 1 if e appears in the subgraph S and 0 otherwise. A subset of vertices $U \subset V$ is *proper* if it is nonempty and does not contain all vertices. For $U \subset V$, let $\delta^+(U)$ denote the set of edges (u, v) with $u \in U$, $v \notin U$ and for a subset of edges $F \subseteq E$, let $x(F) = \sum_{e\in F} x_e$. Finally, edge-weights are given by $w_e \in R^E$.

The problem of finding a minimum directed Steiner tree rooted at r has previously been examined with an ILP based on graph cuts [7, 81, 118]:

$$\begin{aligned} \min \quad & \textstyle\sum_{u,v} w_{u,v} x_{u,v} \quad \text{subject to} && (3.1)\\ & x(\delta^+(U)) \geq 1 \quad \forall \text{ proper } U \subset V \text{ with } r \in U,\ T \cap \overline{U} \neq \emptyset && (3.2)\\ & x_{u,v} \in \{0, 1\} \quad \text{for all } (u, v) \in E. && (3.3) \end{aligned}$$

Constraints (3.2) impose that r has a directed path to all terminal vertices T. Note that in our phylogenetic tree reconstruction problem, the underlying graph for the problem is

the Buneman graph and any input taxon can be chosen as the root vertex r. Since the Buneman graph may have an exponential number of vertices and edges with respect to the size of the input matrix H, the running time for solving this integer program may be doubly-exponential in m in the worst case.

We develop an alternative formulation based on multicommodity flows [118]. In this formulation, one unit of flow is sent from the root vertex to every terminal vertex. Every terminal vertex except the root acts as a sink for one unit of flow and the Steiner vertices have perfect flow conservation. We use two types of binary variables $f^t_{u,v}$ and $s_{u,v}$ for each edge $(u, v) \in E$. The variables $f^t_{u,v}$ are real valued and represent the amount of flow along edge (u, v) whose destination is terminal t. Variables $s_{u,v}$ are binary variables denoting the presence or absence of edge (u, v). The program is then the following:

$$\begin{array}{lcll} \min & \sum_{u,v} w_{u,v} s_{u,v} & \text{subject to} & (3.4) \\ & \sum_v f^t_{u,v} = \sum_v f^t_{v,u} & \text{for all } u \notin T & (3.5) \\ & \sum_v f^t_{v,t} = 1, \sum_v f^t_{t,v} = 0, \sum_v f^t_{r,v} = 1 & \text{for all } t \in T & (3.6) \\ & 0 \leq f^t_{u,v} \leq s_{u,v} & \text{for all } t \in T & (3.7) \\ & s_{u,v} \in \{0, 1\} & \text{for all } e \in E. & (3.8) \end{array}$$

Constraints (5.6) impose the condition of flow conservation on the Steiner vertices. Constraints (3.6) impose the inflow/outflow constraints on terminals in T. Finally, constraints (3.7) impose the condition that there is positive flow on an edge only if the edge is selected. By the max-flow min-cut theorem, the projection of the solution onto the variables s satisfy constraints (3.2) [81]. The results will thus satisfy the following theorem:

Theorem 3.3.1 *All integer variables of the above linear program are binary and the solution to the ILP gives a most parsimonious phylogenetic tree.*

3.4 Alternative Polynomial-Sized LP Formulation

The preceding ILP requires in the worst case an exponentially large number of variables and constraints. It is, however, possible to formulate this problem with only a polynomial number (in n and m) of variables and constraints. The exponential-sized ILP ultimately proved more efficient in practice than the polynomial-sized ILP and we therefore used that one for our empirical validation. We nonetheless include this alternative formulation because it may prove more promising for future improvements and extensions to more general cases of the Steiner tree problem than will our exponential-sized ILP. Note that preprocessing operations B and C above for the exponential-sized ILP will also be relevant to the polynomial-sized ILP. We will therefore assume we have performed those proprocessing steps and in particular that we have eliminated all redundant rows and columns in the data set.

We will use $h_{i,j}, 1 \leq i \leq n$ to denote the state of the i^{th} taxon at site j of the input matrix H. Note that these are not variables of the linear program. We will use $h_{i,j}, n+1 \leq i \leq 2n$ to represent the state of the i^{th} Steiner vertex at site j. We will therefore use nm such

variables in the ILP. However, T^* might not use n Steiner vertices and therefore we associate binary variables p_i to denote the presence or absence of a Steiner vertex i. We use $2\binom{2n}{2}$ edge selection binary variables $e_{i,j}$ to denote the presence or absence of directed edge (i, j). We want $\sum_{i,j} e_{i,j}$ to be the number of edges in T^*.

To define the distance between a pair of vertices, we need some additional auxiliary variables. We use $\binom{n}{2}m$ variables $c_{i,j,k} = |h_{i,k} - h_{j,k}|$ to denote whether vertices i, j differ at site k. The absolute value for this constraint can be expressed as a linear equation. Now distance $r_{i,j} = \sum_{k=1}^{m} w_k c_{i,j,k}$.

To define the objective, however, we need $\sum_{i,j} e_{i,j} r_{i,j}$ which is quadratic. We can instead achieve the same result by defining the following linear constraint $s_{i,j} \geq r_{i,j} - mw_{max} + mw_{max} e_{i,j}$, where $w_{max} = \max_i w_i$. Now the objective function is simply to minimize $\sum_{i,j} s_{i,j}$.

We, however, need additional constraints to ensure that the output is a tree and it connects all the terminal vertices. First, we have $O(n^2)$ constraints: for all i, j, $\sum_k c_{i,j,k} \geq 1$. We also have $2n$ integer variables d_i representing the *depth* of a vertex i from the root (arbitrarily the first row of H). We ensure that vertex a can connect to another vertex of the phylogeny only if it is of depth one smaller with the constraints that for all i, j, $y_{i,j} - d_i + d_j \geq -1, y_{i,j} + d_i - d_j \geq 1, (2n+1)e_{i,j} + y_{i,j} \leq 2n+1$. Also, $\sum_j e_{i,j} + p_i = 1$ for all i to ensure that there exists only one parent for every vertex (except the root). Finally the constraint $\sum_{i,j} e_{i,j} = n-1$ ensures, that the set of edges selected forms a tree. We now have the following theorem. Putting these components together results in the following ILP:

$$
\begin{array}{lcll}
\min & \sum_{i,j} s_{i,j} & \text{subject to} & (3.9) \\
& c_{i,j,k} \geq h_{i,k} - h_{j,k} & \text{for all } 1 \leq i,j \leq n,\ i \neq j,\ 1 \leq k \leq m & (3.10) \\
& c_{i,j,k} \geq h_{j,k} - h_{i,k} & \text{for all } 1 \leq i,j \leq n,\ i \neq j,\ 1 \leq k \leq m & (3.11) \\
& r_{i,j} = \sum_{k=1}^{m} w_k c_{i,j,k} & \text{for all } 1 \leq i,j \leq n,\ i \neq j & (3.12) \\
& s_{i,j} \geq r_{i,j} - mw_{max} + mw_{max} e_{i,j} & \text{for all } 1 \leq i,j \leq n,\ i \neq j & (3.13) \\
& \sum_k c_{i,j,k} \geq 1 & \text{for all } 1 \leq i,j \leq n,\ i \neq j & (3.14) \\
& y_{i,j} - d_i + d_j \geq -1 & \text{for all } 1 \leq i,j \leq n,\ i \neq j & (3.15) \\
& y_{i,j} + d_i - d_j \geq 1 & \text{for all } 1 \leq i,j \leq n,\ i \neq j & (3.16) \\
& (2n+1)e_{i,j} + y_{i,j} \leq 2n+1 & \text{for all } 1 \leq i,j \leq n,\ i \neq j & (3.17) \\
& \sum_j e_{i,j} + p_i = 1 & \text{for all } 1 \leq i \leq n & (3.18) \\
& \sum_{i,j} e_{i,j} = n-1 & & (3.19)
\end{array}
$$

We further constrain all variables to be non-negative and fix the depth of the root node to be zero.

Theorem 3.4.1 *The above linear program uses polynomial number of variables and constraints and the solution of the ILP is a most parsimonious phylogenetic tree.*

Proof: We have nm variables coding unknown allele values for the n Steiner nodes that might be present in the phylogeny, $(2n)(2n-1)$ edge selection variables identifying

the edges in the phylogeny, $\frac{1}{2}(n)(n-1)m$ auxiliary variables used to measure Hamming distances, $\frac{1}{2}(n)(n-1)$ variables specifying the Hamming distances of selected edges only, $2n$ depth variables, $2n-1$ parent variables, and $(2n)(2n-1)$ auxiliary y_{ij} variables used in setting the depth constraints. The total variable set therefore has size $O(n^2m)$.

We have $2n(n-1)m$ constraints for computing absolute values (lines 10-11), $n(n-1)$ for determining edge costs between nodes (line 12), $n(n-1)$ for determining weights of selected edges (line 13), $n(n-1)$ enforcing that all nodes are connected to the phylogeny (line 14), $3n(n-1)$ for enforcing node depth constraints (lines 15-17), n for ensuring each node has a parent (line 18), and one forcing the phylogeny to have $n-1$ edges and thus to be a tree (line 19). The total number of constraints is therefore also $O(n^2m)$.

The correctness of the program is established in the text above explaining its derivation.

3.5 Enumerating All Solutions

For a lot of phylogenetic analysis one wishes to know if the most parsimonious tree is unique or if there are multiple equally good solutions. In the latter case of course, a compact representation of all possible solutions is very much desirable. We handle both of these problems in this section.

We shall assume in this section that there exists a black-box algorithm A to solve the maximum parsimony phylogeny inference from haplotypes problem with the additional constraint that a set of edges α (of the m-cube) are present and a set of edges β (of the m-cube) are absent. One can either use the above explained linear program to achieve this or adapt other well known techniques to solve this problem [50, 107]. In this manner, we can take any algorithm that returns a single optimal solution and convert it to efficiently find all solutions.

The obvious brute-force method to solve the problem is to enumerate over all possible disjoint sets α, β and run algorithm A. Since there are $2^{m-1}m$ edges in the cube, the number of times A is called is $2^{m2^{m-1}}$. Notice the crucial point here. Even if the solution is unique, the running time of this algorithm still remains the same.

The goal of this section is to develop an output-sensitive algorithm such that the running-time is proportional to the number of solutions.

We now start with a traditional branch-and-bound algorithm. For every edge in the hyper-cube, we can branch by deciding to place it either in α or in β. If the resulting problem is infeasible, then we prune the branch. The algorithm is outlined in Figure 3.2. This is a huge improvement over the previous method. We can analyze the number of calls made to A by the algorithm as follows.

We first obtain an upper bound. At every step the branching algorithm considers an edge e. We count the number of times when the algorithm obtains feasibility both when e is added to α and when it is added to β. This means that, at the current stage of branching, there exists at least one solution that contains edge e and at least one solution that does not contain edge e. We can call such steps of the branching algorithm 'b-steps' since both branches of the resulting tree needs to be explored. Since the number of internal nodes of any tree with degree greater than 1 is at most the number of leaves, the number of b-steps

```
EnumerateSolutionsBB(α, β)
  1. if (A(α, β) is infeasible) return Φ
  2. if (∃e ∉ α ∪ β)
     (a) return EnumerateSolutions(α∪{e}, β) ∪
         EnumerateSolutions(α, β ∪ {e})
  3. else return solution A(α, β)
```

Figure 3.2: Branch-and-bound to enumerate all solutions

```
EnumerateSolutionsFast(α, β, integer d)
  1. S ← A(α, β)
  2. R := {S}
  3. for i = |E| to d
     (a) s_i = ¬s_i
     (b) α := {e_j | s_j = 1, j ≤ i} and β := {e_j | s_j = 0, j ≤ i}
     (c) if (A(α, β) is feasible)
         i. R := R∪ EnumerateSolutionsFast(α, β, i)
  4. return R
```

Figure 3.3: An algorithm that is twice as fast as branch-and-bound.

of the branching algorithm is at most k, the number of solutions. Removal of the b-steps from the branch-and-bound tree leaves us with a set of disjoint tree where each node has at most one child that is 'feasible'. The depth of each tree is bounded by the depth of the whole branch-and-bound tree which is $m2^{m-1}$. Therefore, the total number of nodes in all the disjoint trees combined is at most $2m2^{m-1}k$. Including the b-steps, therefore the total number of calls to A is at most $2m2^{m-1}k+k$. It is easy to obtain a lower-bound of $2m2^{m-1}$ since this is exactly the number of nodes in the tree when there is a unique solution.

Although, the branch-and-bound method does offer an output sensitive method, we now show that it is possible to significantly improve the running time.

The faster algorithm is outlined Figure 3.3. While it seems unlikely that the branch-and-bound can be sped up in the worst case, we show that this is possible if we use the solution returned by the A instead of just checking feasibility. By exploiting the structure of the solution, we show that the improved algorithm makes fewer calls to A.

For this algorithm, we assume that the set of all edges in the m-cube is denoted by E and the elements $e_i \in E$ are (arbitrarily) ordered. We also assume that if there is a feasible solution, then black-box algorithm A returns a set S such that s_i is set to 1 if and only if edge e_i is present in the solution.

First, we show the correctness of the algorithm by proving that every feasible solution is obtained. First, it is easy to note that for any function call, $|\alpha \cup \beta| = d$ and α and β are disjoint.

Lemma 3.5.1 *The algorithm provided in Figure 3.3 finds every feasible solution.*

Proof: For the sake of contradiction, assume that $S' \in \{0,1\}^{|E|}$ is a solution that was not found by the routine. During the execution of the routine let (α', β', d') be the parameters of the function call such that d' is the largest and the constraints imposed by α', β' does not invalidate solution S'. Note that such parameters always exist since both $(\{e_1\}, \Phi, 1)$ and $(\Phi, \{e_1\}, 1)$ are function calls that will always be made. Clearly, by the assumption then $A(\alpha', \beta', d')$ returns a feasible solution $S'' \neq S'$. Let p be the lowest index such that $s''_p \neq s'_p$. Due to the constraints imposed, necessarily $p > d'$. Therefore during the `for` loop, there will be an iteration when $i = p$ and at this iteration, algorithm A has to return feasibility (since S'' is feasible) and therefore during the recursive call $i + 1 = p + 1 > d'$ contradicting the assumption that d' was the largest such value.

Lemma 3.5.2 *The number of calls to A made by the algorithm in Figure 3.3 is at most $km2^m$ where k is the number of solutions.*

Proof: A recursive call is made only when a new feasible solution is found. Hence, the number of function calls is at most the number of feasible solutions which can be denoted by k. The number of calls to A made by each function call to `EnumerateSolutionsFast` not including recursive calls is at most $|E| = m2^m$. Therefore the total number of calls made to A is at most $m2^m k$.

The above proof shows that the new algorithm is twice as fast as the branch-and-bound algorithm.

3.6 Empirical Validation

Experience with both ILPs showed the exponential-sized one to be generally the more efficient in practice and we therefore used that variant for our empirical studies. We applied the ILP to several sets of variation data chosen to span a range of data characteristics and computational difficulties. We used only non-recombining data (Y chromosome, mtDNA, and bacterial DNA) because imperfection in non-recombining data is most likely to be explained by recurrent mutations. We used two Y chromosome data sets: a set of all human Y chromosome data from the HapMap [75] and a set of predominantly chimpanzee primate data [111]. Several different samples of mitochondrial DNA(mtDNA) were also included [1, 15, 96, 113]. Finally, we analyzed a single bacterial sample [80].

The pre-processing and ILP formulation was performed in C++ and solved using the Concert callable library of CPLEX 10.0. In each case, the ILP was able to find an optimal tree on the data after preprocessing. We also used the `pars` program of `phylip` which attempts to heuristically find the most parsimonious phylogeny. `pars` was run with default parameters. Empirical tests were conducted on a 2.4 GHz Pentium 4 computer with 1G RAM, running Linux. We attempted to use the `penny` program of `phylip`, which finds provably optimal solutions by branch-and-bound, but it terminated in under 20 minutes only for the smallest mitochondrial data set and was aborted by us after 20 minutes for all other tests.

We first used the mitochondrial data as a basic validation of the correctness of the methods and the reasonableness of the maximum parsimony criterion on these data. The

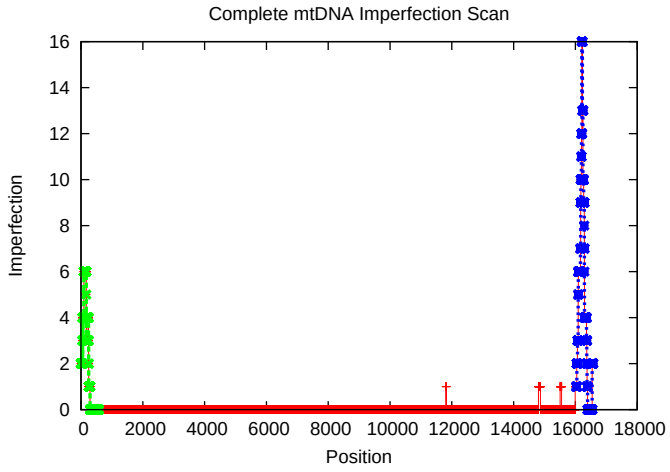


Figure 3.4: Imperfection of the most parsimonious phylogeny for overlapping windows across the complete mitochondrial genome. The x-axis shows the sites in their order along the genomic axis. The y-axis shows the imperfection for the window centered on the corresponding site. The hyper variable D-loop region ($1 \ldots 577$ and $16028 \ldots 16569$) shows significantly larger imperfection.

HVS-I and HVS-II segments of the mitochondrial D-loop region have exceptionally high mutation rates [113], providing a good test case of the ability of our algorithm to distinguish regions we would expect to have perfect or near-perfect phylogenies from those expected to have highly imperfect phylogenies. Figure 3.4 shows a scan of 201-site long windows across the complete 16569-site mtDNA genome. Since the mtDNA is circular, the windows wrap around over the ends in the genome order. The y-axis corresponds to *imperfection*, which is the number of recurrent mutations in the most parsimonious phylogeny. The figure does indeed show substantially larger phylogenies within the high mutation rate D-loop region ($1 \ldots 577$ and $16028 \ldots 16569$) than in the low mutation rate coding regions, confirming the relevance of a parsimony metric for such data sets.

We then ran the ILP on a collection of data sets to assess its efficiency. Figure 3.5 provides two examples of most parsimonious phylogenies for data sets at opposite extremes of difficulty: an mtDNA sample [113] with imperfection 21 (Fig. 3.5(a)) and the human Y chromosome sample, with imperfection 1 (Fig. 3.5(b)). Table 3.1 presents the empirical run-time data for all of the datasets. The columns 'input before' and 'input after' correspond to the size of the original input and that after preprocessing. Run times vary over several orders of magnitude and appear largely insensitive to the actual sizes of the data sets. Rather, the major determinant of run time appears to be a dataset's imperfection, i.e., the difference between the optimal length and the number of variant sites. It has recently been shown that the phylogeny problem under various assumptions is fixed parameter tractable in imperfection [9, 9, 51, 106] possibly suggesting why it is a critical factor in run time determination. The `pars` program of `phylip`, despite providing no guarantees of optimality, does indeed find optimal phylogenies in all of the above instances. It is, however, slower than the ILP in most of these cases.

	input			time(secs)	
Data Set	before	after	length	our ILP	`pars`
human chrY [75]	150×49	14×15	16	0.02	2.55
bacterial [80]	17×1510	12×89	96	0.08	0.06
chimp mtDNA [111]	24×1041	19×61	63	0.08	2.63
chimp chrY [111]	15×98	15×98	99	0.02	0.03
human mtDNA [113]	40×52	32×52	73	13.39	11.24
human mtDNA [1]	395×830	34×39	53	53.4	712.95
human mtDNA [96]	13×390	13×42	48	0.02	0.41
human mtDNA [15]	44×405	27×39	43	0.09	0.59

Table 3.1: Empirical run-time results on a selection of non-recombining datasets.

3.7 Online Tool

In order to provide more general access to our methods, we have implemented a web server based on our worst-case exponential-sized ILP. The server provides a front end to a an implementation of the ILP in C++ using the CPLEX 10 libraries. We call the server SCan for IMperfect Phylogenies (SCIMP). It can be accessed at `http://www.cs.cmu.edu/~imperfect/index.html`. There are two ways to use the web-server as explained below.

Firstly, the users can input a haplotype variation data-set. These are simply a set of n haplotype sequences typed over m SNPs. As stated in the previous sections, this has to be phased data. Therefore essentially the input is an $n \times m$ $\{0, 1\}$ matrix.

Alternatively, the users can select any region of the genome and provide the number of contiguous SNPs to examine in that region. The user also needs to specify the population group to use. The webserver currently has support for the Central European population (CEPH) and Yoruba African population (YRI). The entire HapMap (phase II) phasing data is present in the webserver's backend database and this makes it easy for users to quickly examine and construct phylogenies for any region of interest. Since the HapMap data for these two populations were sequenced in trios, the number of phasing errors should be very small.

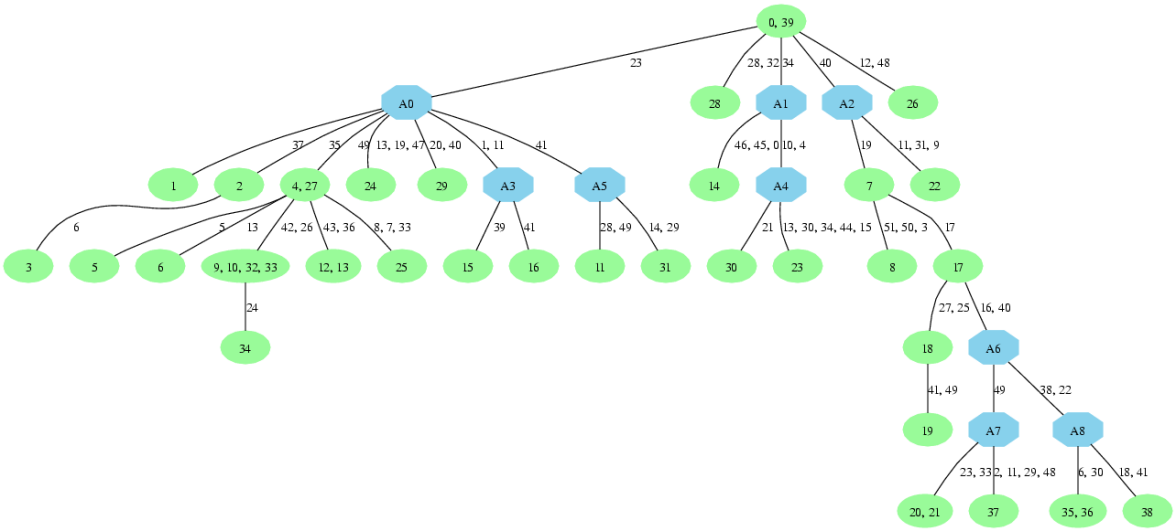
The webserver can be used in two different modes. As has been described until now, the user can just request it to construct the most parsimonious phylogenetic tree and return the topology, the parsimony score (number of mutations) and the imperfection (number of recurrent mutations).

The webserver can also perform an imperfection scan. The user specifies the location and size and additionally for this mode provides a window length w in the number of SNPs. The webserver then slides this window across the genome and for each overlapping set of w consecutive SNPs constructs a maximum parsimony phylogeny. The server returns to the user a plot of the imperfection (number of recurrent mutations) of each of these windows across the entire region examined. It can further provide the maximum parsimony tree found within each window.

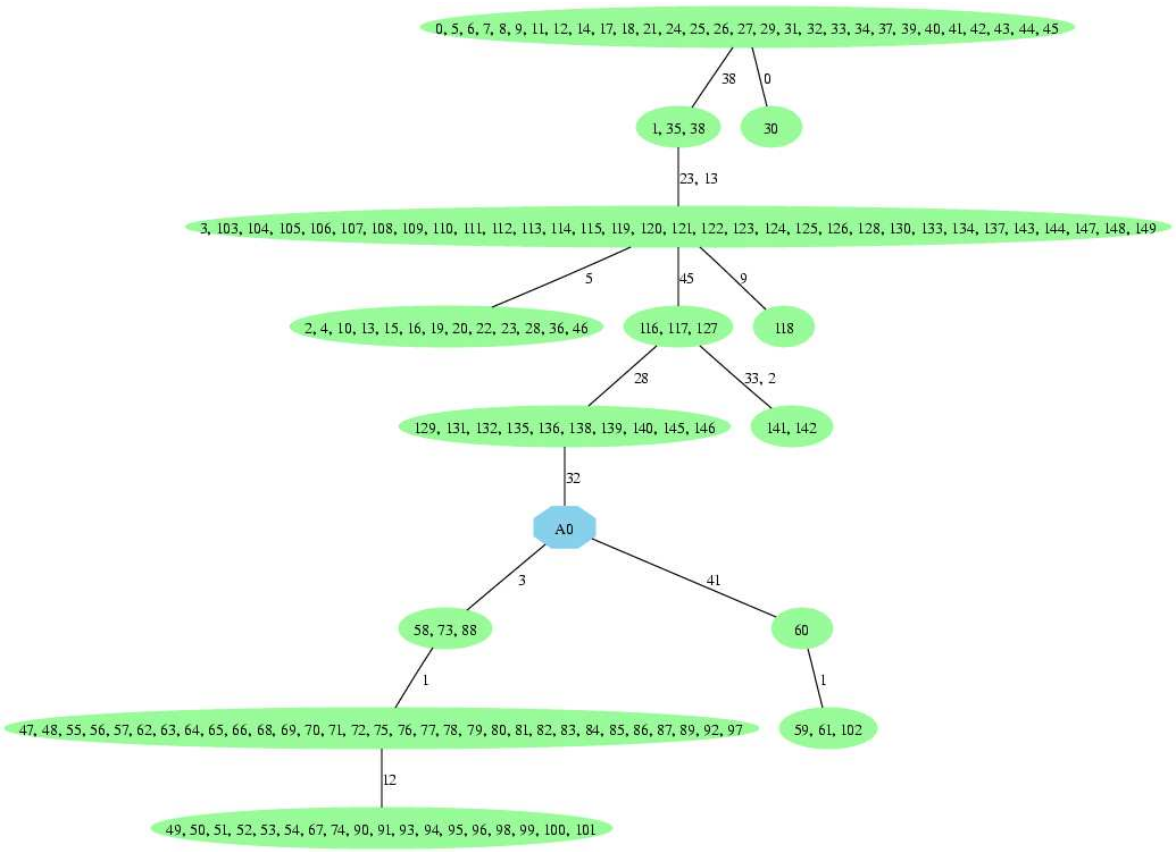
In addition to providing a general interface to the phylogeny inference code, the server also houses a precomputed database of maximum parsimony phylogenies that it constructed offline for more than 3.7 million instances using the HapMap SNPs. Therefore, when users request to see phylogenies that are present in this precomputed data-set (or while performing scans), the results are returned as soon as they are fetched with no online solution required. This precomputed databases currently has phylogenies for every contiguous region of up to 10 SNPs in all of HapMap. Figure 3.5 provides examples of the output. More low-level details can be obtained from the Appendix.

3.8 Conclusion

We have developed ILP formulations for optimally solving for the most parsimonious phylogeny using binary genome variation data. These methods fill an important practical need for fast methods for generating provably optimal trees from large SNP variation datasets. This need is not served well by the heuristic methods that are currently the standard for tree-building, which generally work well in practice but cannot provide guarantees of optimality. More recent theoretical methods that find provably optimal trees within defined run-time bounds are inefficient in practice without a fast sub-routine to solve the general problem on smaller instances. The ILP approach allows extremely fast solutions of the easy cases while still yielding run-times competitive with a widely used fast heuristic for hard instances. Such methods are likely to be increasingly important as data sets accumulate on larger population groups and larger numbers of variant sites.



(a)



(b)

Figure 3.5: Examples of phylogenies of varying levels of difficulty. Nodes labeled with numbers correspond to the numbered input haplotypes, while those labeled A# correspond to inferred Steiner nodes. Edges are labeled with the site variations to which they correspond. (a) Human mitochondrial data from Wirth et al. [113] (b) Human Y chromosome from HapMap [75]

Chapter 4

Near-Perfect Phylogeny Reconstruction from Genotypes

In this chapter, we continue to discuss algorithms for reconstructing phylogenies from SNP sequences. However, we will now assume that the input is presented as *genotypes*. As we will shortly see, this increases the difficulty of the problem significantly. Since the sequences of the reconstructed phylogeny are haplotypes, this algorithm also provides a solution for the *haplotype inference (phasing)* problem.

The problem of haplotype inference has benefited tremendously from contributions from the fields of discrete algorithms and combinatorial optimization. In haplotype inference, also called *phasing*, one seeks to separate the alleleic contributions of two chromosomes observed together in a genotype assay. If we use the symbols 0 and 1 to represent homozygous and 2 to represent heterozygous alleles then '0221' is a genotype typed on four SNP sites. Two pairs of haplotypes 0001, 0111 and 0011, 0101 are both consistent with the genotype and the goal of phasing is to correctly infer the true haplotypes, given the genotypes[1]. The problem has relevance to basic research into population histories as well as to applied problems such as statistically linking haplotypes to disease phenotypes.

The field of haplotype inference began with a fast and simple iterative method based on the idea that a haplotype that is observed in one individual is likely to be found in other individuals as well [18]. Thus, one can identify a few "seed" haplotypes from individuals identical on both chromosomes, find others for which the genotype is consistent with some known haplotype, and find the other implied haplotype for each such genotype, repeating until all possible inferences have been made. This approach is fast and simple, but often fails to resolve all haplotypes, particularly in the presence of large or noisy data sets. Various statistically motivated algorithms based on heuristic optimization techniques such as expectation maximization [87] and Markov chain Monte Carlo methods [110] have since been developed. Such algorithms provide significantly improved robustness and accuracy, although still poor scaling in problem size. Several combinatorial optimization methods have also been formulated based on Clark's original method [61], although these have all

[1]Note: In some prior literature, '1' has been used for heterozygous. These are merely labels and they do not affect the problem. In this chapter, we find it convenient to use '1' for homozygous since it allows us to more easily describe the conversion of a genotype matrix into a haplotype matrix in certain cases.

been intractable in theory and practice. The advent of large-scale genotyping created a need for new methods designed to scale to chromosome-sized data sets.

Recently, a new avenue towards haplotype reconstruction was developed based on phylogenetic methods. In such an approach, one seeks to identify the ancestral history that could have produced a set of haplotype sequences from a common ancestor such that the inferred haplotypes could give rise to the observed genotypes. Gusfield [59] showed that it is possible to directly and efficiently infer phylogenies that could explain an observed set of genotypes provided the phylogenies are *perfect*, meaning that each character mutates only once from the common ancestor over the entire phylogeny. This problem is referred to as *Perfect Phylogeny Haplotyping* (PPH). Several subsequent results simplified and improved on this original method [3, 4, 25, 28]. The work produced a fast, practical method for large-scale haplotyping, in which one breaks a large sequence into blocks consistent with perfect phylogenies and uses the phylogenies to phase those sequences.

The perfect phylogeny assumption is quite restrictive, though, and several approaches have been taken to adapt the perfect phylogeny method to more realistic models of molecular evolution. Data inconsistent with perfect phylogenies can arise from multiple mutations of a base over the history of a species or through the processes of recombination or gene conversion, which can assemble hybrid chromosomes in which different segments have different phylogenies. Heuristic methods have allowed some tolerance to recurrent mutations (for example, the work by Halperin and Eskin [65]) resulting in the generation of *imperfect phylogenies*. Imperfect phylogeny haplotyping has proven to be very fast and competitive with the best prior methods in accuracy. Some recent work has been directed at provably optimal methods for imperfect phylogeny haplotyping (IPPH), resulting in a polynomial-time algorithm for the case when a single site is allowed to mutate twice (or a single recombination is present), under a practical assumption on the input data [104]. But the general IPPH problem remained open. It appears for the moment intractable in both theory and practice to infer recombinational histories in non-trivial cases.

Outline of the chapter. In this chapter, we solve the general IPPH problem and show for the first time that it is possible to infer imperfect phylogenies with any constant number q of recurrent mutations in polynomial time in the number of sequences and number of sites. Our approach builds on both the prior theory on phylogeny construction from genotype data [28, 104] and a separate body of theory on the inference of imperfect phylogenies from haplotype data [9, 51, 62, 73, 106]. Our algorithm reconstructs a *q-near-perfect phylogeny* in time $nm^{O(q)}$ where m is the number of characters (variant sites typed), n is the number of taxa (sequences examined), and q is the *imperfectness* of the solution, defined below. The prior method of Song et. al. [104] that solves the 1-near-perfect phylogeny haplotyping problem relied on an empirically but not theoretically supported assumption that an embedded perfect phylogeny problem will have a unique solution. We relax this assumption to allow a polynomial number of such solutions and show that the relaxed assumption holds with high probability if the population obeys a common assumption of population genetics theory called Hardy-Weinberg equilibrium. We thus provide a theoretical basis for Song et al.'s empirical observation. We demonstrate and validate our algorithm by comparing with leading phasing methods using a collection of large-scale genotype data of

known phase [42]. We find that our method is efficient and more accurate on blocks with small q. We further provide strong empirical support for the value of continuing research into accelerated algorithms for phylogeny construction from genotype data. The findings presented in this chapter were initially reported by Sridhar et. al. [105].

4.1 Preliminaries

We begin by introducing definitions for parsimony-based phylogeny reconstruction. In such problems, we wish to study the relationship between a set of n taxa, each of which is defined over a set of m binary characters. Input I will be represented by a matrix, where row set $R(I)$ represent taxa and column set $C(I)$ represent characters. In a *haplotype matrix* all taxa $r_i \in \{0,1\}^m$ and in a *genotype matrix* $r_i \in \{0,1,2\}^m$. Section 1.3.1 of the introductory chapter contains preliminary definitions for phylogeny reconstruction based on genotypes which we will use here. Furthermore, we will adhere to definition 1 of a phylogeny where the Hamming distance of adjacent vertices is always 1.

Definition 15 *For edge $e = (u, v)$ of a phylogeny $\mu(e)$ represents the mutation on the edge, i.e. $l(u)[\mu(e)] \neq l(v)[\mu(e)]$*

We say that a character i mutates on edge e if $\mu(e) = i$. We will assume that both states $0, 1$ are present in all characters. Therefore the length of an optimum phylogeny is at least m. This provides a natural motivation for the *penalty* of a phylogeny as defined above. For simplicity, we will drop the label function $l(v)$ and use v to refer to both a vertex and the taxon it represents.

The IPPH problem. The input to the problem is an $n \times m$ matrix G, where each row $g_i \in \{0,1,2\}^m$ represents a (genotype) taxon and each column represents a character. The output is a $2n \times m$ matrix H in which each row $h_i \in \{0,1\}^m$ represents a haplotype. Furthermore corresponding to every taxon $g_i \in R(G)$, there are two taxa $h_{2i-1}, h_{2i} \in R(H)$ with the following properties.

- if $g_i[c] \neq 2$ then $h_{2i-1}[c] = h_{2i}[c] = g_i[c]$
- if $g_i[c] = 2$ then $h_{2i-1}[c] \neq h_{2i}[c]$

The objective of the IPPH problem is to find an output matrix H such that the length of the optimum phylogeny on H is minimized. This problem is clearly NP-hard, since if matrix G contains no 2s, then the problem is equivalent to reconstructing the most parsimonious phylogenetic tree [53]. We therefore consider the following parameterized version of the problem. Given matrix G and parameter q, we return matrix H such that there exists an optimal phylogeny T^* on H with $\textit{penalty}(T^*) \leq q$, under the assumption that such a matrix H exists. Note that the PPH problem is a restriction of IPPH when $q = 0$.

Definition 16 *For a set of binary state taxa S and a set of characters C, the set of* gametes *$GAM(S, C)$ is the projection of S on characters C. In other words $(x_1, \ldots, x_{|C|}) \in GAM(S, C)$ i.f.f. there exists $s \in S$ with $s[c_i] = x_i$ for all $c_i \in C$. A set of characters C* shares *k* gametes *if $|GAM(S, C)| = k$. A pair of characters i, j* conflict *if*

$|GAM(S, \{i, j\})| = 4$.

Pre-processing. We perform a well established pre-processing step that ensures that for any optimal output matrix H, $(0, 0) \in GAM(H, \{i, j\})$ for all $i, j \in C(H)$ (See the work of Eskin et al. [28]). We assume that the input matrix has no duplicate rows, since such rows do not change the optimal solution. Note that such a matrix should have at most $q + 1$ more taxa than characters, since otherwise, it does not have a solution to the IPPH problem.

4.2 Algorithm

At a high level, our algorithm has the same spirit as the algorithm of Song et. al. [104]. The algorithm guesses characters that mutate more than once and removes them from the input matrix. It then solves the perfect phylogeny haplotyping problem on the remainder of the matrix. Finally, our algorithm adds the removed characters back and performs haplotyping on the full matrix. In this section, we will use the same assumption of Song et. al. [104]: for any subset of characters of the input matrix, if a solution to perfect phylogeny haplotyping (PPH) exists, then it is unique. In Section 4.3, we show that the number of PPH solutions is polynomial with high probability.

Throughout the paper, we fix an arbitrary optimal phylogeny T^*, which we will use as a reference for expository purposes. Let $t(T^*)$ be the set of all taxa (Steiner vertices included) in T^*. Let Q be the set of characters that mutate more than once in T^*. Since $|Q| \leq q$, we can find Q by brute force in time $O(\sum_{i=1}^{q} \binom{m}{i}) = O(qm^q)$.

After finding Q, we can remove the characters to obtain matrix M with character set $C(G) \setminus Q$. We now use a prior method to solve the perfect phylogeny haplotyping (PPH) problem on M in time $O(nm)$ [25]. Let M' be the unique solution to the PPH problem. The solution matrix M' contains $2n$ taxa and $m - |Q|$ characters. We can now add the characters Q to matrix M' to obtain H'. For all $j \in Q$ and $h'_{2i-1}, h'_{2i} \in H'$ and $g_i \in G$, taxa $h'_{2i-1}[j] = h'_{2i}[j] = g_i[j]$. Note that matrix H' contains state 2 only on the characters Q (see Fig 4.2(b) for an example).

Definition 17 *A pair of taxa $h'_{2i-1}, h'_{2i} \in R(H')$ is defined as a* couple. *For any $c \in C(H')$, if $h'_{2i-1}[c] = 2 (= h'_{2i}[c])$ or $h'_{2i-1}[c] \neq h'_{2i}[c]$ then h'_{2i-1}, h'_{2i} is a $(2, c)$-couple.*

Note that if c contains both 0 and 1 in a couple of H', then the couple contained state 2 at character c in the input (before perfect phylogeny haplotyping). Let $\kappa = \{i \in C(G) | \exists j \in C(G), |GAM(t(T^*), \{i, j\})| = 4\}$ be the set of characters that conflict with some other character in $t(T^*)$. Note that $Q \subseteq \kappa$. To complete the description and analysis of the algorithm we borrow the following definition from prior work [28].

Definition 18 *We say that a (unordered) couple r_1, r_2 induces (x, y) at characters (c, c') if $r_i[c] = x, r_i[c'] = y$ or $r_1[c] = r_2[c] = 2, r_1[c'] = r_2[c'] = y$ or $r_1[c] = r_2[c] = x, r_1[c'] = r_2[c'] = 2$. We define $IND(H', \{c, c'\})$ to denote the set of* gametes induced *by the couples of H' at c, c'.*

The goal now is to convert the {0, 1, 2} matrix H' to a {0, 1} matrix H such that the following **correctness conditions** are satisfied:

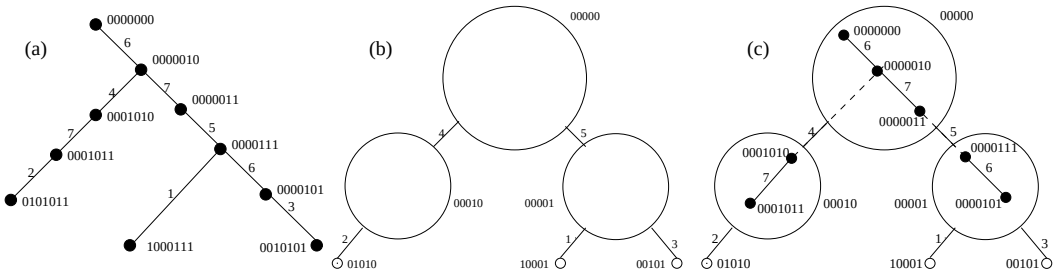


Figure 4.1: Algorithm to determine κ and $GAM(t(T^*), \kappa)$ efficiently. (a) optimal phylogeny T^* with $Q = \{6, 7\}, \kappa = \{4, 5, 6, 7\}$(b) perfect phylogeny T on characters $C(G) \setminus Q$ (c) the four edges mutating characters 6 and 7 are assigned to three vertices of T; rooted trees T_v are constructed on the assigned edges; states on the three roots $0000000, 0001010, 0000111$ determine edges $\{4, 5\}$ that lie between two mutations of $6, 7$. Filled vertices form V_κ.

1. for every $(2, c)$-couple in H' one of the two taxa should contain state '1' and the other '0' on character c in H
2. $GAM(H, \kappa) \subseteq GAM(t(T^*), \kappa)$, i.e. the set of gametes on κ in H is a subset of the set of gametes on κ in T^*.
3. $(|GAM(H, \{c, c'\})| = 4) \Longrightarrow c, c' \in \kappa$, i.e. a pair of characters share four gametes in H only if they are both in κ.

In matrix H', if a couple contains state 2 at character c, then replacing it with state 0 on one taxa and 1 on the other is informally referred to as a *resolution*. The algorithm to solve IPPH is summarized in Figure 4.4. We now go into the details of the steps. The following lemma shows that Steps 5 and 6 of function `solveIPPH` can be implemented efficiently:

Lemma 4.2.1 *Sets κ and $GAM(t(T^*), \kappa)$ can be found in time $m^{2q}q^{O(q)} + O(nm)$.*

Proof: We can easily identify κ by brute force in time $O(m^{|\kappa|})$. Since we do not know $|\kappa|$, this step can take time $O(2^m)$. We can however do better by performing such an enumeration over phylogenies as illustrated in Figure 4.1. First we construct the unique perfect phylogeny T for the matrix H' restricted to $C(G) \setminus Q$ which contains $m - |Q| + 1$ vertices in time $O(nm)$ [25]. Note that contracting edges $e \in T^*$ with $\mu(e) \in Q$ results in tree T (see Figure 4.1(b)). We will begin with T and add the edges $e, \mu(e) \in Q$ to obtain T^* as follows. Since we know Q already, we can identify the set of $|Q| + q$ edges (labels μ) in time $O(\binom{|Q|+q}{q}) = O(\binom{2q}{q}) = O(4^q)$. There are $m - |Q| + 1$ locations to add an edge that mutates a character in Q. All possible edge assignments can be enumerated in time $O((m - |Q| + 1)^{|Q|+q}) = O(m^{2q})$. Each enumeration assigns a set of edges (multi-set on characters) Q_v to each vertex v of T. We now enumerate all possible rooted trees T_v with edge labels in Q_v for all vertices $v \in T$ in time $O((|Q| + q)^{|Q|+q}) = O((2q)^{2q})$. Since the mutations in $C(G) \setminus Q$ are already fixed by the perfect phylogeny, the states in all vertices on characters $C(G) \setminus Q$ are known. For every root of T_v, we guess the states in all Q characters in time $O(2^{q^2})$, which can be improved to $O(q^q)$ (since this is equivalent to enumerating all tree structures with edges mutating Q). Note that since we guessed the

states at the root of T_v, we know the states on all characters for all vertices in T_v (for all v). Further, for any two roots of $T_v, T_{v'}$, the path that connects them is given by the path connecting v, v' in the perfect phylogeny T (see Figure 4.1(c)). Therefore, we can identify the edges that lie between two mutations of a character in Q. We can now find κ since for all $i \in \kappa \setminus Q$, there exists $j \in Q$ and $e_1, e_2, e_3 \in T^*$ such that $\mu(e_1) = \mu(e_3) = j$, $\mu(e_2) = i$ and e_2 lies in the path connecting e_1, e_3.

Since we know states in all characters of T_v and T'_v, we know the states in all the characters for every vertex in the path connecting them as well. We now consider the set of all vertices V_κ that lie in the path connecting two mutations of the same character in T. It is easy to see that $GAM(V_\kappa, \kappa) = GAM(t(T^*), \kappa)$. We can therefore identify κ and $GAM(t(T^*), \kappa)$ in time $m^{2q}q^{O(q)} + O(nm)$.

Figure 4.2 illustrates how the algorithm performs haplotyping given κ and $GAM(t(T^*), \kappa)$. For reference Figures 4.2(a) and 4.2(b) represent input matrix G and matrix H' as found in Step 4 of `solveIPPH`. Function `solveIPPH` selects $c = 2$ (Step 7). Function `processMatrix` determines $\mathcal{G}(2, 6) = IND(H', \{2, 6\}) = \{(0, 0), (0, 1), (1, 1)\}$ and guesses $\mathcal{G}(2, 7) = \{(0, 0), (0, 1), (1, 1)\}$ (Step 2(b)iA). Based on these two sets of three gametes, the $(2, 2)$-couples, rows 9 to 12, are resolved (Step 2(b)iB), Fig 4.2(c). Character 2 is removed (Step 2d). Notice that a character c is removed only when all the $(2, c)$-couples are completely resolved (rows 9 to 12). Function `solveIPPH` then selects $c = 3$ (Step 7), Fig 4.2(d). Function `processMatrix` guesses $\mathcal{G}(3, 6) = \{(0, 0), (1, 0), (0, 1)\}$ and determines $\mathcal{G}(3, 7) = IND(H', \{3, 7\}) = \{(0, 0), (0, 1), (1, 1)\}$ (Step 2(b)iA), and using them $(2, 3)$-couple, rows 7 and 8, are resolved(Step 2(b)iB), Fig 4.2(e). Since the pair of rows (7, 8) is also a $(2, 1)$-couple character 1 is added to Δ (Step 2(b)iC). Character 3 is removed (Step 2d) and $c = 1$ is extracted from Δ (Step 2a), Fig 4.2(f). Since there are no $(2, 1)$-couples, character 1 is removed (Step 2d), Fig 4.2(g). This exhausts all $c \in C(H') \setminus \kappa$. Function `solveIPPH` then resolves state 2 in the first couple resulting in $0000, 0010$ which are both in $GAM(t(T^*), \kappa)$ (Step 8a), Fig 4.2(h). Since the next couple has no state 2, we resolve the third couple which results in $0101, 0111$ (Step 8a), completing the algorithm, Fig 4.2(i). We now prove the main lemma that bounds the running time of our algorithm.

Theorem 4.2.2 *If* `penalty`$(T^*) \leq q$*, then the algorithm described in Figure 4.4 returns a solution matrix H that obeys all correctness conditions in time $nm^{O(q)}$.*

Proof: To prove this theorem, we first need three simple lemmas.

Lemma 4.2.3 *If $c_1 \in \kappa$ and $c_2 \in C(H') \setminus \kappa$, then c_1, c_2 share exactly three gametes in $t(T^*)$.*

Proof: Every pair of characters share at least three gametes in T^*. Characters, c_1, c_2 cannot share four gametes since $c_2 \notin \kappa$.

Lemma 4.2.4 *For a pair of characters $c \notin \kappa, \hat{c} \in Q$, given a set of three gametes $\mathcal{G}(c, \hat{c})$, there exists a unique resolution of state 2 in character $\hat{c}$ for any $(2, c)$-couple s.t. for resulting matrix H', $GAM(H', \{c, \hat{c}\}) \subseteq \mathcal{G}(c, \hat{c})$*

Proof: Let $r_{2i-1}, r_{2i} \in R(H')$ be a $(2, c)$-couple. By definition, $r_{2i-1}[c] \neq r_{2i}[c]$. If $r_{2i-1}[\hat{c}] = r_{2i}[\hat{c}] = 2$ then the resolution will either create $GAM(\{r_{2i-1}, r_{2i}\}, \{c, \hat{c}\}) = \{(0, 0), (1, 1)\}$ or $\{(0, 1), (1, 0)\}$. Only one of the two can be contained in set $\mathcal{G}(c, \hat{c})$ established between c and $\hat{c}$.

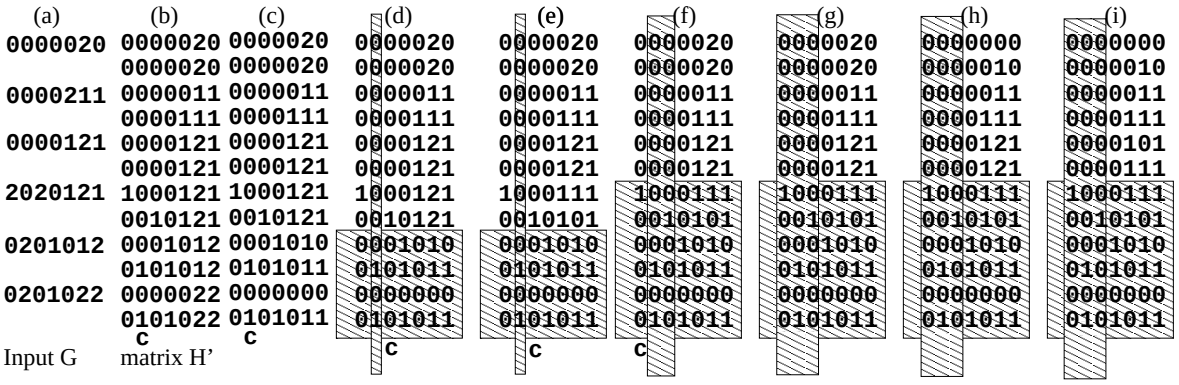


Figure 4.2: Given $Q = \{6, 7\}$, $\kappa = \{4, 5, 6, 7\}$, $GAM(t(T^*), \kappa) = \{0000, 0010, 0011, 1011, 1010, 0111, 0101\}$, the algorithm chooses $c = 2, 3, 1$. Based on a set of three gametes, it resolves states 2 in $\hat{c} \in Q$, in all $(2, c)$-couples. Shaded regions represent deleted (or ignored) characters and couples completely resolved by the algorithm. After exhausting all characters $c \in C(G) \setminus \kappa$, the algorithm considers the remaining couples and iteratively resolves states 2 s.t. the couples are in $GAM(t(T^*), \kappa)$.

Lemma 4.2.5 *In matrix H', for $c \notin \kappa, \hat{c} \in Q$ if every $(2, c)$-couple is in $\{0, 1\}^m$ then $GAM(H, \{c, \hat{c}\})$ is fixed where H is any matrix obtained from H' by resolution of 2s.*

Proof: For any couple in H' if $h'_{2i-1}[\hat{c}] = h'_{2i}[\hat{c}] = 2$ then according to the condition of the lemma, $h'_{2i-1}[c] = h'_{2i-1}[c] = s$. Therefore, for any couple h_{2i-1}, h_{2i} in H obtained by resolving state 2 on $\hat{c}$ will have the property that $GAM(\{h_{2i-1}, h_{2i}\}, \{c, \hat{c}\}) = \{(s, 0), (s, 1)\}$.

Corollary 4.2.6 *Although $\hat{c}$ can contain state 2 in matrix H', Step 2c of function* `processMatrix` *in Figure 4.4 can test if $c, \hat{c}$ are conflicting. Furthermore, if $c, \hat{c}$ do not conflict at Step 2c, then the final matrix obtained by the algorithm will not have a conflict between $c, \hat{c}$.*

We now return to the proof of the main theorem. Step 1 requires at most $q\binom{m}{q}$ enumerations. Lemma 4.2.1 shows that given the correct Q, both κ and $GAM(t(T^*), \kappa)$ can be found in time $m^{2q}q^{O(q)} + O(nm)$. We now bound the run time for function `processMatrix`.

Lemma 4.2.7 *Total run-time for all calls to function* `processMatrix` *is $O(2^q nm^2)$.*

Proof: First notice that once a character c is removed from Δ at Step 2a of function `processMatrix`, it is never added back throughout the execution of the algorithm. This is because, function `processMatrix` resolves all the states 2 present in $(2, c)$-couples and subsequently the character is deleted (or ignored) for the rest of the algorithm in Step 2d. This bounds the number of times Step 2a is executed throughout the algorithm as $O(m)$. To prove the lemma, we now bound the time for executing Steps 2a through 2d as $O(2^q nm)$.

The number of $(2, c)$-couples is $O(n)$ and therefore Step 2b loops for $O(n)$ times. The cardinality of Q is at most q and therefore the loop in Step 2(b)i executes $O(|Q|) = O(q)$ times. The time bound for guessing the set of gametes $\mathcal{G}(c, \hat{c})$ shared between c and $\hat{c}$ is the hardest to analyze.

Consider any one call to function `processMatrix`. Let c_i's represent the characters added into Δ during the execution of a call. We show that if $\mathcal{G}(c_j, \hat{c})$ is guessed

at Step 2(b)iA then for all future c_k encountered during the execution of Step 2(b)iA, $|IND(H', \{c_k, \hat{c}\})| = 3$. When the algorithm guessed $\mathcal{G}(c_j, \hat{c})$, by definition of $(2, c_j)$-couples, $h_{2i-1}[c_j] \neq h_{2i}[c_j]$ and by the condition on Step 2(b)i, $h_{2i-1}[\hat{c}] = h_{2i}[\hat{c}] = 2$. Also $h_{2i-1}[c_k] = h_{2i}[c_k] = x$ since otherwise $|IND(H', \{c_k, \hat{c}\})| = 3$ (c_k and $\hat{c}$ cannot be identical characters). Let c_l be the character extracted from Δ and used in the loop of Step 2b during which c_k was added into Δ. This implies the existence of a couple h'_{2i-1}, h'_{2i} which is both a $(2, c_l)$-couple and a $(2, c_k)$-couple. However $h'_{2i-1}[\hat{c}] = h'_{2i}[\hat{c}] = y$. Again, otherwise $|IND(H', \{c_k, \hat{c}\})| = 3$. Therefore matrix H' induces gametes $(x, 0), (x, 1), (0, y), (1, y)$ on characters $(c_k, \hat{c})$. For all values of $x, y \in \{0, 1\}$, this results in three induced gametes. The above proof therefore shows that the `if` condition in Step 2(b)iA fails at most q times for any function call to `processMatrix`. Therefore the probability of all guesses performed at Step 2(b)iA being correct for any single call to `processMatrix` is at least 2^{-q}. Equivalently, we suffer a multiplicative factor of 2^q in the run-time because of this step.

The check performed at Step 2c takes time $O(nm)$. Assuming $q < m$, the total running time for all calls to `processMatrix` combined is $O(2^q nm^2)$.

Using Lemma 4.2.3, we know that c shares exactly three gametes with all characters $\hat{c} \in Q$ in $t(T^*)$. For character c, Step 2(b)iA (guesses) iterates over the set of all possible gametes shared between $c, \hat{c}$. Given the set of three gametes, Lemma 4.2.4 shows that there is a unique resolution of states 2 in the $(2, c)$-couples and this is performed in Step 2(b)iB. Correctness condition 1 holds by the definition of resolution. Step 2c of function `processMatrix` checks for conditions 2 and 3. Using Corollary 4.2.6, we know that c and $\hat{c}$ cannot conflict because of the resolution of any of the remaining 2s (ensures correctness condition 3). Finally, there can be no 2s in character c (since $c \notin Q$) or in any of the $(2, c)$-couples since it was just resolved. Step 8 of function `solveIPPH` iterates n times. At each iteration, it performs a brute-force step of computing all possible ways of resolving the 2s. Since only the characters in Q can contain state 2 at this point, Step 8a takes $O(m2^q)$ time. This step also checks if the resulting gametes on characters in κ are in the predicted set $GAM(t(T^*), \kappa)$(ensures correctness condition 3). This shows that any matrix found by the algorithm obeys the correctness conditions and the running time is $nm^{O(q)}$. Finally, we know that there exists a set of three gametes $\mathcal{G}(c, Q)$ (as defined by $t(T^*)$) s.t. resolving based on $\mathcal{G}$ will ensure that $c, \hat{c}$ do not conflict and $GAM(H', \kappa) \subseteq GAM(t(T^*), \kappa)$. Using these two observations, we know that if $\texttt{penalty}(T^*) \leq q$, then the algorithm finds matrix H' that obeys the correctness conditions in the stated time.

We now prove the correctness of our algorithm:

Theorem 4.2.8 *Any solution matrix H obeying all the correctness conditions is optimal.*

Proof: The proof is constructive and demonstrates the procedure to construct a q-near-perfect phylogeny for H. The phylogeny along with correctness condition 1 guarantees that the returned matrix H is an optimal solution.

Matrix H satisfies the following two properties: if a pair of characters c, c' conflict in H, then $c, c' \in \kappa$ (third correctness condition); a q-near-perfect phylogeny can be constructed on $GAM(H, \kappa)$ (since $GAM(H, \kappa) \subseteq GAM(t(T^*), \kappa)$, second correctness condition). It can be shown that a q-near-perfect phylogeny can be constructed for any matrix that satisfies the above two properties (see Section 7 of Gusfield and Bansal [62]). Such a phylogeny is

obtained by constructing the q-near-perfect phylogeny on on $GAM(H, \kappa)$, contracting the phylogeny to a vertex and constructing a perfect phylogeny on the remaining characters.

Theorem 4.2.9 *The algorithm of Figure 4.4 returns an optimal solution H to the IPPH problem in time $nm^{O(q)}$.*

Proof: The proof follows directly from Theorems 4.2.2 and 4.2.8.

4.3 Solutions to PPH

We assumed in the preceding sections, following prior work [104], that the perfect phylogeny stage of the algorithm will find a unique solution, but this assumption does not necessarily hold. To guarantee optimality, the algorithm would need to enumerate over all solutions to the PPH sub-problem, increasing run-time proportionally. Prior work showed that the number of PPH solutions is at most 2^k, where k is the number of characters of G that do not contain the homozygous minor allele [28, 59]. In the worst case, this could be as large as m and therefore the number of PPH solutions can be 2^m. This however should not occur in practice since the underlying population typically follows a random mating model.

Hardy-Weinberg equilibrium states that the two haplotypes of any given individual are selected independently of one another at random. For any fixed character, let p be the minor allele frequency and $(1-p)$ be the major allele frequency. Consider G, an $n \times m$ input genotype matrix to the PPH problem. The probability under Hardy-Weinberg that none of the n taxa contain the homozygous minor allele is $(1-p^2)^n$. This probability could be large for very small values of p.

It is reasonable to assume that the value of $p > c$ for some constant c since otherwise SNPs cannot be detected. In this case, with high probability in n (at least $1-(1-c^2)^n$), a specific character contains the homozygous minor allele. Therefore in expectation the number of characters without a homozygous minor allele is at most $m(1-c^2)^n$. This expectation, exponentially tends to zero with n.

We now consider a more general setting when the value of p is assumed to be uniformly distributed in $[0, 0.5]$. Now, the probability that a specific character does not contain the homozygous minor allele is:

$$2\int_0^{0.5}(1-p^2)^n dp$$

$$= 2\int_0^{1/\sqrt{n}}(1-p^2)^n dp + 2\int_{1/\sqrt{n}}^{0.5}(1-p^2)^n dp$$

$$\leq \frac{2}{\sqrt{n}} + 2\int_{1/\sqrt{n}}^{0.5}(1-p^2)^n dp$$

$$\leq \frac{2}{\sqrt{n}} + 2\sum_{i=1}^{\infty}\int_{i/\sqrt{n}}^{(i+1)/\sqrt{n}}(1-p^2)^n dp$$

			Error rate			Total Run Time(secs)		
Perf	#Blocks	#SNPs	PHASE	htyper	our alg	PHASE	htyper	our alg
0	3497	20816	0.11	0.11	0.17	3521.33	337.18	17.21
1	461	4211	0.53	0.47	0.35	805.62	80.62	8.77
2	93	1266	0.83	0.68	0.55	268.02	59.28	1111.18

Table 4.1: Empirical results on Chromosome 21. Blocks with 0, 1, 2 near-perfectness (accounting for 4051 out of 4135 blocks) were analyzed separately using the three algorithms. Error rate is the number of switch errors divided by the number of blocks. Total run time is the sum over all blocks.

$$\leq \frac{2}{\sqrt{n}} + \frac{2}{\sqrt{n}} \sum_{i=1}^{\infty} (1 - \frac{i^2}{n})^n$$

$$\leq \frac{2}{\sqrt{n}} + \frac{2}{\sqrt{n}} \sum_{i=1}^{\infty} e^{-i^2} \leq \frac{4}{\sqrt{n}}$$

Now, if $n = \Omega(m^2)$, then using Chernoff bounds we can show that with high probability the number of characters that lack a homozygous minor allele is bounded by $2 \log m$ and therefore the number of solutions to the PPH problem is bounded by m^2.

This discussion answers the question raised by Gusfield on the theoretical estimate for the number of PPH solutions to expect from a coalescent model [59]. Furthermore, since PPH is always performed on SNP blocks with low diversity, it is not unreasonable to assume $n >> m$. Since the number of solutions returned by PPH is $O(m^2)$, the IPPH algorithm described above with high probability just suffers $O(m^2)$ overhead for finding the optimal extension of each PPH solution.

4.4 Empirical Validation

We demonstrate and validate our algorithm by comparing with leading phasing methods using a collection of large-scale genotype data of known phase from a high resolution scan of human chromosome 21 [42]. The study identified common single nucleotide polymorphisms (SNPs) and typed them on 20 sequences through a method that directly identifies haplotypes rather than genotypes. The SNPs were partitioned into 4135 haplotype blocks by the authors of that study using a diversity test. We extracted haplotypes from the 'haplotype pattern' file provided in the supplementary material, which identifies distinct haplotypes in each block and provides some inference of missing data when it can be done unambiguously. We ignored haplotypes which still had significant missing data, replacing them with haplotypes randomly selected from the multinomial distribution of fully-resolved haplotypes in the corresponding block in order to maintain 20 chromosomes per block. The haplotype blocks were divided into four sets based on their imperfectness($q = 0, 1, 2, 3+$) using a prior method [106]. A large majority of blocks (98%) are 0, 1, or 2 near-perfect. We do not solve optimally for the 2% with imperfectness 3 or greater because the run-time

for optimal solutions would be prohibitive. Such blocks can be solved non-optimally in practice by subdividing into smaller blocks of lower imperfectness, but such data would not be comparable to those solved optimally and is therefore omitted from our analysis. We then randomly paired haplotypes to produce 10 genotypes per block as our final test set. Figure 4.3 shows that the length of the blocks is related to q, the imperfectness. The tails are heavier for larger values of q as expected. For instance, this shows that a significant fraction of the $q = 2$ blocks have large number of characters when compared with $q = 1, 0$.

We compared our method with two popular phasing packages. We ran the `haplotyper` [87] package, which uses an expectation-maximization heuristic, with 20 rounds (the recommended value). We also ran the `PHASE` [110] package, which employs a Markov Chain Monte Carlo method. Although `PHASE` can make use of additional information such as SNP positions, we provided it only the genotypes as input.

For our own method, with any given q, we enumerated all possible phylogenies that are at most q-near-perfect. Note that this does not guarantee finding all possible output matrices that are consistent with a q-near perfect phylogeny. Where multiple solutions were obtained, we selected the one that minimized the entropy of the haplotype frequencies. For the PPH sub-problem we implemented a fast prior algorithm [28]. We note that for simplicity, we do not implement the algorithm exactly as described. Rather, function `processMatrix` does not add characters into Δ to be processed iteratively. The implementation however always returns a haplotype output matrix and we report the switch errors [104] for the output.

Table of Figure 4.1 summarizes the test results[2]. All methods provide comparable accuracy when blocks are perfect. We attribute the slightly worse performance of our method on perfect input to the fact that we do not use any procedure to select a maximum-likelihood output matrix among all perfect matrices, as is done for prior perfect phylogeny methods [28]. All three methods degrade in accuracy with increase in imperfectness. Our method, however, scales much better with imperfectness than do the other two, clearly outperforming them on 1-near-perfect and 2-near-perfect inputs. Imperfectness can result from recurrent mutations, recombinations or incorrect SNP inferences and we attribute our method's superior performance on imperfect data to the fact that it explicitly handles one of these factors while the others suffer from all three. Our method is extremely fast for $q = 0$, where it reduces to the perfect phylogeny algorithm of Eskin et. al. [28] and also substantially outperforms the competing methods for $q = 1$. Our method's run time rapidly increases with larger q, though, as expected from the theoretical bounds.

4.5 Discussion and Conclusions

We have developed a theory for reconstructing phylogenetic trees directly from genotypes that is optimal in the number of mutations. As an immediate application, we solve the general IPPH problem. We demonstrate practical results that show great promise in accurately inferring phase from real data sets. Our results suggest that imperfect phylogeny approaches can lead to significant improvements in accuracy over other leading phasing

[2] `Haplotyper` crashes on 18 blocks which were ignored in the calculation of its accuracy.

methods. Run time, while very fast for perfect and almost perfect data, remains an obstacle for even modest q; this observation suggests a need for further research in improving theoretical and practical bounds for general q. Our new method has several immediate applications in computational genetics:

- Phasing: At present, the method is competitive with the most widely used tools in accuracy and, with optimizations, should become competitive in run-time for larger q. Both `PHASE` and `haplotyper` return confidence scores on its results, which might allow a slower, high-accuracy method such as ours to function as a fall-back for regions that those methods cannot infer with confidence.
- Phylogeny Construction: Our run times are competitive with typical times to be expected from other optimal phylogeny reconstruction algorithms, even when the input consists of haplotypes. Our approach may thus be considered preferable to the standard practice of inferring haplotypes then fitting them to phylogenies.
- Haplotype Blocks Inference: Our method could serve as an improved means of identifying recombination-free haplotype blocks for purposes of association study design by more accurately distinguishing recurrent mutation from recombination. The blocks might thus be useful in improving statistical power in haplotype-based association testing.

Future empirical studies, enabled by our method, will be needed to better establish the nature of imperfectness in real genotype data and the degree to which better handling of recurrent mutations will be of use in practice.

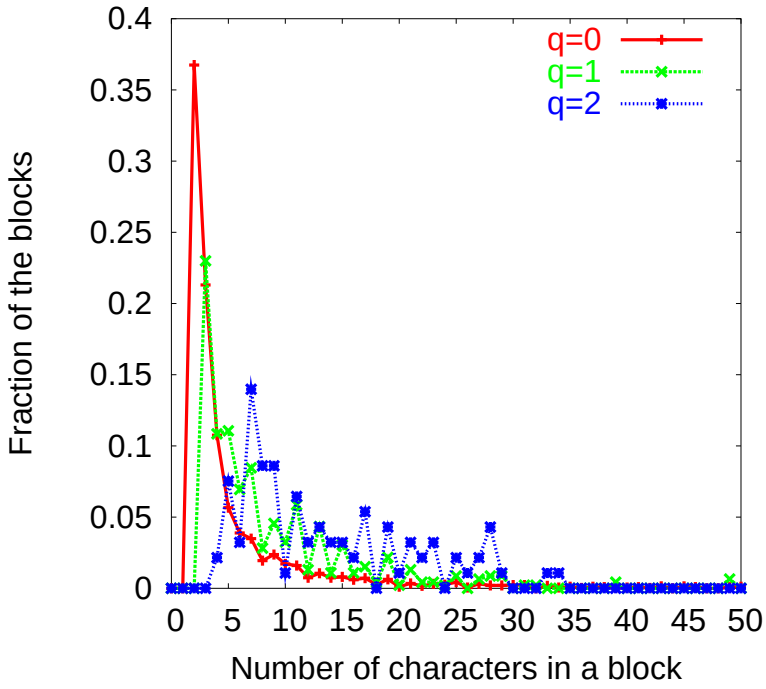


Figure 4.3: Distribution of the fraction of blocks as a function of the number of characters in the block. Data for more than 50 characters not shown.

function `solveIPPH(input matrix` G`)`

1. guess $Q = \{i \in C(G) | \exists e_1, e_2 \in E(T^*), e_1 \neq e_2, \mu(e_1) = \mu(e_2) = i\}$
2. let M be matrix G restricted to characters $C(G) \setminus Q$
3. let M' be the unique solution to perfect phylogeny haplotyping of M
4. let H' be matrix M' defined on characters $C(G)$ s.t. $\forall h'_{2i-1}, h'_{2i} \in R(H')$ and corresponding taxa $g_i \in R(G).\forall \hat{c} \in Q.h'_{2i-1}[\hat{c}] = h'_{2i}[\hat{c}] = g_i[\hat{c}]$
5. guess $\kappa = \{i \in C(G) \mid \exists j \in C(G),\ |GAM(t(T^*), \{i, j\})| = 4\}$
6. guess $GAM(t(T^*), \kappa)$
7. loop for $c \in C(H') \setminus \kappa$
   (a) $H' :=$ `processMatrix`(H', c)
8. for all couples h'_{2i-1}, h'_{2i} in H'
   (a) resolve state 2 on h'_{2i-1}, h'_{2i} s.t.$GAM(\{h'_{2i-1}, h'_{2i}\}, \kappa) \subseteq GAM(t(T^*), \kappa)$

function `processMatrix(matrix` H'`, character` c`)`

1. initialize the set $\Delta := \{c\}$
2. while $|\Delta| > 0$ do
   (a) extract a character c from Δ
   (b) for all $(2, c)$-couples h_{2i-1}, h_{2i}
      i. for all $\hat{c} \in Q$ with $h_{2i-1}[\hat{c}] = 2 (= h_{2i}[\hat{c}])$
         A. if $|IND(H', \{c, \hat{c}\})| = 3$ then $\mathcal{G}(c, \hat{c}) = IND(H', \{c, \hat{c}\})$
            else guess three gametes $\mathcal{G}(c, \hat{c})$
         B. resolve state 2 in $\hat{c}$ on h_{2i-1}, h_{2i} based on $\mathcal{G}(c, \hat{c})$
         C. $\Delta := \Delta \cup \{c' \notin \kappa | h_{2i-1}[c'] \neq h_{2i}[c']\}$
            i.e. add to Δ characters $c' \notin \kappa$ for which the current couple is also a $(2, c')$-couple
   (c) if $\exists \hat{c} \in Q$. s.t.$(c, \hat{c})$ conflict or
   for the set $H2$ of all $(2, c)$-couples $GAM(H2, \kappa) \nsubseteq GAM(t(T^*), \kappa)$ then return `no solutions`
   (d) remove c from $C(H')$
3. return H'

Figure 4.4: Algorithm to solve IPPH

Chapter 5

Maximum Parsimony Phylogeny Reconstruction from Genotypes

In this Chapter, we continue to develop algorithms for reconstructing maximum parsimony (MP) phylogenetic trees from genotypes. While the previous chapter focused on theoretical run-time issues and phasing accuracy in terms of the popular switch-error statistic, this chapter will attempt to simply construct the MP phylogeny from genotypes and compare the resulting trees to those produced using other methods.

As has been stated before, inferring maximum parsimony Steiner trees on binary SNP data (haplotypes) is an NP-hard problem [53], there are excellent methods now available for solving it in practice, including fast heuristics suitable for difficult instances [5, 10, 50, 55], fixed parameter tractable methods for provably efficient optimal solutions in some cases [9, 107], and integer linear programming methods for provably optimal solutions of many harder cases [108]. Several of these methods were explained in detail in the previous Chapters.

Unfortunately, the haplotype input data these methods assume, also known as "phased" data, are not easily available for autosomal genetic regions. Large-scale genetic studies usually instead must gather unphased, or genotype, data, in which haplotype contributions from two homologous chromosomes are conflated with one another.

To illustrate the problem, it will be helpful to arbitrarily denote the minor allele at each SNP site by 1 and the major allele by 0. In a genotype data set, we only observe the number of minor alleles present at each SNP site, which we will denote by 0 for homozygous major, 1 for heterozygous and 2 for homozygous minor. For example, see Figure 5.1. Hence, if we examine m sites, then a genotype sequence is a string of the form $\{0, 1, 2\}^m$ while a haplotype sequence is a string of the form $\{0, 1\}^m$. A pair of haplotype sequences is *consistent* with (explains) a genotype sequence when they have the same allele counts at all sites. In the $\{0, 1, 2\}$ notation above, a pair of haplotypes is consistent with a genotype when the sum of the two haplotype vectors produces the genotype vector.

While mitochondrial and Y chromosome data can serve for tracking population histories on broad scales [117], autosomal phylogenies are still independently valuable for practical applications in association study design, marker selection, and the identification of specific variant sites that are unusually mutable, repeatedly altered by gene conversion,

	True sequences	Binary format	Observed (Input to phasing)	Phased (Output of phasing) Switch errors: 1 Phylogenetic Errors: 3
Ind 1:	ACGTAA ACGGGT	000000 000111	000111	000000 000111
Ind 2:	CGTTAA ACGGGT	111000 000111	111111	000000 111111
Ind 3:	CGTTAA CGTTAA	111000 111000	222000	111000 111000

Figure 5.1: Phasing: computationally inferring genotypes from haplotypes Although DNA sequences consist of four bases, single nucleotide polymorphisms (SNPs) are biallelic. Therefore, the sequence variation can be expressed using binary symbols. The observed genotype sequences consist of conflated combinations of two true haplotype sequences. Programs that computationally infer haplotypes attempt to minimize switch errors.

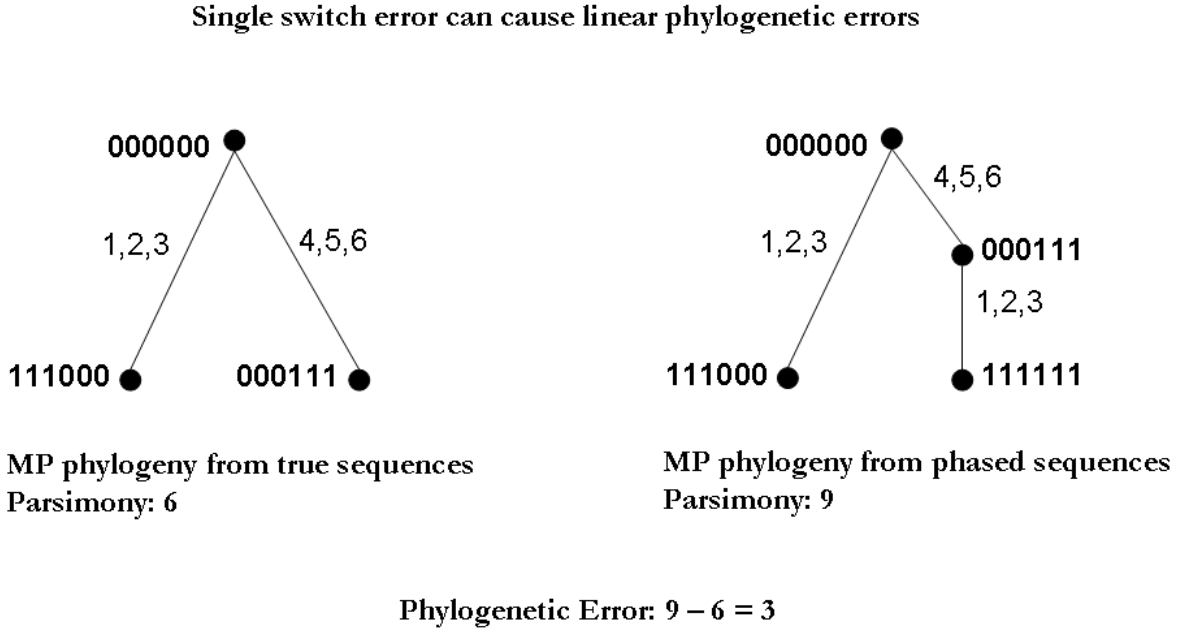


Figure 5.2: Although switch errors in phase inference can be small, in this case 1, the phylogeny size could be significantly altered. Therefore estimates such as mutation rates could be significantly affected if performed on computationally inferred haplotypes as opposed to genotypes. Moreover, in current methods it is impossible to say if the inferred phylogeny size is larger or smaller than that of the phylogeny from the true haplotypes.

or under selective pressure to recurrently mutate. Phylogeny inference cannot generally be performed directly on genotype data and in practice one must therefore analyze autosomal data by first computationally phasing the data to predict the haplotypes [18]. Many methods are now available for this phasing problem, such as `PHASE` [110], `fastPHASE` [98], `HAP` [65] and `PPH` [59]. This phasing step, however, can produce erroneous assignments and the maximum parsimony phylogeny on the computationally phased genotypes need not be the same as, or even close to, the maximally parsimonious tree consistent with the original unphased genotypes. Phasing programs are typically designed to minimize the "switch error," in which the contributions from two homologous chromosomes are swapped between two consecutive markers (see [98] for the formal definition). Yet a single switch error in a phasing assignment can introduce a large number of errors (linear in the number of markers) in the resulting phylogeny assignment, as shown in Figures 5.1 and 5.2. Even high-quality phasing methods can thus produce poor-quality phylogenies.

A limited amount of prior work has examined the prospect of inferring maximum parsimony phylogenies directly from genotype data. Notice that in such problems, we wish to determine a pair of haplotypes for each input genotype sequence such that the maximum parsimony phylogeny size on the set of haplotypes is minimized. Gusfield showed that the problem can be efficiently solved when the genotype data are consistent with a perfect phylogeny [59], an evolutionary tree in which each variant site mutates only once. Several subsequent algorithms were developed for the same problem that were either simpler or faster asymptotically [3, 4, 25, 28, 59]. While the perfect phylogeny assumption is restrictive, this variant does have practical importance as a technique for fast phasing (e.g., [65]). The perfect phylogeny assumption will not be true in general, however. In particular, it will not allow analysis of data containing recurrently mutating sites, the detection of which is an important reason for studying phylogenetics of autosomal DNA. Halperin et al. [65] generalized Gusfield's perfect phylogeny method heuristically to allow limited solution of phylogenies deviating slightly from the assumption of perfection. These are called *near-perfect phylogenies* [51] and specifically *q-near-perfect* (or *q-imperfect*) when q additional mutations are needed beyond those required to produce a perfect phylogeny. Song et al. [104] and Sridhar et al. [105] developed rigorous methods for efficiently finding maximum parsimony near-perfect phylogenies, but these methods proved practical only for small q (at most 2). In practice, the problem of finding maximum parsimony phylogenies from genotype data has remained intractable for all but the simplest data sets.

We note that the parsimony based approach described above is different from finding haplotypes corresponding to the given genotypes based on 'pure parsimony,' an objective that minimizes the number of distinct haplotypes needed to explain the observed genotypes as opposed to minimizing the number of mutations. The pure parsimony problem is NP-complete as well and there are integer program based approaches that solve problem instances of reasonable size [60]. Pure parsimony and maximum parsimony phylogenetic trees share some properties that we can exploit in our method. The solution to the pure parsimony problem provides a lower bound on the size of the maximum parsimony phylogeny, as no phylogeny can have fewer mutations than one less than the minimum number of haplotypes needed to explain the genotypes. Furthermore, the solution of the pure parsimony problem also provides a good starting set of haplotypes from which we can obtain

an upper-bound for the size of the maximum parsimony phylogeny.
In this paper, we provide the first general, practical methods for maximum parsimony phylogeny inference from genotypes and use these methods to assess the inaccuracies introduced by phasing genotypes prior to phylogeny inference. Our algorithm relies on solving integer linear programs and allows for efficient solution of moderate-sized problem instances but large imperfection. As an immediate application, our method can be used to infer the minimum number of recurrent mutations required to explain the given set of genotypes. We apply the resulting methods to a selection of real and simulated data, where we compare the true imperfection, imperfection from haplotypes computationally inferred from genotypes and imperfection directly obtained from genotypes. This analysis shows that the phasing step often increases inferred phylogeny size, overestimating the true maximum parsimony. Motivated by our observations, we introduce a new *phylogenetic error* statistic that is better suited for assessing phase accuracy for phylogenetic applications than the standard switch error statistic [98].

5.1 Algorithms and Data Analysis

We implemented two versions of the integer linear program both of which were competitive in practice. The first is a direct integer linear program implementation and the second is a branch-and-bound algorithm that wraps over a second integer linear program. We describe the direct implementation first followed by the branch-and-bound method. Both methods were implemented in C++ using the Concert Technology of CPLEX 10.0 for integer linear program (ILP) solutions. We found the branch-and-bound method to give generally lower run times in practice than the direct ILP method. We therefore used the branch-and-bound method exclusively in generating the empirical results presented here.

5.1.1 Direct Integer Linear Programming Approach

This section introduces our ILP algorithm to solve the Genotype MP Phylogeny Problem. In the first subsection, we introduce pre-processing techniques that typically reduce the problem size after which we describe the ILP.

Preprocessing

Preprocessing techniques form an integral part of any solution method based on integer programming. We now describe the major preprocessing methods used.

Let G be the $n \times m$ input genotype matrix. Without loss of generality we can assume that all m sites are varying. We can also remove redundant rows (genotypes) of G until all rows are distinct, since this does not change the length of the optimal phylogeny. We now describe a method to remove redundant sites (columns) from G. Note that we are free to exchange labels 0 and 2 (homozygous major and minor alleles) independently at each site without change in the size of the phylogeny. Therefore two sites i and j are considered redundant if they are identical or become identical after relabeling one site.

For all sites k, let weight w_k be initialized to 1. We then iteratively perform the following operation: for any pair of redundant sites i, j, set $w_i := w_i + w_j$, and remove site j from the matrix. Let G' be the final matrix after this sequence of preprocessing steps. We now redefine the length of a phylogeny using a weighted Hamming distance as follows.

Definition 19 *The* length *of phylogeny* $T(V, E)$ *is* $\texttt{length}(T) = \sum_{(u,v)\in E} \sum_{i\in D(u,v)} w_i$, *where* $D(u, v)$ *is the set of sites where* u, v *differ.*

The following lemma justifies the preprocessing step:

Lemma 5.1.1 *The transformation from genotype matrix* G *to weighted genotype matrix* G' *does not change the length of the most parsimonious phylogeny.*

Proof: For any genotype matrix I, let T_I denote the optimal phylogeny on I. For a site i of I, let j be a redundant site and consider the matrix $I \cup \{j\}$. The topology of phylogeny T_I also gives a phylogeny for $I \cup \{j\}$, obtained by mutating j wherever i mutates. The length of $T_{I\cup\{j\}}$ is $\texttt{length}(T_{I\cup\{j\}}) = \texttt{length}(T_I) + \mu(i)$, where $\mu(i)$ is the number of times site i mutates in T_I.

Now, assume that the most parsimonious phylogeny T_G for G resolves redundant genotype sites i and j differently, i.e., there is a haplotype for which sites i and j differ. Without loss of generality, suppose $\mu(i) \leq \mu(j)$ in T_G. Then removing column j from T_G results in a phylogeny $T_{G\setminus\{j\}}$ with $\texttt{length}(T_{G\setminus\{j\}}) = \texttt{length}(T_G) - \mu(j)$. Now, since j is identical to i, the argument above implies that adding site j back to the phylogeny gives a tree with length $\texttt{length}(T_G) - \mu(j) + \mu(i) \leq \texttt{length}(T_G)$. Therefore, there is an optimal phylogeny resolving sites i and j identically.

Due to these preprocessing steps, we assume from now on that the input genotype matrix G has distinct rows, distinct sites, and weights $w_i \geq 1$ associated to sites. For each genotype $g \in G$, we create the set $R(g)$ consisting of all possible pairs of haplotypes explaining g. Note that if p is the number of heterozygous sites in g, then $R(g)$ consists of 2^{p-1} pairs of haplotypes.

Now, consider the matrix $H = \cup_{g\in G} R(g)$, where the rows are all possible haplotypes that can explain input genotypes in G. H is a binary matrix, and for such instances, structural properties of the optimal phylogeny can be captured by a graph known as the *Buneman graph* $\mathcal{B}(H)$ [13]. We will explain the generalization of this graph due to Barthélemy [6].

For a binary input matrix H and a site c of H, the *split* $c(0)|c(1)$ defined by c is a partition of the haplotypes into two sets, where $c(0)$ is the set of haplotypes with value 0 in site c and $c(1)$ is the set of haplotypes with value 1 in site c. Each of $c(0)$ and $c(1)$ is called a *block* of c. Each vertex of the Buneman graph is an m-tuple of blocks $[c_1(i_1), c_2(i_2), \ldots c_m(i_m)]$ ($i_j = 0$ or 1 for each $1 \leq j \leq m$), with one block for each site and such that each pair of blocks $c_j(i_j) \cap c_k(i_k)$ has nonempty intersection. There is an edge between two vertices in $\mathcal{B}(H)$ if and only if they differ in exactly one block. Notice that vertices in the Buneman graph can be viewed simply as haplotypes. An m-tuple $[c_1(i_1), \ldots, c_m(i_m)]$ translates to haplotype $(i_1, \ldots, i_m)$. Buneman graphs are very useful due to the following theorem:

Theorem 5.1.2 *[5, 100] Let* $\mathcal{B}(H)$ *be the Buneman graph for binary (haplotype) matrix* H. *Every optimal phylogeny* T^*_H *is a subgraph of* $\mathcal{B}(H)$.

Using Theorem 5.1.2, we first construct the Buneman graph on H and then solve the phylogeny problem on underlying graph $\mathcal{B}(H)$. The following lemma gives a bound on the time required to construct $\mathcal{B}(H)$.

Lemma 5.1.3 *[107] The Buneman graph $\mathcal{B}(H)$ for input H on m sites can be constructed in time $O(km)$ where k is the number of vertices in $\mathcal{B}(H)$.*

The Buneman graph is simply a method to reduce the size of the underlying graph from an m-cube with 2^m vertices to a (typically significantly) smaller sub-graph. Putting together these methods, we can summarize our preprocessing steps as follows:

1. Create a weighted genotype matrix G where sites are pair-wise distinct.
2. Create a set H of all possible haplotypes explaining rows of G.
3. Construct the underlying graph $F(V, E) = \mathcal{B}(H)$ where $H \subseteq V$ and $(u, v) \in E$ connects two vertices (haplotypes) if and only if they differ in exactly one site. Edge weights $w_{u,v} = w_i$ where i is the site at which u and v differ.

We apply some additional heuristic preprocessing steps that have proven very effective in practice. One of these steps identifies a subset of haplotypes that must occur in any optimal solution and then removes from the input any genotypes that can be produced from pairs of these obligatory haplotypes. As any optimal output can produce these genotypes, their absence will not change the final output. We can also eliminate certain possible haplotypes because they would imply high-weight edges and therefore cannot occur in any low-cost solution.

Once all preprocessing steps have been applied, we have a weighted Buneman graph $B(H)$ that contains every node and edge that might be included in any optimal phylogeny for G. We now show an ILP formulation to simultaneously select the optimal subset $H' \subseteq H$ such that all of G can be derived from H' and connect H' using a tree.

ILP Formulation

We now develop an ILP formulation for the problem based on multicommodity flows [118]. The formulation borrows from prior work on fast ILP solution of maximum parsimony phylogenies on haplotypes [107]. Although this formulation can use exponential numbers of variables and constraints in the worst case, it is fast in practice. It is possible to solve the maximum parsimony genotype problem using an ILP with polynomial numbers of variables and constraints, but all polynomial-size variants that we developed proved intractable in practice.

The high-level idea of the method is to send flow from a designated root to each haplotype that is used to explain an input genotype. Each of these haplotypes acts as a sink for one unit of flow. The program must select a subset of edges that accommodate all flow while minimizing the cost of the edges selected. This flow formulation guarantees that every haplotype is connected to the root and the minimization prevents formation of cycles. The formulation thus forces the output to be a tree. For the sake of simplicity, we assume that the all-zeros haplotype is present in all the solutions. We can treat this as the *root*.

Let h_k be an indicator variable denoting the presence or absence of haplotype $k \in H$. If $h_k = 1$, k is called a *present* haplotype. We have binary variables $p_{i,j}$ that denote the presence of both haplotypes h_i and h_j. All the present haplotypes act as a sink for one unit of flow from the root vertex. On the other hand, all non-present haplotype vertices and Steiner vertices satisfy perfect flow conservation. To enforce this, we use two types of binary variables $f^k_{i,j}$ and $s_{i,j}$ for each edge $(i,j) \in E$. The variables $f^k_{i,j}$ are real valued and represent the amount of flow along edge (i,j) whose final destination is haplotype k. Note that if k is a non-present haplotype, then $f^k_{i,j}$ should be set to 0 for all edges (i,j). Variables $s_{i,j}$ are binary variables that denotes if edge (i,j) of the graph has been selected. We want to enforce that flow can be sent along edge (i,j) only if it has been selected.

We now have the following integer linear program:

$$\begin{aligned}
\min \quad & \textstyle\sum_{i,j} w_{i,j} s_{i,j} && && (5.1)\\
\text{s.t.} \quad & p_{ij} \leq h_i && \forall i,j \in H && (5.2)\\
& p_{ij} \leq h_j && \forall i,j \in H && (5.3)\\
& \textstyle\sum_{(i,j)\in R(g)} p_{ij} \geq 1 && \forall \text{input } g \in G && (5.4)\\
& \textstyle\sum_j f^k_{0,j} = \sum_j f^k_{j,k} = h_k && \forall k \in H && (5.5)\\
& \textstyle\sum_j f^k_{i,j} = \sum_j f^k_{j,i} && \forall i \neq 0,k,\ k \in H && (5.6)\\
& f^k_{i,j} \leq s_{i,j} && \forall i,j,k && (5.7)
\end{aligned}$$

In constraints (5.2) and (5.3), variable p_{ij} indicates the presence of the haplotype pair (h_i, h_j). Constraint (5.4) guarantees that each genotype is explained by at least one pair of haplotypes. Constraint (5.5) imposes inflow/outflow constraints on haplotypes as well as enforcing the condition that there is positive flow to a haplotype h_k only if h_k is selected. Constraint (5.6) imposes flow conservation at all non-present haplotype vertices as well as Steiner vertices and constraint (5.7) imposes the condition that flow can only be sent along edges present in the solution. Note that all integer variables of the above linear program are binary. Finally, we observe that the solution of the ILP is the size of the most parsimonious phylogeny on G.

5.1.2 Branch and Bound Algorithm

We developed an alternative method for the problem that uses a simpler integer linear program embedded in a branch-and-bound routine. The high-level idea behind the method is to first guess the set of haplotypes that would phase the given input genotypes and then construct a most parsimonious phylogeny on the haplotypes. Note that all the pre-processing techniques outlined in the previous sub-section still apply for this method.

We use G to refer to the input set of genotypes. For a given set of haplotypes $\mathcal{H}$, we can construct the most parsimonious phylogeny $T_{\mathcal{H}}$ using the algorithm described by Sridhar et al. [106]. We will use `hapMP` to denote this algorithm, which will take a set of haplotypes and return the size of the most parsimonious phylogeny. The pseudo-code for the branch and bound method is shown in Figure 5.3.

```
function genBB(genotypes G, haplotypes H, integer u)
  1. for all row vectors g⃗ ∈ G
     (a) if ∃h1, h2 ∈ H s.t. h1⃗ + h2⃗ = g⃗ then G ← G \ {g}
  2. if(|G| = ∅) then return hapMP(H)
  3. if (hapMP(H) ≥ u − 1) then return ∞
  4. let g⃗ be a row vector of G
  5. for all h1⃗, h2⃗ s.t. h1⃗ + h2⃗ = g⃗
     (a) G′ ← G \ {g⃗}
     (b) H′ ← H ∪ {h1, h2}
     (c) b ← genBB(G′, H′, u)
     (d) if b < u then u ← b
  6. return u
```

Figure 5.3: Pseudo-code for the branch-and-bound algorithm for phylogeny reconstruction from genotypes

The *branch step* is performed by Step 5, where the algorithm attempts to phase genotype g using all possible pairs of haplotypes h_1, h_2. Integer u of the above pseudo-code refers to the current best upper-bound. The *bound step* is performed by Step 3 which just reconstructs a phylogeny over the current set $\mathcal{H}$ of haplotypes. Step 1a ensures that at least one more haplotype $h \notin \mathcal{H}$ is required to obtain the final set of haplotypes. Therefore, even if $\texttt{hapMP}(\mathcal{H}) = u - 1$, this branch cannot yield a solution of smaller cost than current upper-bound u.

In the above method, the height of the branch-and-bound tree is at most n, the number of input genotypes. The branching factor at each internal node is at most 2^k where k is the number of heterozygous sites on the genotype g. This is always bounded by 2^m. Although the running-time of the final branch-and-bound method is super-exponential, we find that its run time is competitive with and often superior to the ILP described in the previous section.

5.1.3 Data Generation and Analysis

In order to generate simulated data, coalescent trees were created using Hudson's `ms` program [72]. The only parameter required to generate tree topologies is the number of haploid chromosomes n_h. The `ms` program can also use this tree to produce haplotype sequences, but does so under the infinite-sites model (without any recurrent mutations). We therefore instead used the `seq-gen` program of Rambaut and Grassly [93] to generate n_h haplotypes using the `ms` coalescent tree. We varied the number of SNPs m and the mutation rate parameter $\theta = 4N_0\mu$, where μ is the probability of mutation of any of the simulated SNPs in one generation and N_0 is the effective population size. We relate the simulation parameter μ to the per-site mutation rate by assuming an effective population size $N_0 = 10,000$ (a

reasonable estimate for humans [94]). For instance, for 5 sites, we obtain a per-site mutation probability of 10^{-6} for $\theta = 0.2$. `seq-gen` was used under the `GTR` model, a generic time reversible Markov model. Mutation rates between A and C and between G and T were defined using the same parameter θ. For all the other four pairs we set the mutation rate to be 0 in order to produce biallelic data. The exact command line used to execute `seq-gen` for a given mutation rate parameter θ and SNP number m was the following:

`seq-gen -mGTR -r` θ`,0,0,0,0,`θ `-l` m

Each data point was generated from 200 independently generated simulated data sets, with the reported error rates summed over the 200 replicates. In our first set of simulations, designed to test the effect of mutation rate on accuracy, we varied θ over the range 0.2-0.6 in increments of 0.05 for windows of 5 and 10 SNPs and for sample sizes of 30 and 60 input haplotypes. Our second set of experiments, designed to test the effect of sample size on accuracy, fixed θ at 0.5 and varied the number of haplotypes from 30 to 120 in increments of 10 for windows of 5 and 10 SNPs. Data points plotted represent summed errors over the 200 replicates per parameter value.

Mitochondrial data was extracted from of a set 63 complete mitochondrial DNA sequences of 16,569 bases each produced by from Fraumene et al. [54]. We produced artificial diploids from the data by randomly selecting 60 of the sequences and randomly grouping them into 30 pairs. We computationally inferred haplotypes from all of the genotypes using `fastPHASE` and we constructed phylogenies for all sliding windows of 50 bases across the data set by each of three methods: maximum parsimony using true haplotypes, inferred haplotypes and from the genotypes.

Autosomal DNA was extracted from a lipoprotein lipase (LPL) data set due to Nickerson et al. [19]. Because the pairs of haplotypes into genotypes were not published, we duplicated the first haplotype to obtain 78 distinct sequences and then randomly paired them to produce 39 artificial genotypes from the true haplotypes. As in the previous case, we ran `fastPHASE` and `haplotyper` on all of the SNPs put together to obtain inferred haplotypes. In order to reduce the possibility of recombination events confounding our results, we used the `HAP` webserver [65] to break the 86 SNPs into blocks. HAP was also used to infer missing data. We then evaluated phylogeny sizes by our direct method, from the true haplotypes, and from the inferred haplotypes for each block.

For details on web interface for the online tool, please see the Appendix.

5.2 Results and Discussion

The algorithms to solve the MP phylogeny problem are discussed in Section 5.1. In this Section, we present the results of a series of empirical tests to assess the utility of the method on real and simulated genetic data. With both kinds of data, we begin with known haplotypes and then artificially pair them to produce genotypes. For each problem instance, we reconstruct maximum parsimony (MP) phylogenies in three ways: directly from the genotypes using the algorithm presented in this paper, from the original (true) haplotypes and from haplotypes computationally inferred from the genotypes using `fastPHASE` [98] and `haplotyper` [87], two leading methods for haplotype inference. We use the notation T_{min},

T_{true}, and T_{phase} to denote the MP phylogeny from the genotypes, true haplotypes and inferred haplotypes (either using `fastPHASE` or `haplotyper`) respectively. We further denote the parsimony score (number of mutations) of a phylogeny T by $\texttt{length}(T)$. For phylogeny T that is either T_{phase} or T_{min}, we define a *phylogenetic error* based on $\texttt{length}(T_{true})$ as follows.

Definition 20 *The* phylogenetic error *of Phylogeny T (T_{min} or T_{phase}) is* $|\texttt{length}(T_{true}) - \texttt{length}(T)|$. *Phylogeny T is said to have a positive error if* $\texttt{length}(T) > \texttt{length}(T_{true})$ *and negative error if* $\texttt{length}(T) < \texttt{length}(T_{true})$.

Note that it is impossible for T_{min} to have positive phylogenetic error. This is because our algorithm optimizes over all possible haplotypes consistent with the given set of genotypes and selects the one that minimizes the size of the phylogenetic tree. In contrast, T_{phase} can suffer from both types of errors and it is impossible to know if the size of the true phylogeny is larger or smaller than T_{phase}. The following definition of an *imperfection* of a phylogeny has been widely used.

Definition 21 *The* imperfection *of a phylogeny T constructed for an input set of sequences (genotypes or haplotypes) with m varying sites is* $\texttt{length}(T) - m$.

Simply stated, the imperfection is the minimum number of *recurrent* mutations required to explain the sequences using the phylogeny. Notice that if there are m varying sites in an input set of genotypes then every possible set of haplotypes that explain it must have m varying sites as well. The experiments presented in the following section allow us to understand the gap between the size of the phylogeny from genotypes, the true size and the artificially inflated sizes due to incorrect phase inference.

5.2.1 Simulated Data

Due to difficulty of obtaining phase-known autosomal data, we begin by examining simulated data. We used coalescent simulations to generate recombination-free haplotypes and genotypes for varying mutation rates and used these for a series of tests on how the accuracy of our method and the two comparative haplotype-based approaches varied with different parameter values. Each test measured the total number of errors of each method in 200 independently generated data sets. We first varied the mutation rate parameter θ to test its influence on the accuracy of all the methods. The results are provided in Figure 5.4. We find that the relative performance of the three methods is fairly consistent. The greatest number of errors is generally made by `fastPHASE` and the least by direct phylogeny inference from the genotypes, with `haplotyper` in between. As one would expect, the number of errors of all three methods increases with increasing mutation rate. The curves are not monotonic, but additional simulation runs identical to those described here (data not shown) show no conservation of specific peaks and troughs of the graphs, indicating that they reflect only random noise due to a high variance in phylogenetic errors across trials. Table 5.1 separates the results of the two indirect methods, `fastPHASE` and `haplotyper`, into positive and negative errors. Both methods show mixtures of generally similar numbers of positive and negative phylogenetic errors with no apparent consistent trends towards favoring one or the other error type as one particular parameter varies.

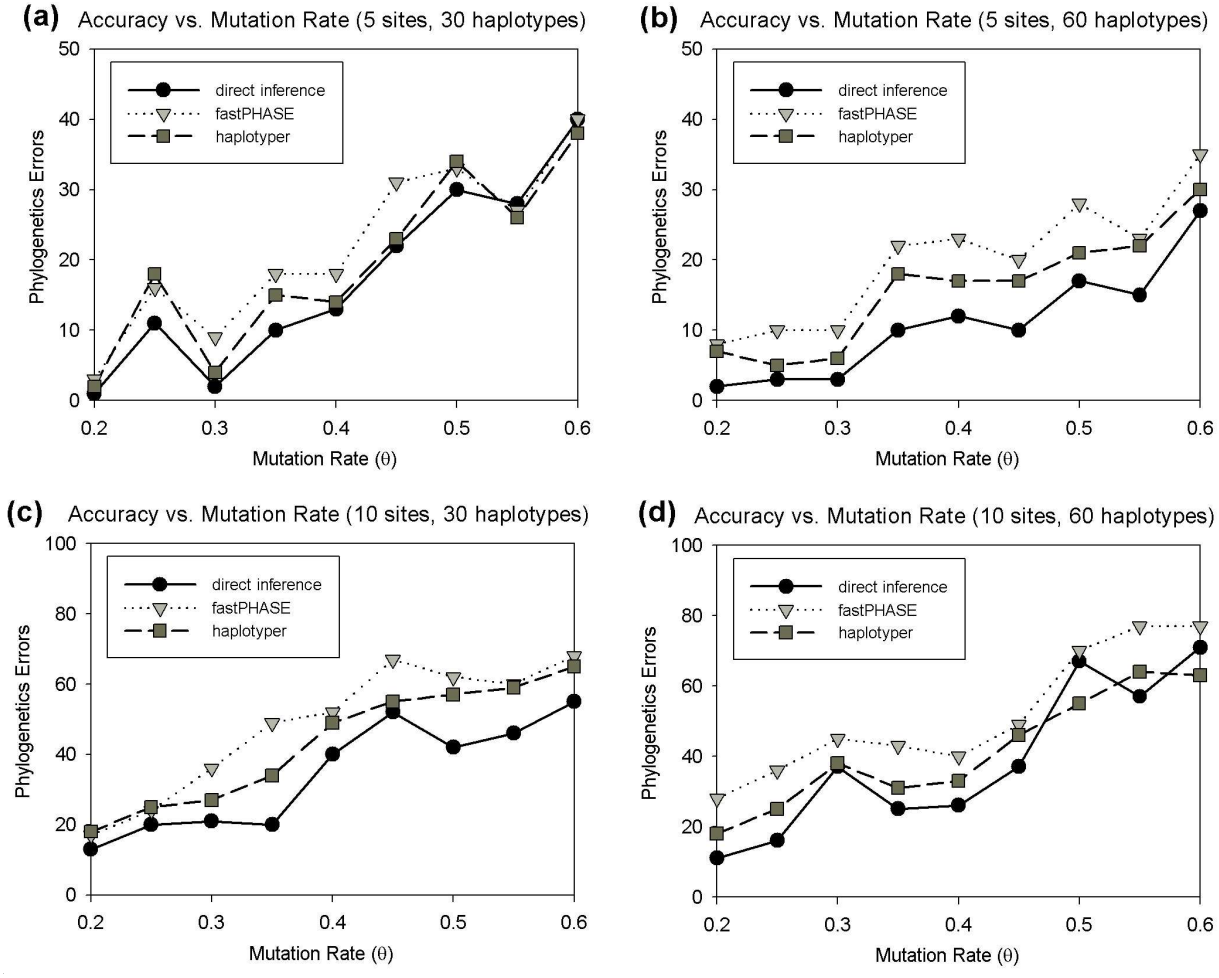


Figure 5.4: Each plot shows the phylogenetic errors for inferences from our direct inference method (black circles), indirect inference using fastPHASE (light grey triangles), and indirect inference using haplotyper (dark grey squares) as a function of the mutation rate θ. Plots are provided for two window sizes (5 and 10 SNPs) and for two population sizes (30 and 60 haplotypes). Each data point in the plot was computed by running each algorithm over 200 randomly generated data-sets. (a) window size 5, 30 haplotypes. (b) window size 5, 60 haplotypes. (c) window size 10, 30 haplotypes. (d) window size 10, 60 haplotypes.

m	n_H	method	θ=0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6
5	30	fastPHASE + err	2	8	8	13	7	15	14	8	14
5	30	fastPHASE - err	1	8	1	5	11	16	19	19	26
5	30	haplotyper + err	1	10	2	10	3	6	12	5	12
5	30	haplotyper - err	1	8	2	5	11	17	22	21	26
10	30	fastPHASE + err	7	12	27	38	28	38	38	27	42
10	30	fastPHASE - err	10	12	9	11	24	29	24	23	26
10	30	haplotyper + err	7	14	15	23	28	24	37	41	40
10	30	haplotyper - err	11	11	12	11	21	31	20	18	25
5	60	fastPHASE + err	6	7	7	15	16	13	16	11	14
5	60	fastPHASE - err	2	3	3	7	7	7	12	12	21
5	60	haplotyper + err	5	3	3	9	7	8	7	7	6
5	60	haplotyper - err	2	2	3	9	10	9	14	15	24
10	60	fastPHASE + err	24	25	25	29	28	32	43	54	41
10	60	fastPHASE - err	4	11	20	14	12	17	27	23	36
10	60	haplotyper + err	11	13	14	13	22	25	27	42	34
10	60	haplotyper - err	7	12	24	18	11	21	28	22	29

Table 5.1: The table separates the phylogenetic errors from the experiments of Figure 2 into positive and negative errors for indirect phylogeny inference using fastPHASE and haplotyper.

Note that by definition, our new method cannot produce positive errors and all errors it produces therefore reflect underestimates of phylogeny size.

We next tested variation in accuracy with the number of haplotype sequences sampled for fixed mutation rate with $\theta = 0.5$. The results are shown in Figure 5.5. Our direct methods show a slightly more pronounced advantage for 10-SNP windows than 5-SNP windows. This could simply be due to higher variance in the results of the 5-SNP windows. Table 5.2 shows the breakdown of the indirect methods into positive and negative errors, with both indirect methods again showing a mixture of comparable numbers of positive and negative errors across the parameter range. `haplotyper` again shows generally better accuracy than `fastPHASE` by this measure. One might expect that with increase in the number of haplotypes, the number of mutations required to explain the data would increase as well. Therefore, the number of errors should increase with the number of haplotypes. This, however, does not seem to be the case in practice, an observation that can be explained by the fact that greater numbers of haplotypes provides more information and thus yield improved accuracy in phase inference. Therefore, the number of phylogenetic errors roughly stay the same with the increase in the number of haplotypes for all the methods.

5.2.2 Mitochondrial DNA

The next step in our analysis used mitochondrial DNA (mtDNA), which is naturally haploid. Although one would not normally need to phase mitochondrial DNA, we use it in our validation because it provides a source of large numbers of known haplotypes and because

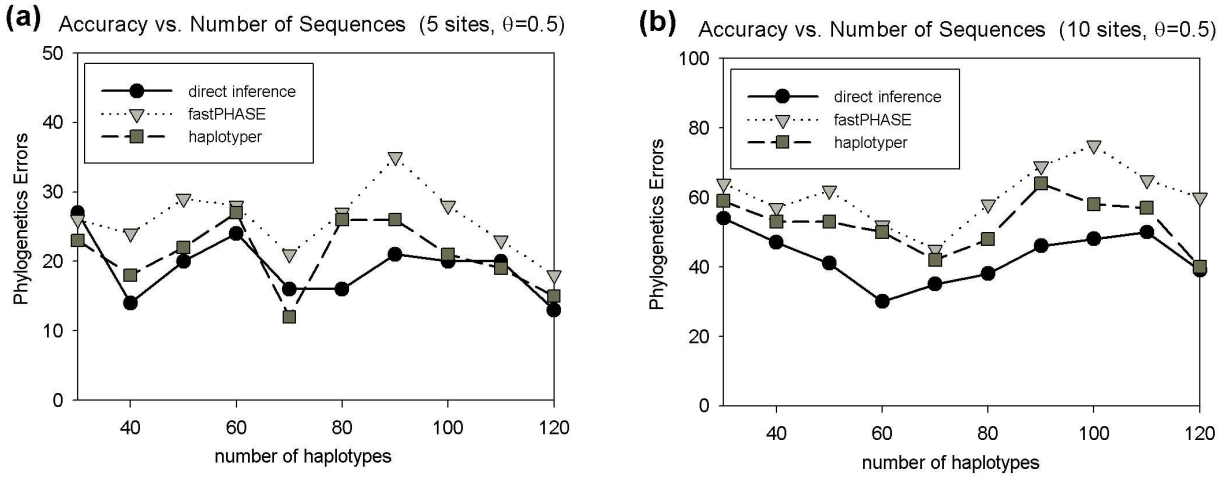


Figure 5.5: Each plot shows the phylogenetic errors for inferences from our direct inference method (black circles), indirect inference using fastPHASE (light grey triangles), and indirect inference using haplotyper (dark grey squares) as a function of number of input haplotypes. Plots are provided for two window sizes (5 and 10 SNPs). Each data point in the plot was computed by running each algorithm over 200 randomly generated data-sets. (a) window size 5. (b) window size 10.

m	method	n_H=30	40	50	60	70	80	90	100	110	120
5	fastPHASE + err	10	12	15	14	11	15	25	15	10	10
5	fastPHASE - err	16	12	14	14	10	12	10	13	13	8
5	haplotyper + err	5	6	7	10	3	11	12	7	5	4
5	haplotyper - err	18	12	15	17	9	15	14	14	14	11
10	fastPHASE + err	38	32	43	34	31	41	46	53	41	38
10	fastPHASE - err	26	25	19	18	14	17	23	22	24	22
10	haplotyper + err	32	26	31	28	35	23	36	30	23	14
10	haplotyper - err	27	27	22	22	17	25	28	28	34	26

Table 5.2: The table separates the phylogenetic errors from the experiments of Figure 3 into positive and negative errors for indirect phylogeny inference using fastPHASE and haplotyper.

it provides a good model of recombination-free DNA. The lack of recombination in the mitochondrial DNA means that if the most parsimonious phylogeny on the genotypes is q-imperfect, then that region must have undergone a minimum of q recurrent mutations. The mitochondrial genome contains known regions of high mutation rate that allow us to validate the ability of phylogenetic imperfection to identify true sites of recurrent mutation, a key application of our method. For the purpose of these tests, we generated artificial diploids by randomly combining 60 mitochondrial complete sequences (16,569 bases) from a data set of Fraumene et al. [54] to produce thirty diploids. We then computationally inferred haplotypes from the all of the genotypes using `fastPHASE`. `Haplotyper` was omitted from these tests because the data set was larger than it could process. We then constructed phylogenies for all sliding windows of 50 bases across the data set by each of three methods: maximum parsimony using true haplotypes, inferred haplotypes and directly from the genotypes. Our method required 116 seconds on a desktop Linux PC to reconstruct the phylogenies for all the sliding windows, clearly demonstrating its practical efficiency.

Figure 5.6 shows the results for two regions of the mitochondrial D-loop that are known to have unusually high mutation rates [113]. The intervening sequence between these two regions, where mutation rate is low, is not shown since all windows have true imperfection zero. The genotype imperfection is identical to the true imperfection for the large majority of windows (zero positive and negative phylogenetic errors). While inferences from genotypes could err in the direction of underestimating the true haplotype imperfection, they nonetheless appear in practice to provide very good estimates of the true imperfection on these data. Genotype imperfection is never less than one below the true imperfection, i.e., at most 1 negative phylogenetic error for any window. Imperfection from inferred haplotypes is usually higher than the true imperfection in the imperfect regions, often substantially so, demonstrating that incorrect phasing can lead to large phylogenetic errors.

5.2.3 Phase-known Autosomal DNA

Only a very limited amount of true phase-known autosomal data is available. We chose to examine a set taken from the lipoprotein lipase (LPL) gene [19], which is the only true phase-known data publicly available that has a sufficiently large population sample and number of SNPs to provide a challenging test for the methods considered here. The dataset consists of 144 haplotypes (77 distinct) belonging to three different ethnicities and 86 SNPs. The true genotypes corresponding to the haplotypes were not published for this data sets and so we duplicated the first haplotype to obtain 78 distinct sequences and then randomly paired them to produce 39 artificial genotypes from the true haplotypes. As in the previous case, we ran `fastPHASE` and `haplotyper` on all of the SNPs put together to obtain inferred haplotypes.

Unlike mtDNA, the autosomal chromosomes undergo recombinations and so we used the `HAP` webserver [65] to break the 86 SNPs into blocks. We obtained 22 blocks which we assume to be recombination-free. We then estimated the size of the phylogenies within each of the blocks separately from the true haplotypes, inferred haplotypes and genotypes directly. Note that we would expect this to be a particularly difficult dataset for our

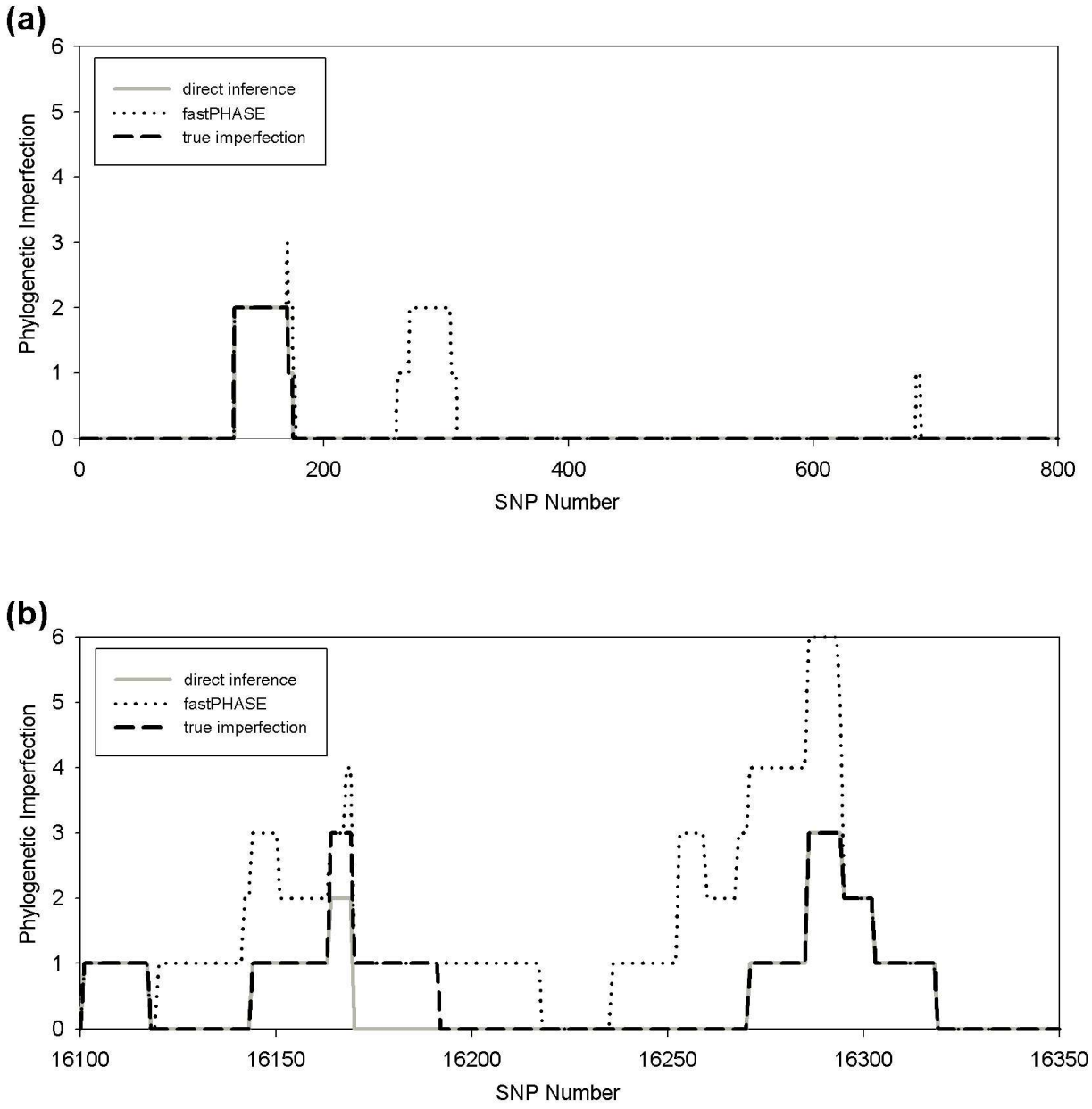


Figure 5.6: Imperfection around two high-variation segments (bp 1:800 and 16100:16350) of the D-loop of the mtDNA. Each position on the x-axis denotes the central nucleotide of the window examined. The y-axis shows the inferred imperfection by our direct method (solid grey line), imperfections inferred by the indirect method using fastPHASE (dotted black line), and the true imperfection (dashed black line). (a) bp 1 to 800. (b) bp 16100 to 16350.

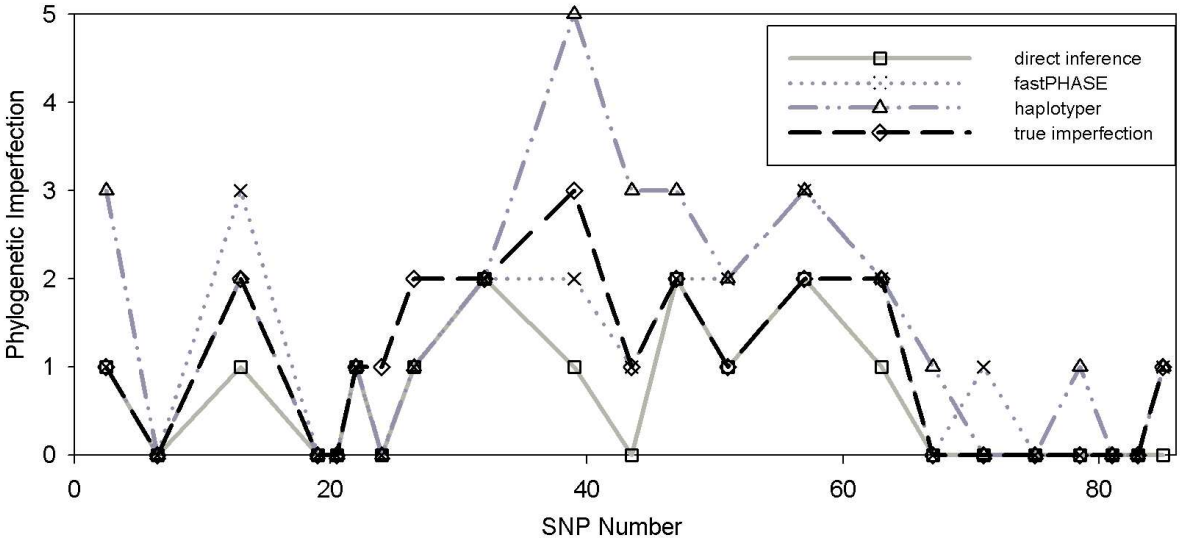


Figure 5.7: Imperfections for 22 blocks from LPL. Each data point has an x-coordinate corresponding to the central SNP of a given block and a y-coordinate corresponding to the imperfection of the inferred phylogeny on that block. Data is shown for our direct method (solid grey line with squares), indirect inference with fastPHASE (dotted grey line with X's), indirect inference with haplotyper (dash-dot grey line with triangles), and the true imperfection (dashed black line with diamonds).

algorithm because `haplotyper` and `fastPHASE` made inferences from all the SNPs at once, whereas our method was run on each block independently.

The results are shown in Figure 5.7, where the x-coordinate of each point is the central SNP of the block and the y-coordinate is the imperfection in that block. Most of the blocks are imperfect. On this dataset, in contrast to the prior ones, the direct and indirect approaches showed almost equal total accuracy, with `haplotyper` being slightly worse. This difference may reflect a failure to eliminate all recombination from the data set or might be because any advantage of direct inference is too modest to stand out on such a small data set. Even on a dataset that would be expected to be unusually easy for a phasing program, though, our method does no worse than the indirect approach. This dataset also suggests that the two approaches could be used in a complementary fashion, as the methods often bracket the true answer from opposite directions.

5.2.4 Resource Usage

We have, finally, examined the performance of our method in run time and space usage using additional simulation tests. We examined a range of data set sizes from 30 to 120 genotypes for fixed mutation rate $\theta = 0.5$ for 5-SNP and 10-SNP windows using averages for 200 repetitions per parameter value. Run times were measured for our method and for `fastPHASE` and `haplotyper`. Figures 5.8(a) and 5.8(b) shows run times for the method for

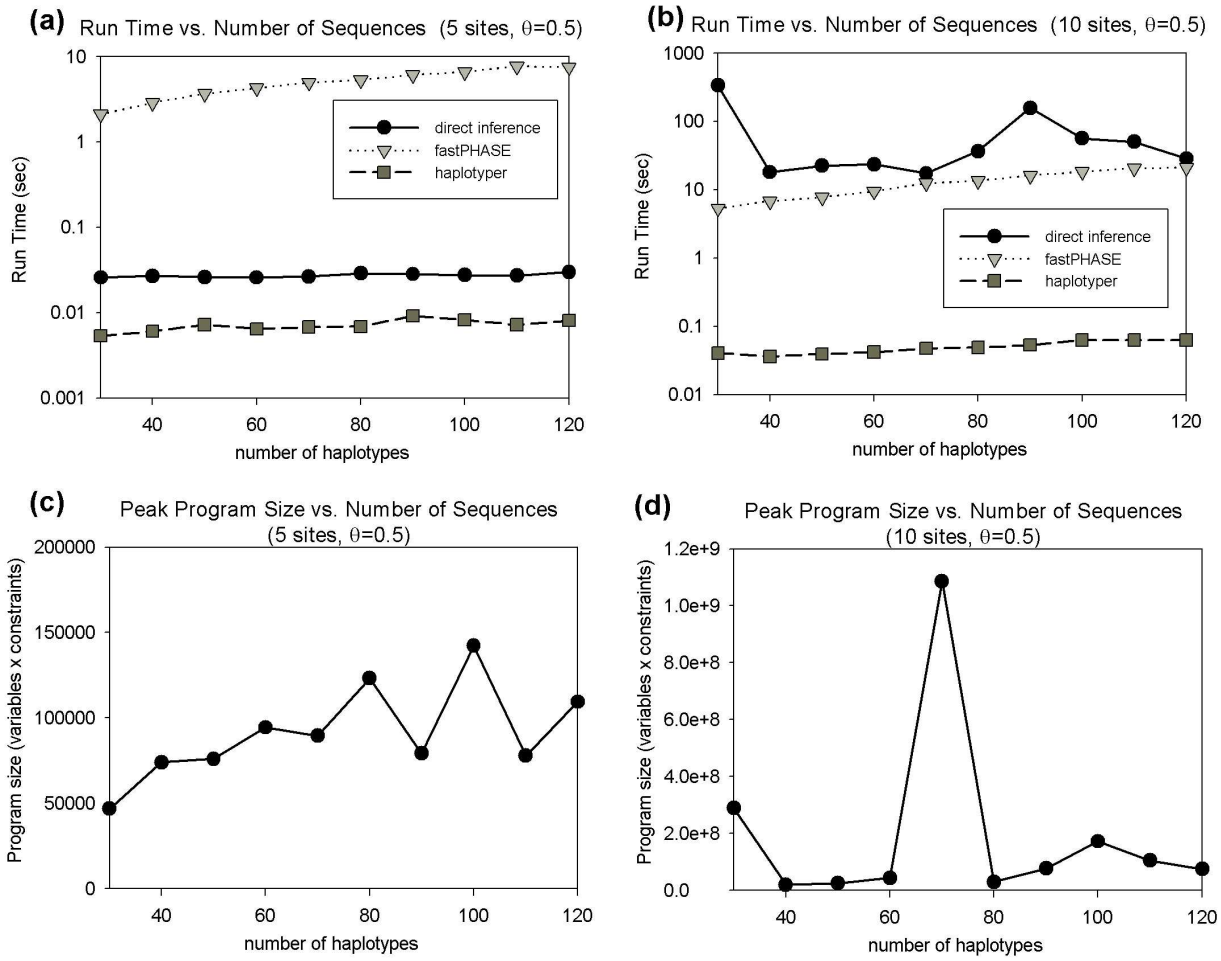


Figure 5.8: Each plot measures performance for fixed mutation rate $\theta - 0.5$. Run time is measured in seconds processor time required per parameter value, averaged over 200 independent runs. Run time data is provided for our direct method (solid block line with circles), fastPHASE (dotted line with triangles), and haplotyper (dash-dot line with squares). Space usage is measured in maximum linear program size in variables $\times$ constraints over the full branch-and-bound execution, averaged over 200 independent runs. (a) Run time performance on 5-SNP windows. (b) Run time performance on 10-SNP windows. (c) Space usage on 5-SNP windows. (d) Space usage on 10-SNP windows.

5- and 10-SNP windows, respectively. Our method is consistently faster than `fastPHASE` and slower than `haplotyper` for 5-SNP windows. Like `haplotyper` and unlike `fastPHASE`, our method appears insensitive to the number of input sequences. Our method shows a substantial slowdown in moving from 5-SNP to 10-SNP windows. While the method is faster than `fastPHASE` for 5-SNP windows it is on average a few times slower with 10-SNP windows. This slowdown is to be expected since our method constructs a program of potentially exponential size in window size. `haplotyper` is consistently the fastest of the methods for both window sizes.

We further assessed space usage of our method based on the maximum linear program relaxation size examined over the course of a given problem instance, averaging this value over the 200 trials. Here size is expressed as the product of the number variables and constraints. Figures 5.8(c) and 5.8(d) show the results for 5- and 10-SNP windows. The results show a high degree of noise, with a single outlier point requiring roughly 100-fold more space than the others. Nonetheless, program size appears generally to increase with number of input sequences. Space usage also increases substantially with window size, which we would again expect given that worst-case program size is exponential in window size.

5.3 Conclusions

We have developed the first practical, general methods for finding maximum parsimony haplotypes from unphased genotype data and have used them to assess the costs introduced by computational phasing prior to phylogenetic inference. Our methods used a collection of heuristics based on the theory of Steiner trees, a variant of a flow-based ILP, and a branch-and-bound approach to solve problem instances with high imperfection that were not solvable by any prior method. While the method presented here is specific to the problem of inferring purely mutational phylogenies, similar approaches may prove productive for inference of ancestry by more general models of molecular evolution, such as ancestral recombination graphs (ARGs). Empirical tests on simulated and semi-simulated data show that direct phylogeny inference from genotypes leads to fewer errors than does the standard practice of building phylogenies from phased data. Methods for this problem have several practical applications. Most important is to estimate the minimum number of recurrent mutations required to explain a set of observed genotypes. A large such value may indicate frequent recurrent mutation or gene conversion or a selective pressure to recurrently alter a given allele. Researchers trying to establish such effects need to ensure that the size of the phylogeny is not an artifact of phase inference. The method should similarly be useful for improving estimates of local mutation rates. Other applications include improving the power of association tests by eliminating spurious effects from recurrent mutation, and providing alternative methods for detecting recombination-free autosomal regions and performing phase inference from genotype data.

Chapter 6

Whole Genome Phylogenies

In this chapter, we apply apply some of the algorithms developed to construct piece-wise phylogenies on a genome-wide scale. In general, computational and statistical genetics depend on a foundation of mathematical models of population and sequence evolution. While such models are always simplifications, they are essential for framing computational inferences and posing statistical hypotheses. Furthermore, finding where these models break down can be an excellent way to deepen our understanding of the basic biological processes shaping genome sequence variation. As we have seen in prior chapters, the *infinite sites* model of genome evolution is one important example of such a highly simplified but widely used model. The model proposes that over relatively short time scales, mutations appearing in a population will occur at distinct genetic loci. Phylogenetic trees produced by mutational evolution in this model are called *perfect*. The perfect phylogeny assumption has given rise to several key advances in computational genetics, many such algorithms have been explored in previous chapters and includes highly efficient algorithms for phylogeny inference from haploid sequence data [57], as well as from unphased genotype data [59] when the data is in fact consistent with a perfect phylogeny.

Unfortunately, the infinite sites model and perfect phylogeny assumption will not generally be true in practice, even over relatively short time scales. Deviation from a perfect phylogeny, called *phylogenetic imperfection*, might derive from recurrent mutation (homoplasy), recombination, gene conversion, or possibly other unknown factors. When the sequence data cannot be reconciled with a perfect phylogeny, more sophisticated and computationally demanding algorithms are needed for phylogeny reconstruction. In prior chapters, we described efficient algorithms for inferring imperfect phylogenies [9, 105, 106], showing that such problems can be efficiently solved when the deviation from a perfect phylogeny is small (a situation called a *near-perfect* phylogeny [51]).

We also developed algorithms to reconstruct general maximum parsimony phylogenies irrespective of whether they are near-perfect or not [107]. In practice, these algorithms were faster when the phylogenies are closer to perfect. In this chapter, we apply this algorithm on sliding windows of small numbers of varying sites to allow for high-throughput analysis of phylogenetic imperfection on genomic scales. We annotate the phased HapMap [75] with the imperfection of a small neighborhood around each SNP site in the dataset. This annotation essentially provides the number of recurrent mutations required to explain

each region as a purely mutational phylogeny. Sites that have been subject to recurrent mutation should therefore produce windows with small amounts of imperfection. We show that imperfection scores are indeed highly correlated with recombination rate but the correlation drops significantly after removing hot-spots of recombination. We further show, that these imperfection scores exhibit local regularities and population conservation beyond what can be explained by recombination rate variation. We conclude with an analysis of outlier windows of exceptionally high imperfection that fall outside known recombination hotspots.

6.1 Algorithm and Data Analysis

For this chapter, we work with phased haplotypes as input and therefore we can assume that the major allele at each site is represented by '0' and the minor allele by '1'. The input data is denoted by I and is an $n \times m$ binary matrix. The n rows $R(I)$ correspond to taxa typed at m binary markers. Readers can refer to See Definitions 2, 4, 6, 5 for formal descriptions of a phylogeny, its length, the penalty and maximum parsimony (optimality). From the definitions, it should be clear that the parameter q of a q-near-perfect phylogeny is precisely the minimum number of recurrent mutations required to explain the genetic diversity in the observed set of sequences. Given a binary matrix as described, we use the algorithm described in Chapter 3 to find the most parsimonious phylogeny. Using this phylogeny, we can trivially obtain penalty (imperfectness) q of the underlying input sequences.

6.1.1 Data Sets

This study primarily uses data from the International Haplotype Map (HapMap Phase II) [75] for the purpose of conducting a fine-scale genome-wide scan of human genetic variations. Because it is currently computationally infeasible to identify maximum parsimony trees on moderately imperfect unphased data sets [105], we use only phased data. We restricted ourselves to the HapMap CEU population of Utah residents of European ancestry and the YRI population of residents of Yoruba in Ibadan, Nigeria because these subpopulations were genotyped for parent-child trios and thus have minimal phasing error. The other two HapMap data sets (Han Chinese in Beijing, China and Japanese in Tokyo, Japan) were genotyped only for unrelated individuals and were therefore omitted here. All HapMap data sets were downloaded in phased form from the HapMap web site, where the PHASE program [110] had been used to identify most likely phases from the trio data. This HapMap build was based on the NCBI human genome assembly build 35 [48]. SNP location assignments and genomic coordinates are therefore based on NCBI build 35. The resulting data contained 120 haplotypes from 60 unrelated individuals for each of the two populations typed at about 3.7 million SNPs.

Several additional datasets were used to study correlation of imperfection with other sequence features. We retrieved the set of non-synonymous coding SNPs (ncSNPs) mapped to the build 35 genome using the Ensemble BioMart tool [29, 45], selecting all ncSNPs with

validated assays. Fine-scale recombination rates and recombination hotspots were retrieved from the HapMap web site [75]. Locations of all short tandem repeats in the human genome were retrieved from the UCSC Genome web site [33]. The set of all human repeats was based on RepeatMasker [103] inferences and was also retrieved from the UCSC Genome resources Table View tool. The locations of high-scoring hits were also manually examined using the UCSC Genome Browser [78] and the dbSNP resource at the NCBI web site [102] to identify the genes and repetitive regions containing the particular SNPs of interest and, for coding SNPs, to identify their corresponding amino acid changes.

6.1.2 Statistical Calculations

Mutual information. In order to test for regional variations in phylogenetic imperfection, we calculated mutual information between windows at varying genomic distances. All pairs of disjoint 5-SNP windows were compared and grouped into buckets based on the genomic distance between the central SNPs of the windows. Bucket widths of 1 kb and 100 kb were used in separate tests. For each bucket, the imperfection of the first window and the imperfection of the second window were treated as random variables and their individual and joint entropies and mutual information were calculated. Given a sequence of points $x_1, \ldots, x_m$ drawn from a countable set of values $\{c_1, \ldots, c_n\}$, where f_i represents the fraction of points with value c_i, the entropy of the sequence is defined as

$$H(x) = -\sum_{i=1}^{n} f_i \log(f_i)$$

The joint entropy of a set of paired data points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, where f_{ij} is the fraction of points with the value (c_i, c_j), is calculated by the formula

$$H(x, y) = -\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \log(f_{ik})$$

The mutual information of the set is then defined as $H(x) + H(y) - H(x, y)$. Mutual information determines the dependence between the two imperfection values, with larger values denoting higher predictability of one window's imperfection given the other.

Imperfection versus recombinations. In order to compare imperfection and fine-scale recombination rates, we first identified for each window in our data set the location of the central SNP in the window. We then retrieved the fine-scale recombination rate at each such SNP from the HapMap-supplied data. The result was two paired lists of data points. We calculated correlation coefficients for the two lists for each chromosome individually and for all chromosomes collectively. A curve was fit to the data points by proposing that imperfection i is related to recombination rate r by a function of the form $i = a(1 - e^{-br})$ and using Newton-Raphson iteration to find the least-squares best fit parameters a and b. We calculated the correlation coefficients with the original data-set as well as with windows spanning recombination hotspots removed.

6.1.3 Computer Resources

Code for phylogenetic imperfection calculations was written in the C++ programming language using Cplex 10.0 library. C++ code was also used to identify a best-fit exponential curve for the relationship between imperfection and fine-scale recombination rate. All other data processing and statistical computations were performed with code written in Perl.

6.2 Results

The section presents the results of constructing maximum parsimony phylogenies over the whole genome. We first show the results of genome-wide scans of the imperfection values. Following that we compare the imperfection values and recombination rates. While imperfection values are correlated to recombinations we show that recombinations far from explain the observed imperfections. We then look at structural elements of the genome and its relationship to imperfection values. Finally, we examine the parts of the genome that show very high imperfection.

6.2.1 Genome-wide Imperfection Scan

Figure 6.1 plots imperfection of overlapping 5-SNP windows across all human autosomal chromosomes, mapping each window score onto the position of the central SNP in that window. Windows spanning recombination hotspots were removed. Imperfection scores around telomeres and centromeres are generally sparse because few SNPs were typed in those regions. The results show substantial variability in imperfection on multiple length scales. Peaks of high imperfection can occur in isolation or as part of regional variations. In addition, imperfection appears in general to be higher towards the telomeres and lower towards the centromeres across chromosomes, although with considerable variability within the regions. Although the amount of data makes it impossible to fully appreciate visually, the full set of SNP-by-SNP imperfection scores are included as supplementary table S1.

The CEU and YRI populations generally appear to share common locations for regions of high or low imperfection, but have different absolute amounts of imperfection. Over all chromosomes, the CEU data shows a mean imperfection of 0.3 and the YRI a mean imperfection of 0.55. These results may reflect the higher genetic diversity of African versus European populations. The two populations do, however, show a strong overlap in regions of high or low imperfection of one versus the other. Overall, comparison of window scores with shared central SNPs across the two populations yields a correlation coefficient of 0.36.

To further test the observation of regional substructure, we examined the mutual information between imperfection scores at pairs of non-overlapping windows. Mutual information is an information theoretic measure that informally describes how much the variability in the two sites individually exceeds their variability when considered collectively. High mutual information indicates that two sites are highly predictive of each other, while low mutual information suggests that they are nearly independent. Figure 6.2A shows the fine-scale dependence, plotting mutual information for distances 0-100 kb in 1 kb buckets. The plot shows for both populations a sharp spike for the closest windows (0-1 kb apart)

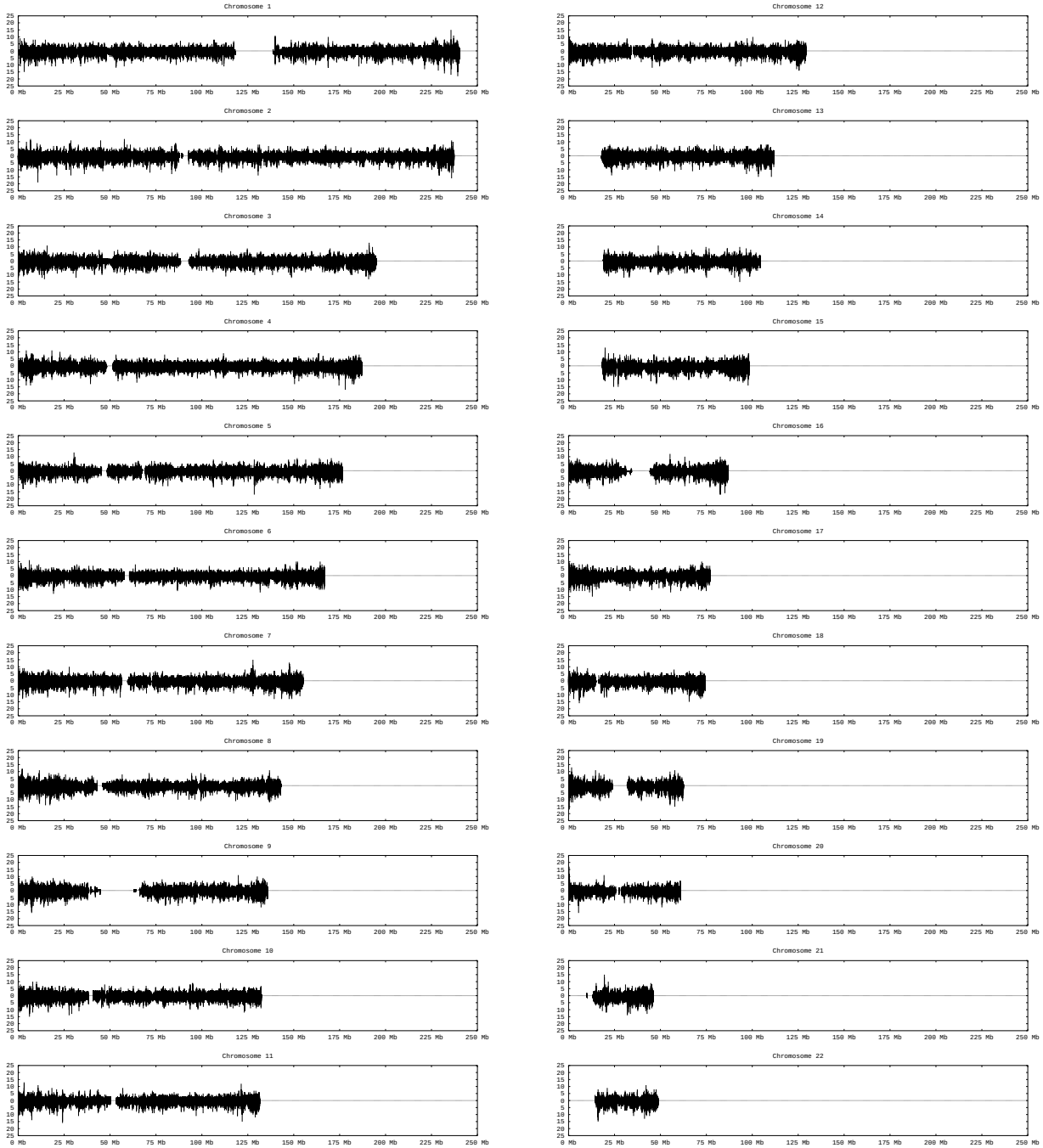
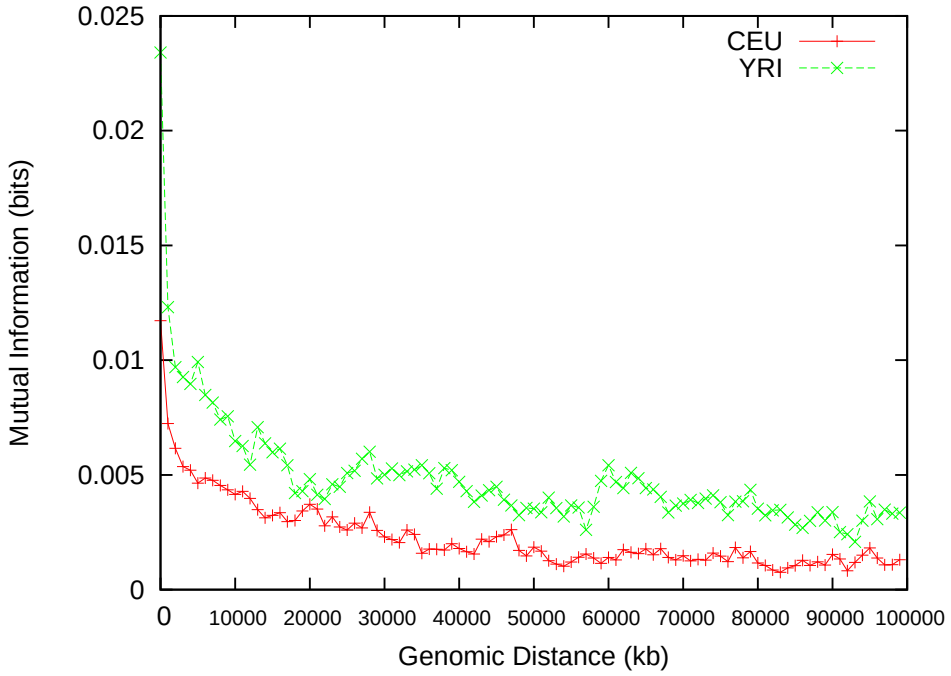
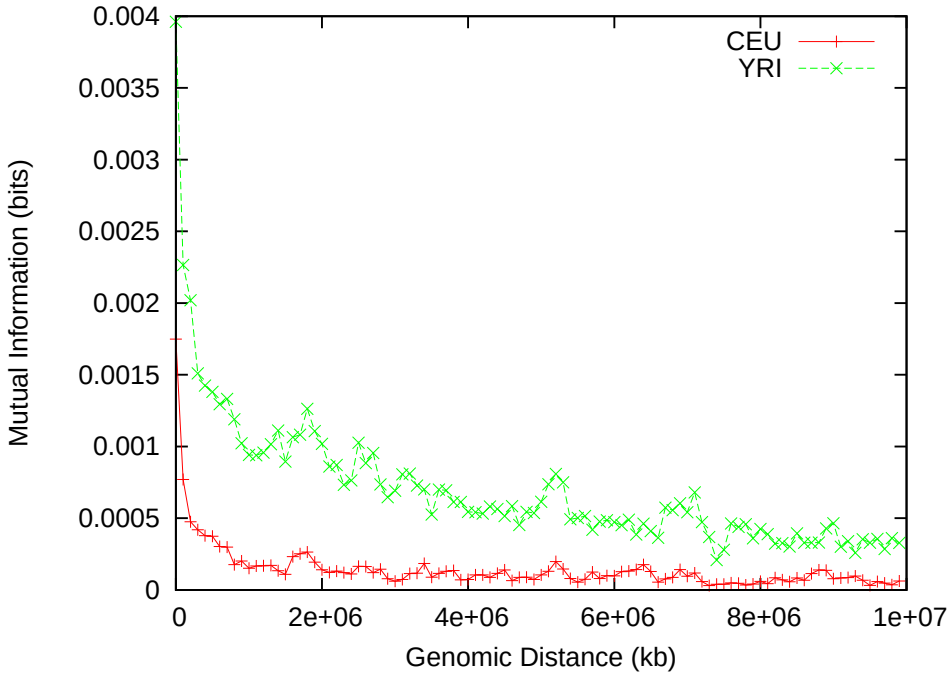


Figure 6.1: Genome-wide scans of phylogenetic imperfection. Each plot shows imperfection for overlapping 5 SNP windows across a single autosomal chromosome. Plots are based on haplotypes determined from trios from the CEU and YRI populations from the HapMap. CEU imperfections are plotted above the x-axis and YRI imperfections below the x-axis.



A



B

Figure 6.2: Dependence of phylogenetic imperfection between windows grouped by distance between the central SNPs of the corresponding windows. Graphs show mutual information between imperfection levels of non-overlapping windows whose central SNPs fall within a given distance range. A: Plot of fine-scale correlation, showing mutual information for 1 kb bins up to 100 kb. B: Plot of coarse-scale correlation, showing mutual information for 100 kb bins up to 10 Mb.

followed by a rapid drop and then a slow decline for the majority of the window. Figure 6.2B plots dependence at a coarser scale of 0-10 Mb measured in 100 kb buckets. The pictures at the two scales are qualitatively similar, showing a rapid drop for both populations near the origin, followed by a more gradual decline for the remainder of the range covered. The results suggest distance dependence occurs on multiple scales, with strong correlations for sites within a few kilobases of one another, but still a noticeable excess of information above background levels even at megabase scales. While both populations yield qualitatively similar results, the YRI population shows in general a somewhat greater mutual information between windows than does the CEU population.

6.2.2 Imperfection as a Statistic of Fine-scale Recombination

We next sought to test the hypothesis that imperfectness is a useful statistic for studying fine-scale recombination rate. We can gain an intuitive understanding of the value of imperfection as a measure of recombination rate by comparing it to an established measure. Figure 6.3A illustrates the correlation between local imperfection and fine-scale recombination rate as assessed by the method of MacVean et al. [83, 85] for chromosome 21. We chose chromosome 21 for illustrative purposes as it is small enough that fine-scale features can still be discerned in a whole-chromosome plot. The image reveals that spikes in local recombination rate almost invariably correspond with spikes in local phylogenetic imperfection. The converse does not follow, however; many peaks in phylogenetic imperfection coincide with low inferred recombination rates. Figure 6.3B just examines the windows that fall outside recombination hotspots and clearly shows the high imperfection values at regions of low recombination rates. This observation suggests that the phylogenetic imperfection measure detects both recombination and frequently recurring mutation. Regions of sustained low recombination rate, such as that observed around 28 Mb do appear to correspond to sustained low imperfection.

Tables 6.1 and 6.2 assesses the significance of the coincidence between fine-scale recombination rate and phylogenetic imperfection excluding and including recombination hotspots respectively. The graphs show significant correlations ($p < 0.001$) for all chromosomes and for both populations. Correlations are nearly identical for the two populations when all SNPs are examined and are slightly higher for YRI when recombination hotspots are excluded. Overall the correlation is typically higher than 0.4 when recombination hotspots are included and drops down to as low as 0.23 when the hotspots are excluded. This is a fairly large reduction in the correlation. These correlations are consistent with the notion that sources of imperfection is a conflation of recombination rate and recurrent mutation. While the correlation values are small for recombination rate against imperfection outside the hotspots, its significantly larger when we obtain the correlation of imperfection values between the two populations. Outside of the recombination hot-spots, the correlation is 0.36. This is a crucial observation that shows that the imperfection scores are significantly influenced by other local genomic properties than just recombination rates.

In order to better understand this correlation, we plotted a histogram of mean imperfection versus local recombination rate. Given the uneven distribution of data points, we used exponentially increasing bin sizes for recombination rates. After removing bins with

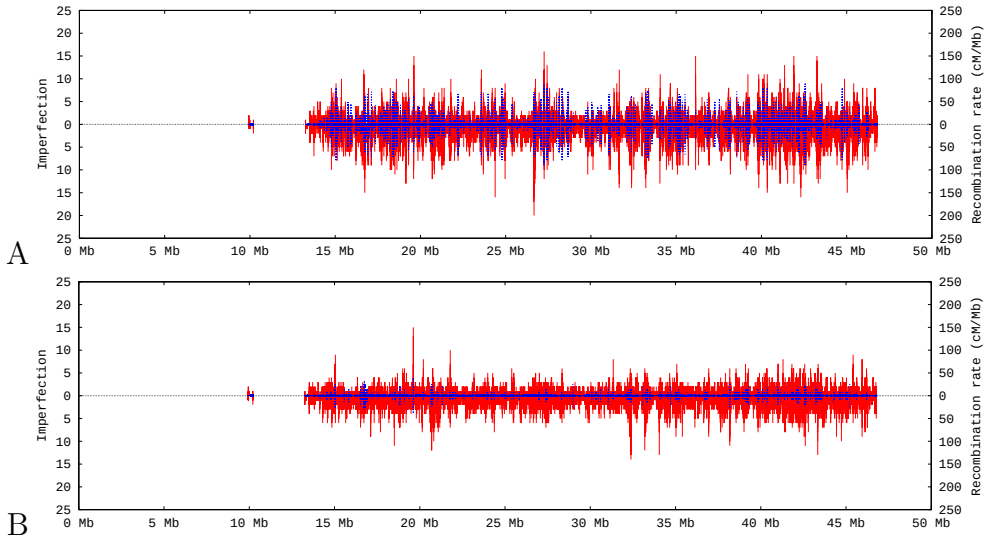


Figure 6.3: Coincidence of imperfection and fine-scale recombination rate for chromosome 21. Imperfection scores are shown as solid gray bars mapped to the position of the central SNP of the corresponding window. Fine-scale recombination rates, supplied by the HapMap web site [75] are marked by dashed black lines. CEU data appear above the x-axis and YRI data below the x-axis.

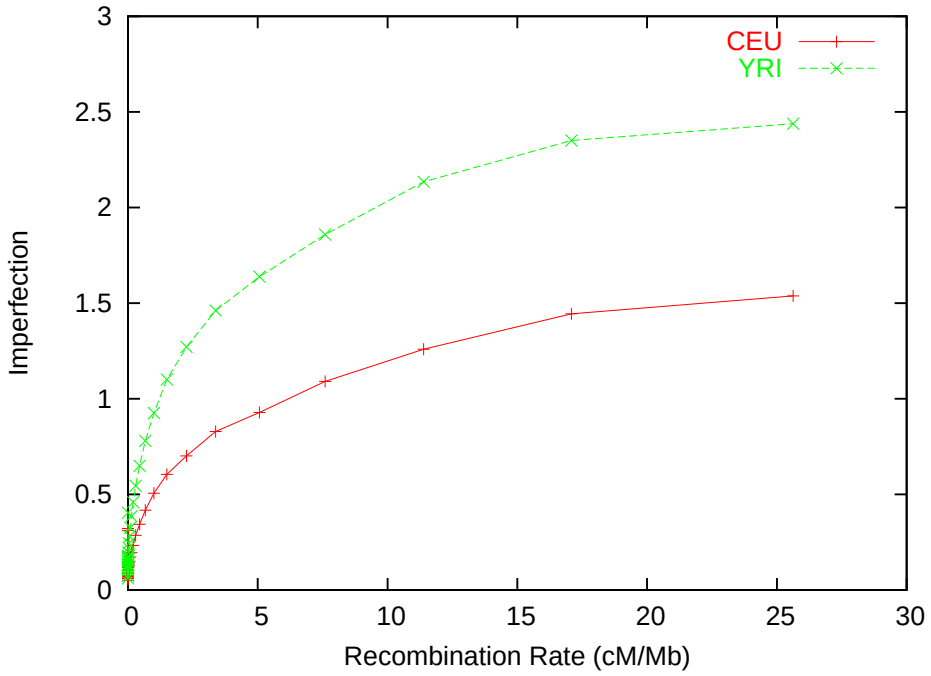


Figure 6.4: Dependence of mean imperfection on local recombination rate. The plot shows mean imperfection scores grouped into binned recombination rates using bins of exponentially increasing size. Each point plots the mean imperfection of a single bin on the y-axis with the lower endpoint of that bin's range on the x-axis.

Chromosome	r_{CEU} [a] (CEU)	r_{YRI} [b] (YRI)
1	0.23	0.30
2	0.24	0.29
3	0.26	0.30
4	0.24	0.28
5	0.25	0.30
6	0.25	0.29
7	0.26	0.30
8	0.27	0.31
9	0.24	0.30
10	0.24	0.27
11	0.26	0.30
12	0.23	0.28
13	0.26	0.31
14	0.27	0.31
15	0.28	0.32
16	0.28	0.31
17	0.32	0.34
18	0.27	0.34
19	0.32	0.33
20	0.28	0.28
21	0.24	0.27
22	0.29	0.34

(a)Correlation coefficient in the CEU population.
(b)Correlation coefficient in the YRI population.

Table 6.1: Chromosome-by-chromosome correlations of local phylogenetic imperfection and fine-scale recombination rate outside of recombination hotspots.

Chromosome	r_{CEU} [a] (CEU)	r_{YRI} [b] (YRI)
1	0.37	0.39
2	0.39	0.41
3	0.41	0.41
4	0.39	0.40
5	0.42	0.42
6	0.42	0.41
7	0.41	0.39
8	0.41	0.41
9	0.37	0.38
10	0.38	0.39
11	0.39	0.39
12	0.40	0.38
13	0.42	0.43
14	0.44	0.44
15	0.43	0.41
16	0.38	0.37
17	0.42	0.41
18	0.42	0.45
19	0.38	0.39
20	0.37	0.36
21	0.43	0.41
22	0.39	0.40

(a)Correlation coefficient in the CEU population.
(b)Correlation coefficient in the YRI population.

Table 6.2: Chromosome-by-chromosome correlations of local phylogenetic imperfection and fine-scale recombination rate including recombination hotspots.

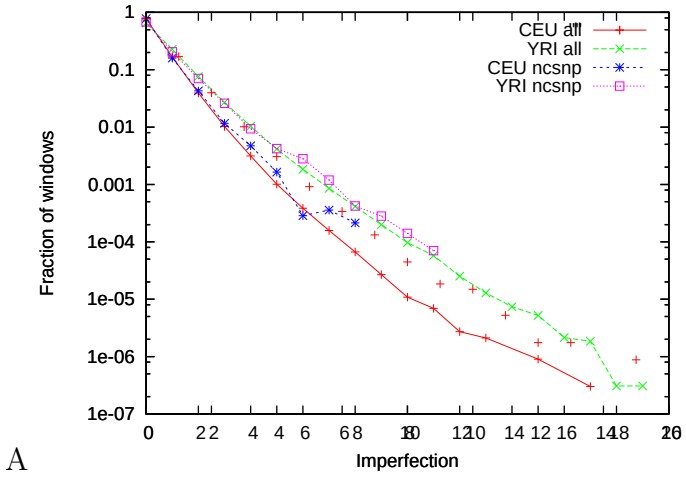
less than 100 data points, the bin with the smallest rate was $(1.5^{-26}, 1.5^{-25})$ and the largest was $(1.5^2, 1.5^3)$. Figure 6.4 presents the results. Both populations show imperfection near zero for the smallest recombination rates, rising smoothly before leveling off for high recombination rates. The rate of increase and the height of the apparent asymptote varies between the populations, though, with a more rapid rise and a higher asymptote for YRI (roughly 2.5) compared to CEU (roughly 1.5). We conjectured that the data could be fit by exponentially decaying curves of the form $i = a(1 - e^{-br})$. A least-squares fit to this form resulted in the parameters $a = 0.82$ and $b = 1.02$ for the CEU data and $a = 1.36$ and $b = 1.27$ for the YRI data. Note that best-fit curves level off below the asymptote in both cases (a is the asymptote), suggesting that a single exponential cannot provide a good fit simultaneously to the low and high-recombination rate windows.

6.2.3 Imperfection by SNP Class

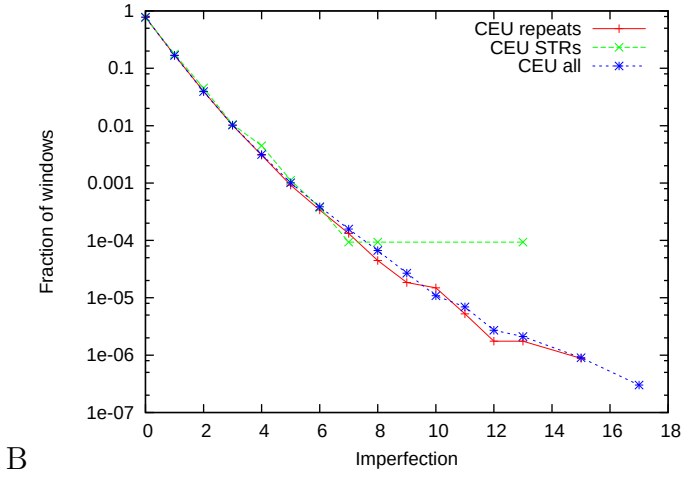
We next asked whether imperfection showed any significant variation based on the functional context of the region examined. We examined this question by comparing the overall distribution of phylogenetic imperfection scores across the genome for various functional classes of windows, with class defined by the sequence context of the central SNP in the window.

We first asked whether SNPs likely to be under selection showed any significant bias in imperfection scores. Frequencies of scores were computed for the complete set of windows and for the set of windows centered on validated, non-synonymous coding SNPs, a set chosen because they are likely to be under stronger selection than SNPs in general. Figure 6.5 show the results for the CEU and YRI populations respectively. A logarithmic scale is used to allow us to view the full range of frequencies, although this does understate the differences between the four data sets. At a gross level, all four data sets appear quite similar. Each follows an approximately geometric decay in frequency with increasing imperfection. It is clear, at all imperfections except 0, the fraction of windows for YRI is substantially and consistently larger than that of CEU. This is clearly not surprising since Africans are a more diverse and older population group than Europeans. Low imperfection scores account for a substantial majority of windows for all four sets and the absolute differences between the frequencies at the first few imperfection scores are small. ncSNP windows very slightly higher fraction of perfect phylogenies compared to other windows (77.92% versus 77.86% for CEU and 68.02% versus 65.13% for YRI). While there are greater differences between ncSNPs and general windows for larger imperfection scores, these can be explained by the small numbers of examples for the largest imperfection values. There therefore appears to be only a modest bias from selection on window imperfection scores.

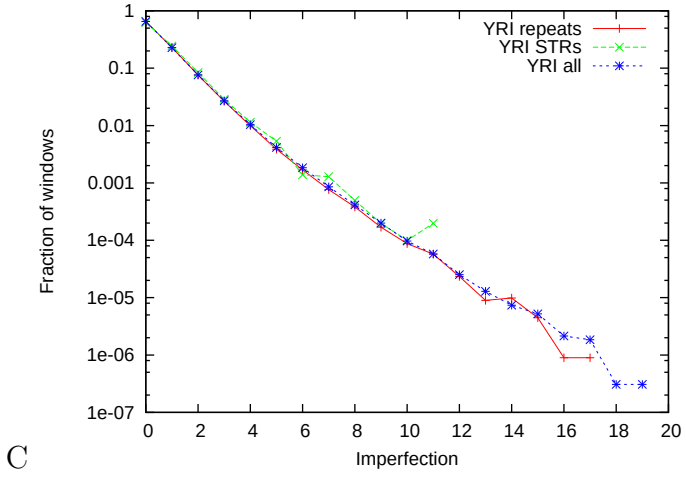
A bias in imperfection scores might also be expected for SNPs found in repetitive regions of the genome. We might anticipate some excess of imperfection in this set from a greater frequency of genotyping errors or genome mis-assembly around repetitive elements. Likewise, we might expect some errors to be introduced from a greater probability of mis-assembly of the genome around repetitive elements. We might also anticipate a higher fraction of large imperfection scores due to genuine hypermutable sites, which are known to be associated with some short tandem repeat (STR) regions [52, 115]. We therefore



A



B



C

Figure 6.5: Histograms of imperfection separated by sequence context. Frequencies are plotted for each imperfection score on a logarithmic scale. A: Comparison of all windows to windows centered on non-synonymous coding SNPs (ncSNPs) for CEU. B: Comparison of all windows to windows centered on ncSNPs for YRI. C: Comparison of all windows to all repeats and to simple tandem repeats (STRs) for CEU. D: Comparison of all windows to all repeats and to STRs for YRI.

central SNP[a]	chr.[b]	pos.[c]	i_{CEU} [d]	i_{YRI} [e]	sequence context
rs2486545	1	240569616	15	17	intergenic region
rs10174559	2	241396766	11	16	intron of KIF1A kinesin family member 1A
rs11683248	2	29360497	11	14	intron of ALK anaplastic lymphoma kinase (Ki-1)
rs7405052	16	84225446	7	17	intron of KIAA0182
rs12918736	16	84226491	7	17	intron of KIAA0182
rs10926263	1	236937389	8	16	intron of FMN2 formin 2
rs6037439	20	296188	17	7	intergenic region
rs7173687	15	24667811	9	15	intron of GABRA5 gamma -aminobutyric acid (GABA) A receptor, alpha 5
rs2493310	1	3317579	9	15	intron of PRDM16 PR domain containing 16
rs8045380	16	84226877	10	14	intron of KIAA0182

(a)refSNP ID of the central SNP of the window
(b)chromosome on which the window is found
(c)genomic map position of the central SNP
(d)imperfection in the CEU population
(e)imperfection in the YRI population

Table 6.3: Windows exhibiting the highest imperfection in the genome. This table identifies those windows with combined imperfection scores above 23.

compared the set of all windows with those whose central SNP falls in any repetitive region and those whose central SNP falls in an STR region. We also separately examined windows whose central SNP overlaps STR regions. Figures 6.5B and 6.5C show the results for CEU and YRI populations. The graphs again show nearly no differences between the data sets for the well-populated imperfection values. Comparing all repeat windows versus all windows, we find that the frequency of perfect windows is nearly identical (77.6% versus 77.9% for CEU and 65.1% in both data-sets for YRI) STR SNPs again do not show pronounced differences. They are slightly less likely to be perfect (76.1% versus 77.9% for CEU and 62.3% versus 65.1% for YRI). It therefore appears that repetitive elements do not lead to any dramatic systemic bias in local phylogenetic imperfection.

6.2.4 Imperfection Outliers

Although the primary purpose of this study was characterizing imperfection on the genomic scale, exceptional individual examples of imperfection can also be informative. There are too many perfect windows to allow for an examination of individual cases of small imperfection, but we can however, examine those windows of highest imperfection. Table 6.3 lists

central SNP[a]	chr.[b]	pos.[c]	i^d_{CEU}	i^e_{YRI}	gene [f]	variation[g]
rs1810247	15	19915318	5	11	LOC650137 seven transmembrane helix receptor	C85R
rs17690844	8	17656239	4	10	MTUS1 mitochondrial tumor suppressor 1	T453K
rs2368406	10	29824078	6	8	SVIL supervillin	A809P
rs2802808	1	201698085	4	8	NFASC neurofascin homolog	
rs4973588	2	233660480	4	9	NGEF neuronal guanine nucleotide exchange factor	T111M
rs7208422	17	73642170	8	5	TMC8 transmembrane channel-like 8	I306N
rs3751928	17	68792947	7	6	CDC42EP4 CDC42 effector protein (Rho GTPase binding) 4	
rs7627615	3	185301118	4	9	HTR3E 5-hydroxytryptamine (serotonin)receptor 3, family member E	T86A
rs10790715	11	124298892	5	7	HEPACAM hepatocyte cell adhesion molecule	V218M
rs557806	19	54069054	5	7	PPP1R15A protein phosphatase 1 regulatory (inhibitor) subunit 15A	P251R
rs1356410	15	40222129	5	7	PLA2G4F phospholipase A2 group IVF	V740M
rs351111	19	795020	6	6	PRTN3 proteinase 3 (serine proteinase neutrophil, Wegener granulomatosis autoantigen)	I119V

(a)refSNP ID of the central SNP of the window (b)chromosome on which the window is found (c)genomic map position of the central SNP (d)imperfection in the CEU population (e)imperfection in the YRI population (f)gene containing the central SNP of the window (e)amino acid change produced by the SNP. Annexin A13 has two splice isoforms resulting in two possible sites of variation.

Table 6.4: Windows centered on non-synonymous coding SNPs exhibiting the greatest imperfection.

the most extreme examples of imperfection observed in the data set. Of the 10 windows, 9 are centered in introns of genes and 1 in intergenic region. None of the top outliers occurs in the coding region of a gene.

The results from the previous section suggest the absence of coding SNPs is more likely due to their scarcity in the full SNP set rather than any bias against them and we therefore chose to examine ncSNP outliers separately. Table 6.4 describes those ncSNP windows yielding the highest imperfection. The set of high-scoring ncSNPs is not obviously particularly informative. The top-scoring hit is to a gene prediction that may in fact not be a functional gene. The others are each found in known proteins, but for most we know of no reason why they would be particularly disposed to high imperfection. The list contains several genes known to have relationship to human diseases and therefore might have high imperfection due to selective pressure. Mitochondrial tumor suppressor (MTUS1) has been shown to have significantly lower expression in pancreatic tumor cells. The absence of the gene causes excessive microtubule growth and inhibited spindle formation and therefore their mutation could be related to cancer. NFASC gene belongs to the immunoglobulin gene family and again genes involved in the immune system might have unusually high mutation rates. PPP1R15A mediates apoptosis and hence has been suspected to play a role in cancer. PRTN3 is an autoantigen gene involved with Wegener granulomatosis, a severe disease whose pathophysiology is still not completely understood.

6.3 Discussion

Our recent algorithmic advances in optimal parsimony-based phylogeny inference have provided a new practical tool for examining the history of genomic variation data sets. We have applied this tool to a global examination of genetic variation in the human populations. The results provide several useful insights into the nature of phylogenetic imperfection in the human genome and its value as a statistic for detecting historical recombination and recurrent mutation.

Phylogenetic imperfection shows strong correlation with fine-scale recombination, a property that may make it useful as an alternative means of estimating recombination rates. The correlation drops significantly when recombination hotspots are removed. Moreover in either case, the correlation with recombination is far from perfect, which we believe results from the fact that imperfection reflects the influence of other sequence properties beyond local recombination rate, particularly local mutation rate. Phylogenetic imperfection in conjunction with other measures of recombination rate may be a useful way to separate these possibilities, allowing us to better identify both recombination hotspots and highly mutable sites or regions of the genome. This result may also in part reflect the fact that recombination rate and mutation rate are themselves correlated [44, 67]. It is possible that other mechanisms, such as gene conversion, significantly affect observed imperfection scores. Wiehe et al. [112] showed that one can distinguish recombination and gene conversion by local patterns of LD; a characteristic pattern of high LD between non-consecutive SNPs with mutual low LD to intervening SNPs might similarly allow one to distinguish genuine recurrent mutation from other possible sources of high phylogenetic imperfection.

Phylogenetic imperfection also exhibits several interesting properties when viewed at a genomic scale. First, imperfection shows a pattern of local correlation on multiple scales, from a strong peak for nearby but non-overlapping windows on the kilobase scale to a gradual decline in correlation on even megabase scales. It is likely that these correlations in part reflect the contribution of regional variation in recombination rate. It is also reasonable to assume that regional variability in mutation rates also plays a role. We cannot, however, yet determine the degree to which these two factors, or others unknown to us, might contribute to the overall regional variability. Because the information calculations excluded windows sharing SNPs, the very strong local peak on the kilobase scale could only derive from regions extremely dense in SNPs. It is therefore plausible that the fine-scale peak corresponds to local correlations in imperfection due to hypermutable regions of the genome. Further study of regional patterns for known sources of phylogenetic imperfection may help to separate these effects and detect any unanticipated contributing factors.

We do note that it might be beneficial to remove SNPs whose minor allele frequency is very low or to more uniformly select SNPs based on physical chromosome position. For instance, when we used HapMap Phase I data, SNPs on the HLA-B gene produced imperfections in excess of 18. However, several of these SNPs were missing in HapMap Phase II. Therefore, care should be taken while studying the ancestry for a specific gene or locus either by sampling SNPs or by extending the size of the window to encompass the whole region under study to more accurately understand its ancestry.

Part III

Detecting Population Substructure

Chapter 7

Pure Populations

In this chapter, we study SNP variation at the population-level typically referred to as *population substructure* or *population stratification*. Although the study is interesting in itself as it attempts to discover population specific SNP alleles, substructure plays an even greater role in the problem of discovering genetic basis of diseases. Studying the etiology of common complex disease, such as cancer or Parkinson's disease, is an important task in the search for better treatments and diagnosis tools for these diseases. A common practice towards this task is to perform an association study, in which the genetic variation of a set of cases (individuals carrying the disease) and a set of controls (background population) is compared, and large discrepancies between the two populations indicate an association of a specific locus with the studied phenotype.

There are different forms of genetic variations that can be studied in the context of association tests, the most common one is single nucleotide polymorphisms (SNPs). Due to decreasing cost of assaying SNPs, whole genome association studies, in which hundreds of thousands of SNPs are genotyped for thousands of individuals is becoming a common practice [11].

A significant discrepancy between the allele frequencies in the cases and the controls gives evidence for an association between the SNP and the phenotype, and therefore links the SNP to the disease. However, the validity of the results of an association study heavily depends on the statistical analysis performed. One of the main growing concerns is that population substructure may raise spurious discoveries. In association studies, the discrepancies in the SNP-allele frequencies between the cases and the controls are believed to imply an association of the SNP with the disease, but if the cases and controls were collected from two very different populations, this discrepancy may be explained by the difference between the two populations, and hence the SNP is not necessarily associated with the disease. Even subtle differences in the population structures of the cases and the controls may result in spurious associations. In particular, this problem is becoming more acute when large scale association studies are performed(see for e.g., [32, 69]).

There are many computer programs that try to cope with this problem, most notably the widely used software `STRUCTURE` [90] and a recently developed method `EIGENSTRAT` [89]. `STRUCTURE` uses a Markov Chain Monte Carlo (MCMC) approach to find population substructure of a given population using DNA variation data. `EIGENSTRAT` is based on princi-

pal component analysis (PCA). Mathematically, this problem can be seen as a clustering problem, in which the different clusters correspond to different populations. Such clustering problems have been studied under a variety of different theoretical frameworks that share close similarities. For instance the max-cut of a graph shares properties with the eigenvectors of the corresponding (adjacency or Laplacian) matrix and therefore Spectral methods or PCA, `STRUCTURE` and finding max-cuts of graphs share close mathematical relationships [12, 14, 76, 82].

`STRUCTURE` has been used extensively in genetic studies (cited more than 700 times), and it has been shown to find population substructure quite accurately in many examples. Even though `STRUCTURE` performs very well in terms of accuracy, it is quite inefficient, and it may take weeks to run over one whole genome data-set. Furthermore, even though `STRUCTURE` outputs a likelihood score which assists in interpreting the results, it is not clear whether this likelihood score can be used to determine whether there is actually a significant presence of population substructure. Finally, as the MCMC is inherently a heuristic approach, it is hard to know which parameters to set for the algorithm; in particular, as we show in this paper, there is no uniform set of parameters that performs well for all the data-sets.

In order to cope with these problems, we introduce a new graph based method for clustering populations. We concentrate in this paper on the clustering of two populations, although the method can be easily extended to multiple populations. Our technique is based on a simple paradigm. We define a distance between every pair of individuals, and we then search for a maximum cut in the graph induced by these distances. From that cut, we perform a local search that maximizes the likelihood of the data, similar to the criterion used in `STRUCTURE`. The main advantage of our method is that it is extremely efficient, and at the same time very accurate. Furthermore, since eventually the algorithm optimizes the same score function as that of `STRUCTURE`, it can be viewed as a fast method that finds a local optimum for this criterion.

It is important to note that the efficiency of our method allows us to measure the significance of the population substructure by running our algorithm on thousands of permutations of the data. For instance, we find that both our method and `STRUCTURE` find a population substructure in the YRI population, genotyped by the HapMap project [75]. On the other hand, after the permutation test, we observe that the p-value is 0.75, indicating that this partition is probably just an artifact. Since `STRUCTURE` is too slow to perform such a test, our method gives a rigorous alternative to the significance estimators of `STRUCTURE`.

We measured the performance of our method and compared it to `STRUCTURE` on the HapMap populations, as well as on simulated data. We find that our method is at least as accurate as `STRUCTURE`, and an order of magnitude more efficient. Furthermore, we find that the accuracy of `STRUCTURE` degrades when many SNPs are used (thousands), while the accuracy of our method consistently improves when the number of SNPs increases. We have also compared our method to `EIGENSTRAT`, a recent program that corrects for population stratification using the eigenvalues of the genotype covariance matrix [89]. In [89] they suggest a method based on principle component analysis, that assigns each individual a vector representing its ancestral composition. Although their method is not specifically

designed for clustering populations, we have adapted their method in a natural way and compared it to the method developed in this paper. We found that EIGENSTRAT is quite efficient, but it appears not to perform very well on many of our datasets. We believe that this is due to the fact that the principal component analysis fails when the sub-populations structures are not independent.

Technically, our method is based on a distance defined between pairs of individuals. There are many possible distance measures, and the resulting algorithm is very sensitive to the choice of the distance measure. Surprisingly, one of the most natural measures, i.e., the Hamming distance, performs quite poorly. We therefore use as a starting point the mother-father distance defined in [76]. This measure satisfies the property that the expected distance between two individuals drawn from the same sub-population is zero while the distance between individuals from two different sub-populations is positive. Furthermore, in [76] it is shown that the max-cut induces the correct partitioning asymptotically, at least when the sub-population sizes are equal. Our final distance uses a more complicated procedure which takes into account the genotypes of the whole population in order to determine the distance between a pair of individuals. We show empirically that this procedure is advantageous and that the resulting distance better represents the population structure. This distance measure may be of independent interest, as it may be used in other population based applications.

7.1 Problem and Algorithm

We consider the setting in which a set of n individuals are genotyped over m SNPs. The problem of population stratification focuses on the assignment of each of the individuals to a population cluster. In practice, an individual could belong to more than one cluster, (for instance when the individual's ancestors come from two or more different populations). In this paper, however we concentrate on the simpler case, in which each individual is assumed to belong to exactly one population. Furthermore, we assume that the number of populations K is known. We will observe later that this assumption is not too restrictive, as one can test for the validity of the solution. Our goal is to cluster the set of individuals into K clusters, based on their genotype information.

In order to define the problem mathematically, we first introduce a random generative model for the individuals' genotypes. Each genotype is represented by a vector $g \in \{0,1,2\}^m$, where g_j represents the minor allele in SNP j, that is, $g_j = 1$ for heterozygous, and it is 0 and 2 for the homozygous major or minor alleles respectively. A population is characterized by the minor allele frequency in each of the SNPs. Thus, a population i is defined by an m-dimensional vector $\vec{p^i} = (p^i_1, \ldots, p^i_m)$, where p^i_j represents the minor allele frequency of population i in position j. The random generative model assumes that all individuals are sampled independently, and that for each individual g, the different SNP values are sampled independently, where g_j is sampled from the distribution $\{(p^i_j)^2, 2p^i_j(1-p^i_j), (1-p^i_j)^2\}$ (e.g., the probability that $g_j = 1$ is $2p^i_j(1-p^i_j)$). This model has been used by previous approaches, and in particular by STRUCTURE. The assumption that the different SNPs are independent can be justified if the SNPs are physically distant

from each other (and thus, they are in *linkage equilibrium*). We define the distance between two sub-populations i, i' as:

$$d(i,i') = \sqrt{\sum_j (p_j^i - p_j^{i'})^2}$$

Formally, we assume that we get as an input an $n \times m$ genotype matrix A, where the rows $R(A)$ denote diploid individuals and the columns $C(A)$ represent SNP sites. Each entry in A is in $\{0,1,2\}$. We search for a classification $\theta : R(A) \to \{1,\ldots,K\}$, that assigns every individual to a particular sub-population. Let $\hat{\theta}$ be the correct classification. Our objective is to minimize the number of errors made by the algorithm, that is, we would like to minimize $|\{r \in R(A) \mid \theta(r) \neq \hat{\theta}(r)\}|$.

7.1.1 The Graph Based Approach

It is convenient to think of the above problem as a clustering problem in a graph. In this case, we construct a complete graph $G = (V, E)$, where vertex set V corresponds to the set of individuals, and edge set E is the set of all pairs of individuals. We assign a distance for each edge, which will intuitively represent the genomic distance between the two individuals. Then, the main idea of the algorithm is to find a max-K-cut in the resulting graph. This makes sense since G captures the fact that the genomic distance between two individuals from the same sub-population is small, while the distance between two individuals from different sub-populations may be large. Clearly, the resulting algorithm is sensitive to the choice of the distance measure.

The most natural distance measure is the Hamming distance, which counts the number of differences between the two vectors. However, we observe that in practice the Hamming distance does not provide very good results[1]. We therefore follow [76], and start from the so called *Mother-Father distance (MF)*. The MF-distance satisfies the property that the expected distance between two individuals from the same population group is 0 and the expected distance between two individuals of different populations is positive. Actually, it is not hard to see that the MF-distance is the only pair-wise distance measure that satisfies this property (up to a constant factor).

Formally, for any two individuals r_1, r_2, we define $\delta_j(r_1, r_2)$, the MF-distance at SNP j as follows. We set $\delta_j(r_1, r_2) = -1$ if $r_{1j} = r_{2j} = 1$, $\delta_j(r_1, r_2) = 2$ if $r_{1j} = 0, r_{2j} = 2$ or $r_{1j} = 2, r_{2j} = 0$, and 0 otherwise. We then define the MF-distance $\delta(r_1, r_2)$ to be the sum of the MF-distances over all SNPs. That is, $\delta(r_1, r_2) = \sum_j \delta_j(r_1, r_2)$.

We can now compute the expected distance between two individuals r_1, r_2 from popu-

[1]For example, when $p_j^1 = 2/3, p_j^2 = 1$, the expected distance within population 1 is larger than the expected distance across

lations i and i' $E[\delta(r_1, r_2)]$:

$$\begin{aligned} &= \sum_j E[\delta_j(r_1, r_2)] \\ &= \sum_j 2(p_j^i)^2(1-p_j^{i'})^2 + 2(p_j^{i'})^2(1-p_j^i)^2 - 2p_j^i(1-p_j^i)(2p_j^{i'}(1-p_j^{i'})) \\ &= 2\sum_j (p_j^i(1-p_j^{i'}) - p_j^{i'}(1-p_j^i))^2 = 2\sum_j (p_j^i - p_j^{i'})^2 = 2d(i,i')^2 \end{aligned}$$

Consequently, if $i \neq i'$, then the expected MF-distance between two individuals of different populations is positive. On the other hand, if $i = i'$, then $d(i, i') = 0$, and the expected MF-distance between two individuals of the same population is zero. It is further shown in [76], that if the distance between two different sub-populations $d(i, i') >> \sqrt{1.5}(m \log n)^{0.25}$, then with high probability, all the pair-wise distances within a sub-population are at most $d(i, i')^2$, while pair-wise distances across the two sub-populations are at least $d(i, i')^2$. In that case, the max-K-cut algorithm may be reduced to a connected component algorithm. Furthermore, it can be shown that even with much smaller separation of the two populations, the max-cut on the graph with MF-distances produces the correct cut [76].

7.1.2 Triplets-based distance

Even though the MF-distance has some very nice properties, our empirical studies (see Appendix 7.4) show that the max-cut solution obtained from this distance is sometimes biased towards an unbalanced partition. Intuitively, although the expected value of the MF-distance is monotone with the distance between the populations, unbalanced cuts may be chosen by the algorithm by pure chance. It is therefore essential to find a distance measure that has smaller variance than the MF-distance.

We build on top of MF-distances to obtain a more sensitive distance measure, which we call the triplet distance. The main idea of the triplet measure is to utilize information from all genotypes to determine the distance between a pair of individuals.

We will now formally define the triplet distance for a pair of individuals r_1 and r_2. The triplet distance depends on two parameters a, b that will be fixed later. For every third individual in the population, r, we consider the unordered set $\{r_1, r_2, r\}$, which we refer to as a *triplet*. For each such triplet, we define two indicator variables X_r and Y_r such that $X_r = 1$ if $\delta(r_1, r_2) \geq \max(\delta(r_1, r), \delta(r_2, r))$, and $X_r = 0$ otherwise. Similarly, $Y_r = 1$ if $\delta(r_1, r_2) \leq \min(\delta(r_1, r), \delta(r_2, r))$, and it is zero otherwise. We define the triplet-based distance as $d_{a,b}(r_1, r_2) = \sum_r (aX_r + bY_r)$. In other words, to compute the triplet distance of r_1 and r_2, we consider every third individual r and if $\delta(r_1, r_2)$ is the largest among the three MF-distances, then we add a and if it is the smallest, then we add b.

We now find the expected triplet distance $d_{a,b}(r_1, r_2)$ for a pair of individuals r_1, r_2. We will implicitly assume that all MF-distances are different (this is true if the number of SNPs is sufficiently large). For a triplet (r_1, r_2, r), we consider the following two cases. First, assume that all three individuals are from the same population. Then by symmetry,

$\Pr(X_r = 1) = \Pr(Y_r = 1) = \frac{1}{3}$. Otherwise, if r_1, r_2 are from population i, and r is from another sub-population i', we will bound the probability $\Pr[\delta(r_1, r_2) \geq \delta(r_1, r)]$. Intuitively, this probability should be small if the distance $d(i, i')$ is large enough. Formally, we know that

$$E[\delta(r_1, r_2) - \delta(r_1, r)] = -2d(i, i')^2.$$

Furthermore, $\delta(r_1, r_2) - \delta(r_1, r)$ is the sum of m random variables that lie in the interval $[-2, 2]$. This is because, if $\delta(r_1, r_2) = -1$ then $\delta(r_1, r) \neq 2$ and vice-versa. Therefore, we could use the following tail bound, known as Hoeffding bound[70]:

Theorem 7.1.1 *Let $X_1, \ldots, X_n$ be n independent random variables, and let a, b be such that for every i, $a \leq X_i \leq b$. Denote $X = X_1 + \ldots + X_n$. Then,*

$$\Pr(X - E[X] > \alpha) \leq \exp\left(\frac{-2\alpha^2}{n(b-a)^2}\right).$$

Thus, using the Hoeffding bound, we get $\Pr[\delta(r_1, r_2) - \delta(r_1, r) > 0]$

$$= \Pr[\delta(r_1, r_2) - \delta(r_1, r) + 2d(i, i')^2 > 2d(i, i')^2] \leq \exp\left(-\frac{d(i, i')^4}{2m}\right)$$

If $d(i, i') = (6tm \log n)^{0.25}$ for $t > 1$, we get by the union bound that with very high probability all triplets satisfy the property that edge distance within a sub-population is lesser than any edge distance across two populations. The probability that this event does not happen is smaller than $\frac{1}{n^{3t-2}}$. We now use these observations to compute the expected triplet distances. Assume that r_1, r_2 are from sub-population i, and that P_i is the frequency (prior) of this sub-population in the entire population. Then,

$$\begin{aligned} E[d_{a,b}(r_1, r_2)] &= E[a \sum_r X_r + b \sum_r Y_r] \\ &\leq n\left(a\frac{P_i}{3} + b(1 - \frac{2P_i}{3})\right) + \frac{|a| + |b|}{n^{3t-2}} \\ &\approx n\left(a\frac{P_i}{3} + b(1 - \frac{2P_i}{3})\right) \end{aligned}$$

Similarly, for r_1, r_2 from different sub-populations i, i', it is easy to see that

$$E[d_{a,b}(r_1, r_2)] \geq \frac{an}{2} - \frac{|a|+|b|}{n^{3t-2}} \approx \frac{an}{2}$$

If we know the frequency of the sub-populations in the entire population, then we can take $P = \max_i P_i$. For instance, if we set $a = (2/P) - 2, b = -1$ we get that the expected distance between individuals from two different populations is positive, while the expected distance between individuals of the same population is non-positive. For a balanced cut, selecting $a = 4, b = -1$, gives positive expected distance between individuals of different populations and zero otherwise. In practice, even though we do not know the correct value of P, we try different values of P to determine a, b, each giving different partitions.

We then pick the partition with the largest likelihood score, where the likelihood score is similar to the one used for STRUCTURE, as we now describe.

Recall that A is the input genotype matrix with $R(A)$ being the genotypes of the n input individuals, $\theta : R(A) \mapsto \{1, \ldots, K\}$ is the classification of individuals to sub-populations and $\vec{p^i}$ is an m-dimensional vector of the MAF of sub-population i. Given θ, the maximum likelihood estimate of $\vec{p_i}$ is obtained by simply counting the allele frequencies in each of the sub-populations defined by the partition. The posterior probability is given by

$$\Pr[\theta, \vec{p^i}|A] \propto \Pr[\theta] \Pr[\vec{p^i}] \Pr[A|\theta, \vec{p^i}]$$

We set the priors for θ and $\vec{p^i}$ to be fixed and uniform, and thus maximizing the posterior is equivalent to maximizing the likelihood $\mathcal{L}(A|\theta, \vec{p^i}) = \Pr[A|\theta, \vec{p^i}]$.

Algorithm (GRAPH-TRIPLETS). We can now describe the whole algorithm. The algorithm begins by computing the MF-distance for each pair of individuals. Then, for every pair of individuals r_1, r_2, we compute $X(r_1, r_2) = \sum_r X_r$, and $Y(r_1, r_2) = \sum_r Y_r$. The algorithm then proceeds in iterations. In each iteration we pick a value for P, and we search for a partition that maximizes the likelihood score, based on the prior information that one of the sub-populations is of size P. We take values of P ranging from 0.5 through 0.9 in 0.1 increments. Each such value determines the values of a and b. The triplet distances are then computed for each pair of individuals, by setting $d_{a,b}(r_1, r_2) = ((2/P) - 2)X(r_1, r_2) - Y(r_1, r_2)$. These distances induce a complete graph $G = (V, E)$, where the vertices represent individuals and the edges are weighted by the triplet distances. We are then interested in finding the maximum K-cut.

Unfortunately, finding the max-K-cut of the graph is an NP-hard problem even when $K = 2$ [56]. We therefore use the Kernighan-Lin heuristic [70], which is a hill-climbing method to find the optimal cut. The algorithm for the case when $K = 2$ is presented in Figure 7.1. The algorithm randomly partitions the vertices $V(G)$ into two disjoint sets V_1 and V_2. The algorithm then proceeds in *rounds* each of which involves performing $|V(G)|$ *iterations*. At each iteration we move a vertex u from one side of the cut to the other. The vertex u is chosen so that the resulting cut is maximized. Unlike standard local search techniques, the algorithm swaps u even if this results in the reduction of the cut-size. Once a vertex u is swapped, it cannot be swapped again until the next round. At the end of a round all vertices have been swapped, and the best partition in that round is chosen for the next round. We repeat until the cuts in the beginning and the end of a round are identical, and thus no improvement can be achieved. In Figure 7.1, set $V_x(V_{x'})$ denotes the vertex set of V_1, V_2 that currently contains x (does not contain x) and $\text{cut}(V_1, V_2)$ denotes the cost of the cut.

Using Kernighan-Lin, for each setting of a, b we find a max-K-cut and we select the one that maximizes $\mathcal{L}(A|\theta, \vec{p^i})$. Finally, we perform a greedy local search by moving a vertex from one side to another, if it improves the likelihood. This final step, typically improves the accuracy by a little in practice.

kernighanLin(graph G)

1. randomly partition $V(G)$ into V_1, V_2
2. $\alpha \leftarrow 1$
3. `while` $\alpha = 1$ `do` `(* rounds *)`
   (a) $\alpha \leftarrow 0, \chi \leftarrow V_1 \cup V_2$
   (b) `while` $|\chi| > 0$ `do` `(* iterations *)`
      i. $u \leftarrow \text{argmax}_{x \in \chi} \text{cut}(V_x \setminus \{x\}, V_{x'} \cup \{x\})$
      ii. $\chi \leftarrow \chi \setminus \{u\}$
      iii. `if` $u \in V_1$ `then` $V_1 \leftarrow V_1 \setminus \{u\}, V_2 \leftarrow V_2 \cup \{u\}$
      iv. `else` $V_1 \leftarrow V_1 \cup \{u\}, V_2 \leftarrow V_2 \setminus \{u\}$
      v. `if` $\text{cut}(V_1, V_2) > \text{cut}(V_1^*, V_2^*)$ `then` $V_1^* \leftarrow V_1, V_2^* \leftarrow V_2, \alpha \leftarrow 1$
   (c) $V_1 \leftarrow V_1^*, V_2 \leftarrow V_2^*$

Figure 7.1: Kernighan-Lin heuristic to find max-cut of graph G.

7.2 Results

To evaluate the performance of our method, we compared GRAPH-TRIPLETS to two state of the art methods that deal with population stratification, namely STRUCTURE and EIGENSTRAT, which are described below.

STRUCTURE [90] is a well established package that uses Markov Chain Monte Carlo method (MCMC) to heuristically maximize the posterior probability. Structure can be seeded with a number of different parameters. We used $K = 2$, to denote that the program should look for two sub-populations. By default, the program assumes that the allele frequencies of the two populations are independent. For closely related sub-populations, however, the software allows for a mode in which the frequencies are assumed to be correlated. We ran the program on both modes, with the parameter turned off and on. The default number of *MCMC* and *Burnin* iterations is 2000 each. We varied this number to analyze the trade-off between run-time and accuracy. We used the default values for the rest of the parameters.

We note that STRUCTURE is a software that does much more than just clustering individuals. Among other things, it can cope with admixed populations, and it can incorporate linkage disequilibrium into its model. We have not compared our method to these modes of STRUCTURE, as it is beyond the scope of this paper, and our algorithm is not optimized for such tasks at this point.

EIGENSTRAT [89] is a relatively new software tool, which corrects population substructure by the spectral properties of the covariance genotype matrix. In a nutshell, EIGENSTRAT takes A an $m \times n$ input genotype matrix, where rows are SNPs and columns are individuals and normalizes each entry of A by subtracting the row mean (minor allele

frequency of the SNP) and dividing by the row's standard deviation. It then takes the largest eigenvectors of the covariance $n \times n$ matrix Ψ, and uses those to correct for population sub-structure. Even though `EIGENSTRAT` is not explicitly described as a genetic clustering method, we adapt their algorithm in a natural way, resulting in a clustering algorithm in which the clusters are determined by using the sign of the entries of the highest eigenvector of Ψ. We implemented this clustering algorithm in MatLab and compared it to our method. We refer to our implementation as `Spectral` in the results presented.

Datasets. For the evaluation, we used datasets from two different sources. First, we used simulated data generated using the following model. Each sub-population i is represented by an m-dimensional vector of allele frequencies $\vec{p^i}$, and an individual of the population is sampled by randomly and independently picking allele counts according to the allele frequency distributions of the sub-population. For simulations, we assumed that all SNPs within a sub-population had the same allele frequency, i.e., for any $i, p^i_j = p^i_{j'}$.

We have also used the publicly available data from the International HapMap consortium [75]. This data-set consists of four population groups: Utah residents with ancestry from northern and western Europe(CEU), Yoruba in Ibadan, Nigeria (YRI), Han Chinese in Beijing, China (CHB) and Japanese in Tokyo, Japan (JPT) with 90, 90, 45 and 44 individuals respectively. The Central Europeans and Yoruba Africans consisted of thirty trios each, and therefore in order to avoid these dependencies, we used the 60 parents from each of the two populations, ignoring the 30 children. To obtain a test set where the SNPs are independent, we sampled m SNPs uniformly at random from chromosome 10. We evaluated the programs on each of the six pairs of populations, with different numbers of SNPs, ranging from 1000 to 8000.

Evaluation Measures. There are many possible ways to evaluate the performance of the algorithms. We chose to let each of the program separate the genotypes of two populations (say Africans and Chinese in the HapMap data) into two clusters, and the error rate of such an experiment would be the number of individuals misclassified (for instance, the number of Africans classified as Chinese). We have also compared the running-time of the methods. In summary, our experiments show that the graph-based method is significantly faster while being at least as accurate as existing methods.

Simulated Data. On simulated data, we studied closely related populations. We fixed the minor allele frequency of one population to be 0.1 for each of its SNPs, while for the other population the minor allele frequency varied from 0.12 to 0.19. On all the data presented, `STRUCTURE` running on default parameters fails to find two sub-populations, i.e., it returns solution with all individuals in one cluster. We therefore only report `STRUCTURE` running for 2000 iterations with the correlation mode turned on. Figure 7.3(a) shows that `GRAPH-TRIPLETS` is an order of magnitude faster than `STRUCTURE`. Figure 7.2 shows that it makes substantially less errors overall than `STRUCTURE` and the spectral clustering which is based on `EIGENSTRAT`. It is conceivable that `STRUCTURE` would find better solutions if it

uses even more time, although this seems rather prohibitive. We note that the performance of STRUCTURE seems not to be monotonic with the number of SNPs used.

HapMap. For the HapMap datasets, we considered all six pairs of populations. For most pairs, all methods worked perfectly (no errors were made) with as few as 200 SNPs. The only hard instance was the Chinese-Japanese pair. For this pair, none of the methods could give a perfect clustering prediction even when 8000 SNPs were used. As can be seen from Figure 7.4, the spectral method and GRAPH-TRIPLETS seem to produce lesser errors than the classification returned by STRUCTURE. Furthermore, we used the two different modes of STRUCTURE (with or without correlations), and the results were inconclusive regarding which one works better on this dataset, as when small number of SNPs were used (1000, 2000 or 4000), the correlation mode seemed to perform better. When more SNPs were used (8000 SNPs, 2000 or 4000 iters), the 'no correlation' mode of STRUCTURE was better.

As before, Figure 7.3(b) demonstrates that GRAPH-TRIPLETS is much more efficient than STRUCTURE. Specifically, the run time for GRAPH-TRIPLETS $\approx$ 300 times faster for 1000 SNPs and $\approx$ 1000 times faster for 8000 SNPs.

7.3 Significance of Clusters

An obvious and important question to consider is whether the clusters obtained from the methods are significant. In practice, we could run STRUCTURE or GRAPH-TRIPLETS with K, the number of sub-populations set to 2. When the software returns a solution, with individuals divided into two populations, there is no guarantee on whether the input set of taxa actually even contains two sub-populations. To test the significance of the clusters, one could perform the statistical tests described by Pritchard et al. [90], which as the authors themselves point out either uses dubious assumptions or does not work for large number of SNPs in practice. Alternatively, we could examine the change in the likelihood function or use information based evaluation such as minimum description length to decide on whether there is truly two sub-populations or not.

A simple and direct approach is to permute the alleles in each site independently such that the input no longer has sub-structure. We can then re-run the algorithm on the new input. The p-value is simply the fraction of times, the permuted input had solution larger (or more likely) than that of the original input.

However, such tests can only be performed if the algorithm to find clusters is very efficient. Here, we simply considered a randomly drawn data set with 8000 SNPs from each of the HapMap populations. We ran structure (default parameters) and triplets with $K = 2$. We report the number of errors made (size of the smaller set) along with the p-value which can be efficiently computed using the triplets method. We believe that the computation of this p-value is a great benefit in practice that our new technique can offer. The results for 1000 permutations is presented in Table 7.1.

	Errors		
	`structure`	Triplets	Triplets p-value
Central Europeans, CEU	21	2	0.01
Yoruba Africans, YRI	26	20	0.75
Chinese, CHB	17	16	0.267
Japanese, JPT	18	12	0.929

Table 7.1: Triplets can be used to compute p-values directly.

7.4 Empirical Comparison of MF with Triplets.

To motivate our choice of triplet based distance instead of MF-distance, we show empirically that the triplet distance gives a better separation of the two populations into two clusters. In order to do so, we randomly generate two sub-populations according to the following model. We generate the two populations, such that one population has minor allele p_1 for all the SNPs, and the other population has minor allele p_2 for all the SNPs. Furthermore, each of these sub-populations has the same number of individuals. We measure the cut distance of the correct cut using the MF-distance and the triplet distance. Let d_{mf} and d_t be the correct cut weight for mother-father and triplets. We then find $10,000$ random balanced cuts of the graph and measure the cut weights. We then measure the fraction of times the random cut cost was larger than d_{mf} or d_t respectively. We call this measure the *balanced p-value*. We also measured the max-cut for $10,000$ randomly generated unbalanced cuts, where 0.25-fraction of the vertices were on one side. We call this measure the *unbalanced p-value*. .

The results of the various simulation tests are shown in Figure 7.5. Figure 7.5(a), shows that the balanced p-values are similar for triplets based distance and for the MF-distance, and that the balanced p-value of both methods quickly go down to zero. More importantly, Figure 7.5(b) shows that triplets clearly has a lower p-value in the case of unbalanced cuts. While MF-distance contains several unbalanced cuts of high weight, on the triplets, the correct cut has cost typically higher than all other cuts. This provides an evidence that the triplet distance is advantageous.

7.5 Conclusions

The problem of population stratification is an increasing concern in the context of disease association studies. In particular, its influence on whole genome association studies (for e.g. [32, 69]) are severe. Even though existing methods for clustering individuals based on their SNPs provide relatively accurate predictions, there is no rigorous theory that ensures the convergence of these methods to the correct solution. Furthermore, there is no study that compares these methods on a variety of datasets, both real and simulated. Our paper has been motivated by these two concerns.

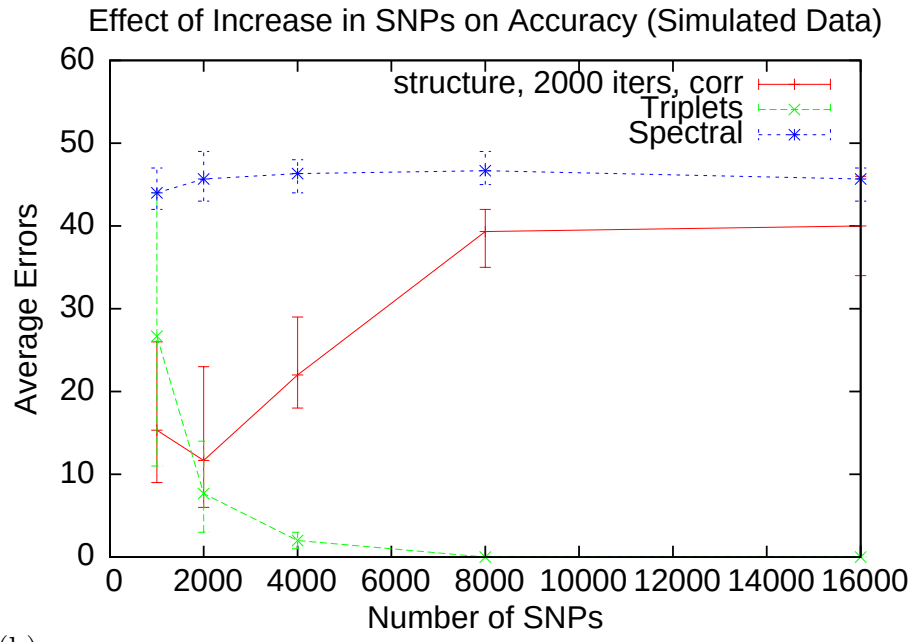
In this paper, we suggest a graph based method for detecting population stratification. The distance measure used in our method builds on the Mother-Father distance that was

suggested in [76], where it has been rigorously and analytically shown that if the sample size is large enough, the measure will represent the correct distance between individuals, and therefore our algorithms will converge to the true population clusters. We believe that this theoretical foundation for our algorithm is an important advantage that proves itself in practice. In particular, we show that our algorithm is consistently at least as accurate as `STRUCTURE` and `EIGENSTRAT`.

One of the questions we raised in this paper is the validity of the population substructure found by the clustering algorithm. We have demonstrated that the different methods will tend to cluster the individuals in two clusters, even if in reality there is only one population. In order to assess the significance of these partitions, we suggest a permutation test, which seems to give the correct significance scores on the HapMap populations. This permutation test can only be carried out if the clustering methods are highly efficient. As our method runs in seconds over thousands of SNPs and hundreds of individuals, this was feasible.

We note that our paper focuses on the clustering of two populations while `STRUCTURE` and `EIGENSTRAT` can do much more than that. In particular, they can deal with admixed populations, correlated populations, and linkage disequilibrium. We hope that a combination of the existing methods such as `STRUCTURE` and `EIGENSTRAT`, together with our graph based approach may lead to improved tools for these cases as well.

(a)



(b)

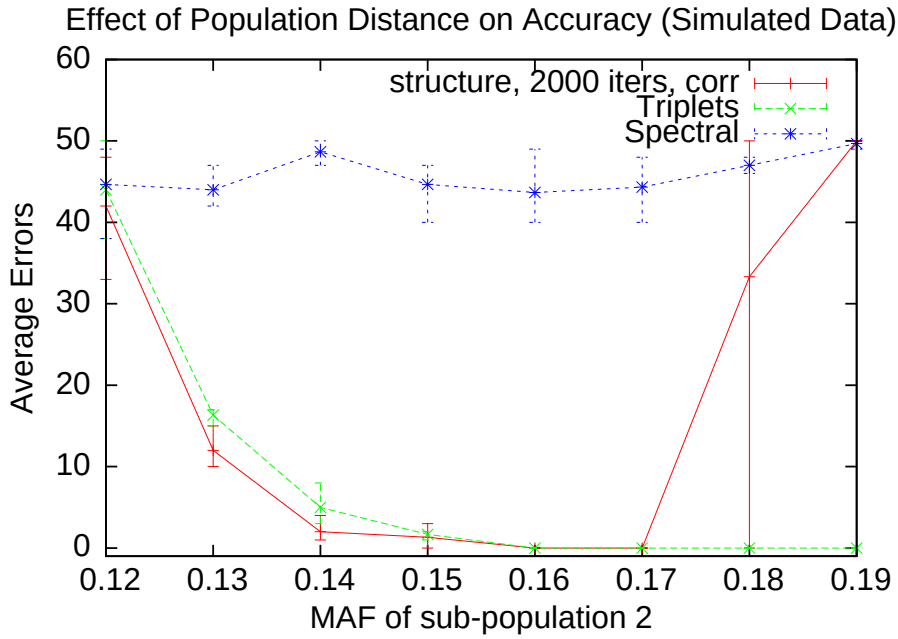
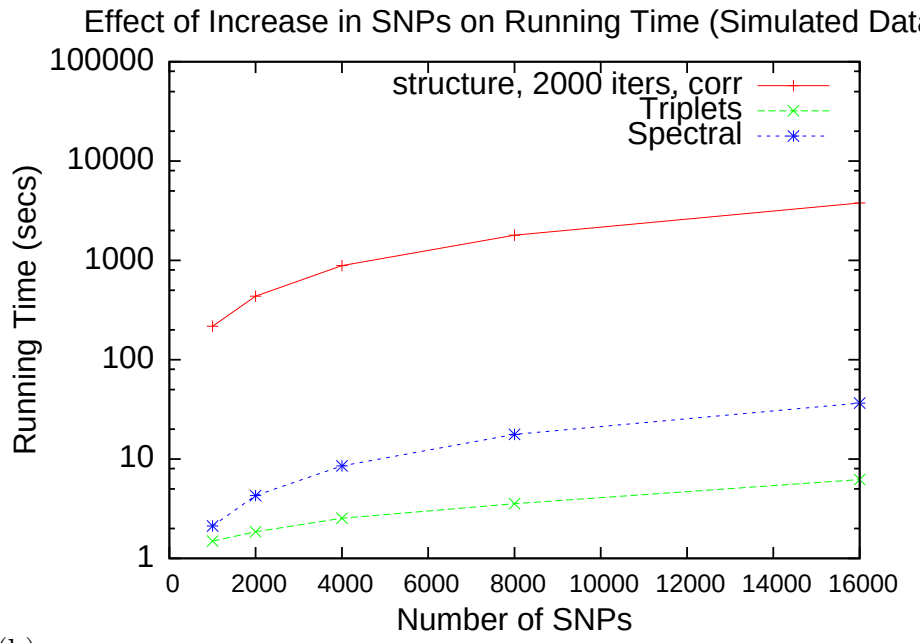


Figure 7.2: Comparison of accuracy on simulated data. `GRAPH-TRIPLETS` is consistent in its accuracy and converges to the correct partition with increase in SNPs or increase in distance. On (a) MAF of sub-populations 1 and 2 were 0.1 and 0.13 respectively. On (b) the number of SNPs was fixed to 1000 and MAF of sub-population 1 was fixed at 0.1. We used an average of three randomly generated data sets to obtain every point. Error bars indicate highest and lowest values obtained.

(a)



(b)

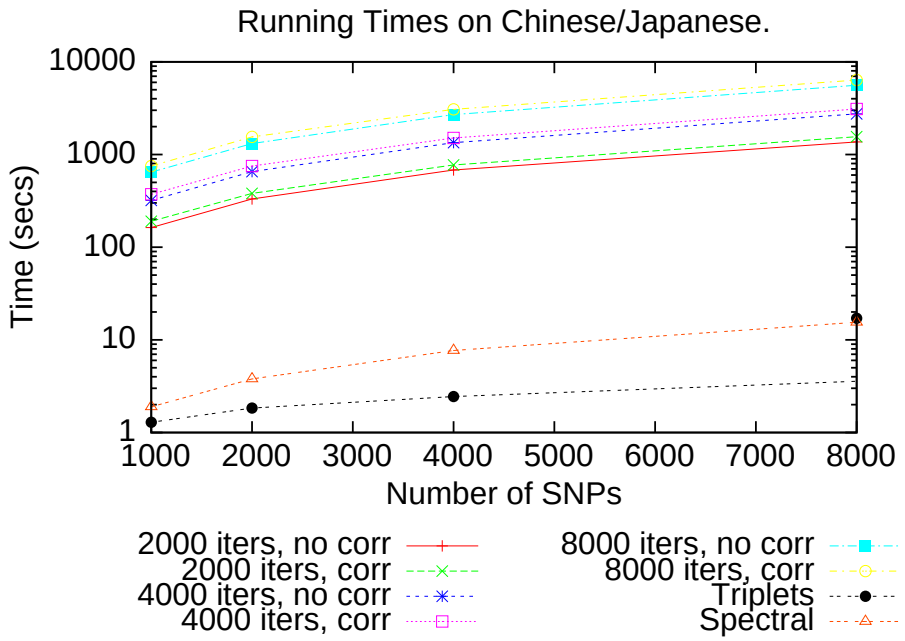
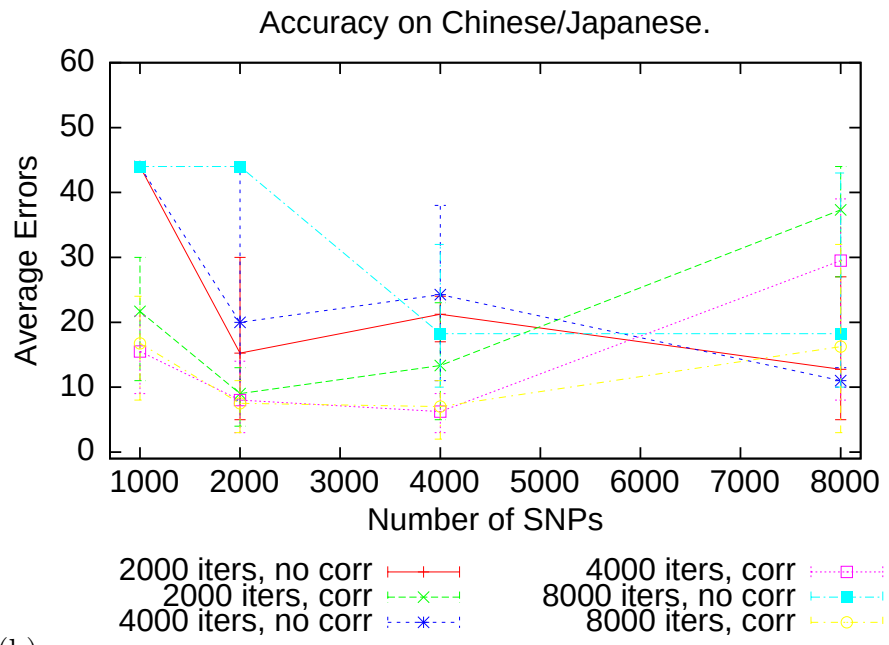


Figure 7.3: Comparison of run-times on simulated and real data. `GRAPH-TRIPLETS` is hundreds of times faster than `STRUCTURE`.

(a)



(b)

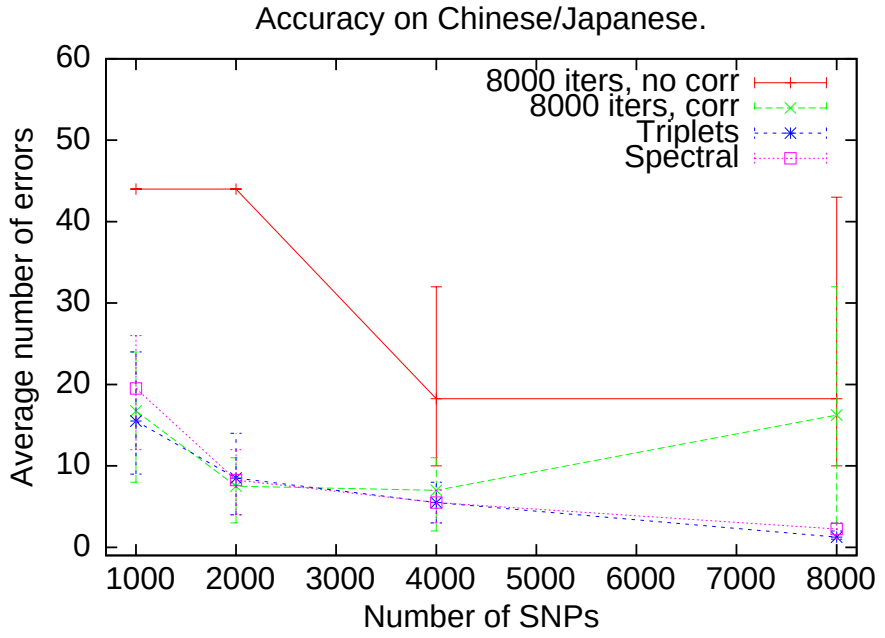
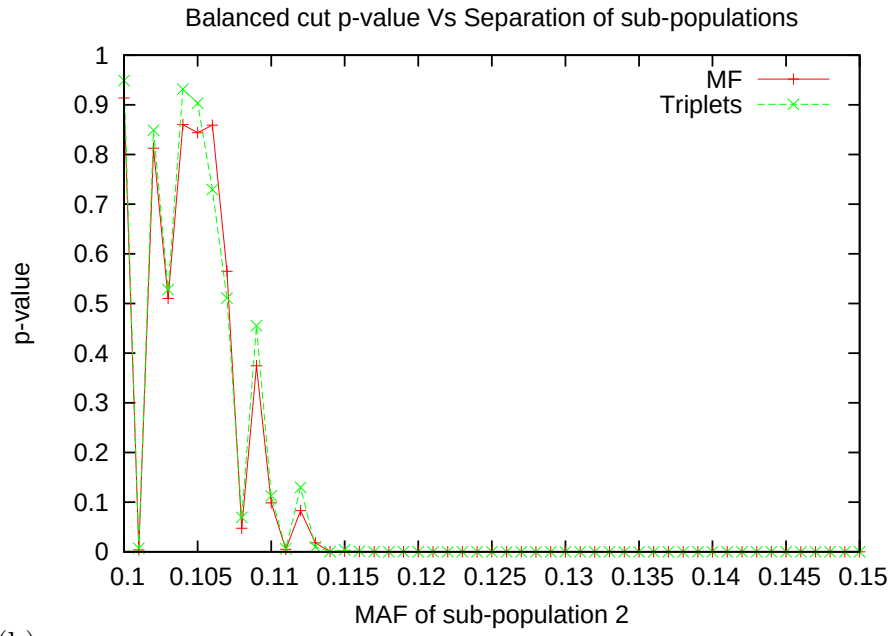


Figure 7.4: Comparison of accuracy on HapMap Chinese/Japanese. `GRAPH-TRIPLETS` is accurate and converges to the correct partition with increase in SNPs. It is unclear what parameters of `STRUCTURE` to use. We used the average of four randomly drawn data sets to obtain every point. Error bars indicate the highest and lowest values obtained.

(a)



(b)

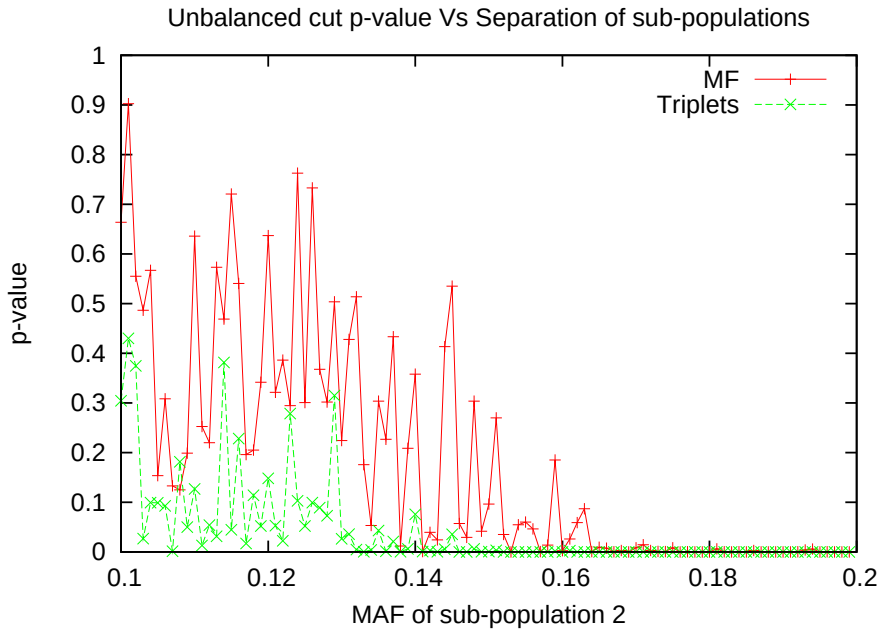


Figure 7.5: Comparison of MF and Triplet distance measures. Figure (a) shows that when only balanced cuts are considered, the distance measures provide comparable p-values. Figure (b) shows that more often unbalanced cuts in the MF distance contains cost larger than that of the correct cut. In both figures MAF of sub-population 1 was fixed to 0.1. For Figure (a) we used 100 individuals in each population and for Figure (b) we used 10 individuals in each population. We used parameters $a = 2, b = -1$ for the triplets.

Chapter 8

Admixed Populations

In this chapter we continue to develop algorithms to detect population substructure. Unlike the previous chapter, here we focus on the case where the underlying individuals may be admixed. The application, however, remains the same – we wish to detect substructure if it exists, so that disease association testing can be accurate.

The problem of inferring the population sub-structure is especially challenging when recently admixed populations are involved. In these populations (e.g., African Americans and Latinos), two or more ancestral populations have been mixing for a relatively small number of generations, resulting in a new population in which the ancestry of every individual can be explained by different proportions of the original populations. Due to recombination events, even within the DNA of a single individual, different regions of the genome may originate from different ancestral populations. This adds to the complexity of the problem of finding the ancestral information of an individual, since in non-admixed populations the whole genome can be used as an evidence for the population membership of an individual, while in the admixed case the genome of each individual is fragmented into shorter regions of different ancestry. It is therefore challenging to find the ancestral information of these individuals, and in particular, to find the locus-specific ancestries.

An accurate inference of locus-specific ancestry in admixed populations may lead to improved analysis of studies based on admixture mapping. In these studies, a set of cases from a recently admixed population is genotyped, and the genome is scanned for regions in which the proportions of ancestral populations are significantly different than the rest of the genome [34, 47]. Unfortunately, most of the current methods for inference of locus-specific ancestral information [43, 49, 71, 90] do not scale to large data sets. The only existing method that copes with large data sets is `SABER` [114], which is based on an extension of a Hidden Markov Model [92] to deal with local haplotype blocks.

Here, we propose a new method, `LAMP` (Local Ancestry in adMixed Populations), for *de-novo* estimation of the locus-specific ancestry in recently admixed populations (see Figure 8.1). Our method is based on the observation that previous methods that use a Hidden Markov Model or extensions of it, are set to infer a very large set of parameters, including the exact position of the recombination events, which makes the search over the parameter space infeasible. Instead, our method operates on sliding windows of contiguous SNPs. We first calculate an optimal window length. Next, we use a clustering algorithm that

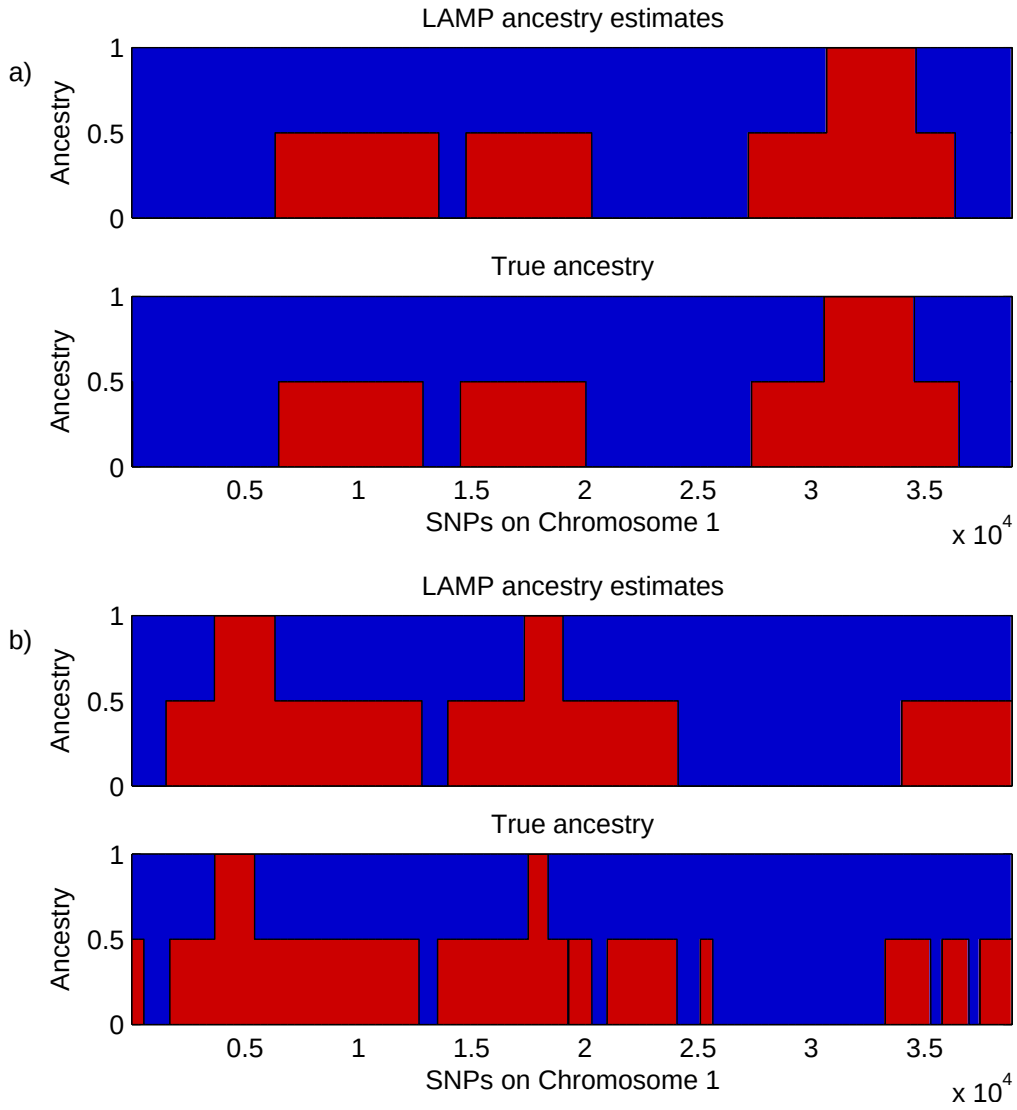


Figure 8.1: Two individuals in an admixed population. Ancestries predicted by `LAMP`(top panel) and true ancestries (bottom panel) are shown for each individual. As shown in the Figure, the ancestries (represented by red and blue) vary across the genome, and `LAMP` performs well in inferring the ancestry at each location.

operates on these windows and estimates each individual's ancestry. We then use a majority vote for each SNP, over all windows that overlap with the SNP, in order to decide the most likely ancestral populations at the SNP. This simple approach has two advantages over previous ones. First, we show analytically that the estimates of the algorithm are asymptotically correct across the entire genome. Second, it optimizes fewer parameters than previous methods and hence, the optimization is much faster and more robust than previous methods.

We tested `LAMP` extensively on various data sets of admixed populations generated from the HapMap resource. Our simulations show that `LAMP` is significantly more accurate than the state of the art methods such as `SABER` and `STRUCTURE`. In addition, `LAMP` is highly efficient with a running time that is about 200 times faster than `SABER` and about 10^4 times faster than `STRUCTURE`. The efficiency of `LAMP` allows us to estimate ancestries across the genome in several hours on a single computer.

An additional advantage of `LAMP` is that unlike previous methods such as `SABER`, it does not require the ancestral genotypes to infer the locus-specific ancestries (though it can take advantage of these if available). This may be crucial when the ancestral genotypes cannot be typed or are unknown. For instance, if one studies the population genetics of populations in remote geographic locations where historical admixing has not been recorded, a method such as `LAMP` could be used to reveal such recent admixing. Furthermore, even in cases where the history of admixing is known, it is not always possible to genotype all the ancestral populations, since some of the subpopulations have become extinct and some have entirely mixed with other populations. On the other hand, as genotypes of major population groups become available, it would be beneficial to use `LAMP-ANC` which can take advantage of the pure genotypes.

Surprisingly, we find that in many cases where `LAMP` does not receive the genotypes of the ancestral populations as input, it performs considerably better than `SABER`. In particular, on a simulated dataset of African-Americans, when measuring the percentage of individuals that are predicted with an accuracy of at least 90%, LAMP achieves high accuracies on 90% of the individuals while SABER and STRUCTURE achieve less than 10%.

Finally, we used `LAMP` to estimate the individual admixture, and showed empirically that this results in much more accurate estimates than methods such as `STRUCTURE`[90] or `EIGENSTRAT`[89]. This reduction in errors may be used to considerably reduce the rate of spurious association results in disease association studies.

8.1 Problem and Algorithm

The inference of locus specific ancestry depends on the mathematical model representing the mixing process of the populations. We will first describe the model assumptions, and then describe the inference algorithm under the model.

8.1.1 Model assumptions

We assume that there are K ancestral populations $A_1, \ldots, A_K$ that have been mixing for g generations. If the populations have mixed at different times, then g is taken to be an upper bound on the number of generations since the beginning of admixture. The fraction of population A_i in the ancestral population which we call the *admixture fraction* is α_i, where $\sum_i \alpha_i = 1$. We assume for convenience that $\alpha_1 \geq \alpha_2 \ldots \geq \alpha_K$. In each generation, we assume random mating within the combined pool of the k populations. We denote the recombination rate at position j by r_j. Note that r_j is the recombination rate at position j at a specific meiosis (one generation), and not through history. We model the transmission of a chromosome from a parent to a child by walking along the chromosome from the 5' end to the 3' end with crossovers between chromosomes occurring as a Poisson process with rate r_j [64]. For simplicity of the presentation, we will assume a uniform recombination rate, i.e., that $r = r_j$ for every position j. The algorithm and analysis remain qualitatively the same when applied to non-uniform recombination rates.

We denote the genotype data of individual i at position j as g_{ij}, where $g_{ij} \in \{0, 1, 2\}$ is the minor allele count at that position. At position j, the two alleles of individual i have descended from one or two of the K ancestral populations. We denote by $a_{ij}^p \in \{0, 0.5, 1\}$ the fraction of alleles descended from population p at position j in individual i. The quantities a_{ij}^p are unknown; the objective of this paper is to present a method `LAMP` that accurately estimates these quantities.

8.1.2 The `LAMP` framework

In this work we consider a recently admixed population in which the number of generations g since the beginning of the mixing is small. Therefore, we expect the total number of recombinations in these g generations to be small as well. The resulting chromosomes are mosaics of the k populations, where the *ancestral breakpoints* in which the chromosome ancestry changes from one population to the other are determined by the recombination events.

We assume that the quantities g, α_i, and r are known for the admixed population. The basic idea in `LAMP` is to estimate the ancestries of the individuals in a sliding window that spans l sites. We term l the length of the window. The choice of the length l will be discussed later. Intuitively, if l is small enough, and the number of generations g is not too large, a typical window of length l will have almost no recombination events throughout history, and therefore almost no breakpoints. Therefore, within each window, it is reasonable to use an inference algorithm that assigns the sequence of genotypes in the window to one or two of the populations under the assumption that there are no breakpoints in any of the chromosomes. The latter is a simple clustering problem, although the accuracy of the inference in a given window improves when the number of SNPs l in the window increases. We therefore search for a window length l, which is short enough so that most individuals have no breakpoints and large enough so that there is enough information to correctly cluster the individuals within the window. This procedure is repeated by sliding the window to cover all the SNPs on the genome. The windows that overlap a SNP are

then combined into a single solution using a majority vote for the ancestry assignment. We note that unlike previous methods (e.g. SABER [114], or STRUCTURE [90]), we are not attempting to estimate the exact positions of the breakpoints, but instead we are trying to minimize the errors in the locus-specific ancestry prediction across the genome.

The LAMP algorithm works as follows. We first find the optimal window length based on the parameters g, α_i, and r. Then, we use a clustering algorithm that operates on a window and estimates for each individual i, and for each ancestral populations A_j, A_k, the probability p^i_{jk} for individual i to have one chromosome descended from population A_j at this window and another descended from population A_k. We then use a majority vote for each SNP, over all windows that overlap with the SNP, in order to decide the most likely ancestral populations at the SNP. As we argue below, even though this scheme optimizes less parameters than previous methods, such as SABER, or a regular HMM, we show analytically and empirically that the estimates of the algorithm are asymptotically correct across the entire genome.

8.1.3 Estimating the ancestry in a single window

We assume that none of the individuals have a breakpoint within a window and estimate a single ancestral origin for each individual across the length of the window. This assumption is largely true if the length of the window is determined correctly (see Section 8.1.4 and Section 8.4 in the supplementary materials). We further assume that the values $\alpha_1, \ldots, \alpha_K$ are known. These values are the admixture fractions of each of the populations across the whole genome, and they can be estimated using existing tools such as STRUCTURE[90]. In the results section we show that our method is robust to reasonable inaccuracies in the estimates of $\alpha_1, \ldots, \alpha_K$.

Clustering Algorithm

We assume that sub-population A_i has minor allele frequencies $\vec{f_i} = f_{i1}, \ldots, f_{in}$ for n SNPs in a given window of length l, and that the different SNPs in the window are independent. The latter assumption can be achieved in practice by greedily removing SNPs having a high correlation coefficient ($r^2 > 0.1$) from the window. We look for a classification function $\theta : I \to \{1, \ldots, K\}^2$, where I is the set of individuals, and the range corresponds to the possible pairs of sub-populations. In particular, we write $\theta(i) = (\theta_1(i), \theta_2(i))$ to denote the ancestries of the two chromosomes of individual i in the current window. We use a clustering algorithm known as Iterated Conditional Modes (ICM) [8] to find an optimal classification of each individual in terms of the likelihood. For increased efficiency in the running time, we seed the algorithm with an initial classification as described in Section 8.1.3.

The updates in the ICM algorithm differ from those in a traditional EM method only in the E-step. In the latter, the E-step consists of obtaining the expected classification θ, given the values $\vec{f_i}$. This would provide fractional class membership for each individual i. However, since we assume that the initial classification provides a reasonable solution, we find the maximum aposteriori estimate of θ as shown below. For brevity we use $\mathcal{G}_i$ to refer

to the genotype $(g_{i1}, \ldots, g_{in})$ of the individual i.

$$\begin{aligned}\hat{\theta}(i) &= \text{argmax}_{A_sA_t \in \{1,\ldots,K\}^2} \Pr[\theta(i) = A_sA_t \mid \vec{f_1}, \ldots, \vec{f_K}, \mathcal{G}_i] \quad (8.1)\\ &= \text{argmax}_{A_sA_t \in \{1,\ldots,K\}^2} \Pr[\mathcal{G}_i \mid \vec{f_1}, \ldots, \vec{f_K}, \theta(i) = A_sA_t] \cdot \Pr[\theta(i) = A_sA_t \mid \vec{f_1}, \ldots, \vec{f_K}]\end{aligned}$$

Since $\alpha_1, \ldots, \alpha_K$ are known, under the assumption of random mating, we can estimate the first term $\Pr[\theta(i) = A_sA_t \mid \vec{f_1}, \ldots, \vec{f_K}]$ as $\Pr[\theta(i) = A_sA_t] = 2^{1-\delta(s,t)}\alpha_s\alpha_t$ where $\delta(x, y)$ is 1 iff $x = y$ and 0 otherwise.

The other term can be estimated as:

$$\begin{aligned}&\Pr[\mathcal{G}_i|\vec{f_1}, \ldots, \vec{f_K}, \theta(i) = A_sA_t]\\ &= \textstyle\prod_{g_{ij} \in \mathcal{G}_i | g_{ij}=2} f_{sj}f_{tj} \cdot \prod_{g_{ij} \in \mathcal{G}_i | g_{ij}=0} [(1 - f_{sj})(1 - f_{tj})] \cdot \prod_{g_{ij} \in \mathcal{G}_i | g_{ij}=1} [f_{sj}(1 - f_{tj}) + f_{tj}(1 - f_{sj})]\end{aligned}$$

In the M-step, we obtain the maximum likelihood estimate of $\vec{f_1}, \ldots, \vec{f_K}$ by finding

$$\text{argmax}_{\vec{f_1},\ldots,\vec{f_K}} \Pr[(\mathcal{G}_i)_{i=1}^m \mid \vec{f_1}, \ldots, \vec{f_K}, \theta] = \prod_{i=1}^{m} \Pr[\mathcal{G}_i \mid \vec{f_1}, \ldots, \vec{f_K}, \theta(i)] \quad (8.2)$$

If the phase of the individuals is known, the maximum likelihood estimate of $\vec{f_1}, \ldots, \vec{f_K}$ could have been computed by simply counting the number of alleles in each of the sub-populations at every position. However, when the phase is not known, the problem becomes more complicated. Consider for instance a heterozygous site j in an individual i, with $\theta_1(i) \neq \theta_2(i)$. In this case, it is not clear whether the minor allele count should be added to $f_{\theta_1(i)j}$ or to $f_{\theta_2(i)j}$. To solve this problem, we introduce another classification function per site, $\vec{\lambda_j} : I \to \{0, 1\}^K$. This function is defined on the set of SNPs for which the assignment of counts is ambiguous *i.e.*, heterozygous SNPs in individuals i with classification $\theta_1(i) \neq \theta_2(i)$. We denote this set of heterozygous SNPs $\mathcal{H}_i$. The function $\vec{\lambda}_j$ is defined as

$$\vec{\lambda}_j(i) = \begin{cases} \vec{e}_s, \text{if } j \in \mathcal{H}_i, \text{ one of } (\theta_1(i), \theta_2(i)) = s, \text{ and the minor allele is counted to } f_{sj} \\ \text{not defined for } j \notin \mathcal{H}_i \end{cases}$$

Here $\vec{e}_s$ is the vector with 1 in coordinate s and 0 elsewhere.

For a heterozygous site j in individual i such that $j \in \mathcal{H}_i$, we can now define

$$\begin{aligned}\Pr[\vec{\lambda}_j(i) = \vec{e}_{\theta_1(i)} \mid f_{1j}, \ldots, f_{Kj}, \theta(i)] &= f_{\theta_1(i)j}(1 - f_{\theta_2(i)j})\\ \Pr[\vec{\lambda}_j(i) = \vec{e}_{\theta_2(i)} \mid f_{1j}, \ldots, f_{Kj}, \theta(i)] &= f_{\theta_2(i)j}(1 - f_{\theta_1(i)j})\\ \Pr[\vec{\lambda}_j(i) = \vec{e}_{s \notin \{\theta_1(i), \theta_2(i)\}} \mid f_{1j}, \ldots, f_{Kj}, \theta(i)] &= 0\end{aligned}$$

Using the assumption of independence of the SNPs and the $\vec{\lambda}_j$ just defined, we can rewrite Equation 8.2 as follows. The usefulness of this will be apparent later.

$$\begin{aligned}(\hat{f}_{1j}, \ldots, \hat{f}_{Kj}) = \text{argmax}_{f_{1j},\ldots,f_{Kj}} &\prod_{i=1}^{m} \left(\prod_{j \in \mathcal{H}_i} \sum_{u=1}^{K} \Pr[\vec{\lambda}_j(i) = \vec{e}_u \mid f_{1j}, \ldots, f_{Kj}, \theta(i)] \right)\\ &\times \left(\prod_{j \in \{1,\ldots,n\} \ \mathcal{H}_i} \Pr[g_{ij} \mid f_{1j}, \ldots, f_{Kj}, \theta(i)] \right) \quad (8.3)\end{aligned}$$

The MLE for $\hat{f_{ij}}, \ldots, \hat{f_{Kj}}$ can be found using an EM algorithm where

$$\begin{aligned} \text{E-step} \quad : \quad \overline{\lambda_{j,s}}(i) &= E[\lambda_{j,s}(i) \mid f_{\theta_1(i)j}, f_{\theta_2(i)j}, \theta(i), g_{ij} = 1] \\ &= \begin{cases} \frac{f_{\theta_1(i)j}(1-f_{\theta_2(i)j})}{f_{\theta_1(i)j}(1-f_{\theta_2(i)j})+(1-f_{\theta_1(i)j})f_{\theta_2(i)j}}, \text{for } s = \theta_1(i) \\ \frac{f_{\theta_2(i)j}(1-f_{\theta_1(i)j})}{f_{\theta_1(i)j}(1-f_{\theta_2(i)j})+(1-f_{\theta_1(i)j})f_{\theta_2(i)j}}, \text{for } s = \theta_2(i) \\ 0, \text{for } s \notin \{\theta_1(i), \theta_2(i)\} \end{cases} \end{aligned} \tag{8.4}$$

$$\text{M-step} \quad : \quad \hat{f_{sj}} = \frac{2n_{2,2}^{sj} + n_{2,1}^{sj} + n_{1,2}^{sj} + \sum_{j \in \mathcal{H}_i} \overline{\lambda_{j,s}}(i)}{2n_{2,2}^{sj} + 2n_{2,1}^{sj} + 2n_{2,0}^{sj} + n_{1,2}^{sj} + n_{1,1}^{sj} + n_{1,0}^{sj}} \tag{8.5}$$

Here $\lambda_{j,s}(i)$ refers to coordinate s of the vector $\lambda_j(i)$. $n_{k,u}^{sj}$ refers to the number of individuals who have $u \in \{0, 1, 2\}$ minor alleles and $k \in \{1, 2\}$ copies of alleles from population A_s at site j. The counts of these individuals can be obtained based on the classification $\theta(i)$. Notice that the term corresponding to the heterozygous sites which have a single allele from population A_s has its contribution modified by $\overline{\lambda_{j,s}}(i)$. We can now perform expectation-maximization iterations using equations 8.5 and 8.4. The convergence of these iterations provides us a maximum likelihood estimate of $\vec{f_1}, \ldots, \vec{f_K}$. These estimates can then be used in the next iteration to estimate θ using Equation 8.1.

Initializing the clusters

We now describe how we obtain an initial setting of the parameters *i.e.*, the classification function θ or the allele frequencies $\vec{f_1}, \ldots, \vec{f_K}$, which are used as starting points by the EM algorithm. We focus here on two specific scenarios. The first scenario is the case where there are two ancestral populations *i*.e., $K = 2$ and unknown allele frequencies $\vec{f_1}, \ldots, \vec{f_K}$. In this instance, we use an algorithm called `MAXVAR` to provide an initial solution to the EM algorithm. The main motivation behind `MAXVAR` is to quickly produce a reasonable classification. The algorithm runs in time linear in the number of SNPs and can take advantage of results computed from adjacent windows. We have also considered using spectral clustering but in practice we found that the final classification accuracy is nearly the same as `MAXVAR` though the running time is increased. The result from `MAXVAR` is a classification of the individuals which is then used in Equation 8.2 of the EM.

The second scenario is the case where $K \geq 2$ and estimates of the allele frequencies $\hat{f}_1, \ldots, \hat{f}_K$ in the ancestral populations are known. In this case, these allele frequencies are used as a starting solution in Equation 8.1 of the EM algorithm.

The (`MAXVAR`) algorithm. When we have two populations, we estimate a window length l such that most of the individuals have no breakpoints within a window. Thus the ancestries of these individuals are A_1A_1, A_1A_2 or A_2A_2. We define $\alpha = \alpha_2$ as the admixture fraction of the smaller of the two populations. We now describe a method to find the ancestry of each individual in this window. We call this the `MAXVAR` algorithm.

We first define a similarity score $\mathcal{S}$ between a pair of individuals. For each SNP j, let $\mu_j = \frac{\sum_i g_{ij}}{n}$, where n is the number of individuals, and let $\sigma_j = \sqrt{\frac{\sum_i (g_{ij} - \mu_j)^2}{n}}$. For two

individuals i_1, i_2, we define

$$\mathcal{S}(i_1, i_2) = \sum_{j=1}^{l} \frac{(g_{i_1 j} - \mu_j)(g_{i_2 j} - \mu_j)}{\sigma_j^2}.$$

For each $i \leq n$, let $Var(i) = \sum_{i':i' \neq i} \mathcal{S}(i, i')^2$ denote the similarity of all other individuals to individual i, and let $i^* = \text{argmax}_i\{Var(i)\}$. The `MAXVAR` algorithm simply finds i^*, and clusters the individuals according to the values $\mathcal{S}(i^*, i)$. In particular, we order the individuals according to these values, and the smallest $(1-\alpha)^2 n$ individuals are assigned as the ancestry of A_1A_1, the largest $\alpha^2 n$ individuals are assigned as the ancestry of A_2A_2, and the rest are assigned A_1A_2. We provide a formal proof of correctness of the `MAXVAR` method in the supplementary materials (Section 8.2).

Known ancestries. The problem of estimating the ancestry is considerably simpler if we are provided estimates of the ancestral allele frequencies. In this case, as before, we first estimate the window length l. Within each window, we then estimate the ancestries using the likelihood function given by Equation 8.1 with the given ancestries $\hat{f}_1, \ldots, \hat{f}_K$ used as the starting solution. The ancestries predicted at each SNP are combined using a majority vote.

8.1.4 Choosing the window length

In order for the local predictions to achieve reasonable accuracy, the length of the window l should be short enough so that most individuals do not have a breakpoint in the window and long enough so that the SNPs provide sufficient information to observe a difference between the populations. Note that we use the term *breakpoint* to refer to a recombination event that results in a change in ancestry of the adjacent SNPs. The power of our method stems from the fact that long windows provide much more information than any local behavior, provided that there are not too many individuals with breakpoints in the window. We are looking for the maximum window length l so that the errors in the classification due to breakpoints in the window are bounded. We present empirical results that validate the window length estimates in Section 8.4 of the Supplementary Material.

In each window, the errors in the classification depend on the length of the window, the number of individuals, and the distance between the populations. Evidently, it is hard to predict these errors as the distance between the populations is unknown, and the performance of the EM algorithm is unpredictable for a finite sample. To obtain a bound on the errors, we consider the most accurate classification of the individuals in a window. Such a classification is allowed to assign ancestries to the individuals in a window with knowledge of their true ancestral states a_{ij}^p for $p = 1, \ldots, K$. Thus an individual whose ancestry is A_sA_t over the length of the window is always classified correctly. The only errors made by such a classification are due to the locations of the breakpoints. In the presence of a breakpoint, an individual would be assigned an ancestry so that the number of errors is minimized. For instance, an individual with a breakpoint at position j and

ancestries A_{s_1} and A_{s_2} on either side of the breakpoint gets assigned the majority ancestry over the length of the window *i.e.*, the individual gets classified as A_{s_1} if $j > \lceil \frac{l}{2} \rceil$ and A_{s_2} otherwise. It is easy to see that the larger the window size l, the more likely it is for an individual to have a breakpoint and hence, the more the errors in the optimal classification.

The number of recombination events throughout time along a specific window is assumed to be Poisson distributed with parameter $(g-1)lr$. Therefore, as long as $(g-1)lr << 1$, it can be verified that the probability to have a breakpoint in the window is upper-bounded by $2(g-1)lr \sum_{i<j} \alpha_i \alpha_j$ under the assumption of random-mating and that the admixture fractions of the population right before recombination are α_i. Therefore, the probability for a breakpoint on either chromosome is bounded by $\gamma = 4(g-1)rl \sum_{i<j} \alpha_i \alpha_j$.

For a given window, the above analysis shows that the expected fraction of individuals with no breakpoints is $1-\gamma$. We can now use this to obtain a bound on the fraction of errors in a window. Let X be the fraction of errors in a window of an algorithm that makes the optimal classification. Let I be the number of breakpoints in the window. We compute

$$E[X] = E[E[X|I]] = \sum_i Pr[I=i] E[X|I=i]$$

Note that $E[E[X|I=0]] = 0$ since the optimal classification in this case makes no errors. When there is a single breakpoint $I=1$, the breakpoint is distributed uniformly over the length of the window. We denote the position of the breakpoint $J \sim Unif(1,l)$. The fraction of errors in the presence of a single breakpoint at position J is

$$(X|I=1, J=j) = \begin{cases} 1 - \frac{j}{l} & j > \lceil \frac{l}{2} \rceil \\ \frac{j}{l} & \text{otherwise} \end{cases} \tag{8.6}$$

We now have

$$E[X|I=1] = 2 \sum_{j=1}^{\lceil \frac{l}{2} \rceil - 1} \frac{j}{l} \frac{1}{l} \leq \frac{1}{4}$$

If $glr << 1$,we can ignore $\Pr[I>1]$ so that

$$\begin{aligned} E[X] &\leq 0 \cdot \Pr[I=0] + \frac{1}{4} \cdot \Pr[I=1] + 1 \cdot \Pr[I>1] \\ &\approx \gamma \frac{1}{4} \end{aligned} \tag{8.7}$$

For a bound ϵ on the expected fraction of errors, we get $\gamma < 4\epsilon$. Rewriting the window length l in terms of ϵ, we get

$$l \leq \frac{\epsilon}{(g-1)r \sum_{i<j} \alpha_i \alpha_j} \tag{8.8}$$

While these arguments bound the errors in a single window, it is also possible to bound the errors due to overlapping windows at a SNP. In this case, the use of a majority vote can be shown to further improve the accuracy of the predictions. The details of this analysis can be found in the supplementary materials (Section 8.3).

The analysis presented here is specific to the model of admixture described at the start of Section 8.1.1. However, it is easy to see that the analysis can be extended to the case of non-uniform recombination rate, where the probability for a recombination in position i is r_i. In that case, the term $(g-1)lr$ should be replaced by $(g-1)\sum_{i=0}^{l} r_i$.

The model considered so far does not take into account the distance between the ancestral populations while choosing the window length. When the ancestral genotypes are known, the window length can be chosen to trade-off the accuracy in separating the ancestral genotypes with an increase in the errors due to breakpoints. A binary search over the window lengths can then pick the optimal window length as discussed in the supplementary materials (Section 8.7).

8.2 Correctness of MAXVAR

In this section, we analyze the correctness of the MAXVAR algorithm. We have two populations A_1 and A_2. We denote $\alpha = \alpha_2$ as the admixture fraction of A_2 - the smaller of the two populations. The MAXVAR algorithm classifies the individuals into three types of ancestries, i.e., A_1A_1, A_1A_2, and A_2A_2. The algorithm works by first picking a specific individual termed a *pivot*, and then clustering individuals based on their similarity to the pivot. We show that when the the populations are significantly different from each other, the pivot will have an ancestry A_2A_2 with high probability. In this case, we show that one can define a similarity score $\mathcal{S}$ (as defined in the Methods Section), such that the individuals who are also of ancestry A_2A_2 have positive similarity score to the pivot while those with ancestry A_1A_1 have negative similarity scores in expectation. Thus, the individuals with the smallest $(1-\alpha)^2 n$ values of the similarity score are assigned an ancestry of A_1A_1, the largest $\alpha^2 n$ values are assigned an ancestry of A_2A_2 and the rest are assigned A_1A_2.

Let $p_{A_1A_1}, p_{A_1A_2}, p_{A_2A_2}$ be the frequencies of individuals of the three types in the population. We assume that $p_{A_1A_1} = (1-\alpha)^2$, $p_{A_2A_2} = \alpha^2$, and that $p_{A_1A_2} = 2\alpha(1-\alpha)$. Let p_k, q_k be the minor allele frequencies of population A_1 and A_2 respectively in position k. Furthermore, we assume that the values of μ_k and σ_k (as defined in the Methods Section) are constants, and that $\mu_k = \alpha p_k + (1-\alpha)q_k$, $\sigma_k^2 = 2\alpha^2 p_k(1-p_k) + 2\alpha(1-\alpha)[p_k(1-p_k) + q_k(1-q_k)] + 2(1-\alpha)^2 q_k(1-q_k)$. If the number of individuals is large enough, the variance is quite low, and therefore this is not a restrictive assumption. We define the *distance* between the two populations as $W = \sum_k (p_k - q_k)^2 \sigma_k^2$. Under these assumptions, it is easy to see that if aa, ab, bb are three given individuals with ancestry A_1A_1, A_1A_2, A_2A_2 respectively in the window, then the expected similarity score $\mathcal{S}$ between pairs of individuals is:

$$\begin{aligned} E[\mathcal{S}(aa, aa')] &= 4\alpha^2 W, & E[\mathcal{S}(aa, ab)] &= -2(1-2\alpha)\alpha W, \\ E[\mathcal{S}(aa, bb)] &= -4(1-\alpha)\alpha W, & E[\mathcal{S}(ab, ab')] &= (1-2\alpha)^2 W, \\ E[\mathcal{S}(ab, bb)] &= 2(1-2\alpha)(1-\alpha) W, & E[\mathcal{S}(bb, bb')] &= 4(1-\alpha)^2 W, \end{aligned} \tag{8.9}$$

where aa', ab', bb' are individuals with ancestries A_1A_1, A_1A_2, A_2A_2, but they are different individuals than aa, ab, and bb. From this, it is easy to verify that the expected sum of squares over all individuals that are different from aa, ab, and bb can be approximated as:

$$\begin{aligned}
\sum_{i:i\neq bb} E[\mathcal{S}(i,bb)]^2 &\approx 8(1-\alpha)^3\alpha W^2 n \\
\sum_{i:i\neq ab} E[\mathcal{S}(i,ab)]^2 &\approx 2\alpha(1-\alpha)(1-2\alpha)^2 W^2 n \\
\sum_{i:i\neq aa} E[\mathcal{S}(i,aa)]^2 &\approx 8\alpha^3(1-\alpha)W^2 n
\end{aligned}$$

The only reason for the approximation is that the number of individuals with ancestry A_2A_2 that are different from bb is $(1-\alpha)^2 n - 1$, while we consider it as $(1-\alpha)^2 n$. This approximation is not restrictive when the number of individuals is reasonably large.

Defining $P_k = p_k(1-p_k), Q_k = q_k(1-q_k)$, we can write the variance of these scores as

$$\begin{aligned}
V[\mathcal{S}(aa,aa)] &= 4\textstyle\sum_k \frac{Q_k^2}{\sigma_k^4}, & V[\mathcal{S}(aa,ab)] &= 2\displaystyle\sum_k \frac{Q_k^2+P_kQ_k}{\sigma_k^4} \\
V[\mathcal{S}(aa,bb)] &= 4\textstyle\sum_k \frac{P_kQ_k}{\sigma_k^4}, & V[\mathcal{S}(ab,ab)] &= \displaystyle\sum_k \frac{P_k^2+Q_k^2+2P_kQ_k}{\sigma_k^4} \\
V[\mathcal{S}(ab,bb)] &= 2\textstyle\sum_k \frac{P_k^2+P_kQ_k}{\sigma_k^4}, & V[\mathcal{S}(bb,bb)] &= 4\displaystyle\sum_k \frac{P_k^2}{\sigma_k^4}
\end{aligned}$$

We can conclude that the variance of the similarity scores from the rest of the individuals to an individual with one of the 3 ancestries is

$$\begin{aligned}
V[\sum_{i:i\neq bb}\mathcal{S}(i,bb)] &\approx 4\alpha^2 n\sum_k \frac{P_k^2}{\sigma_k^4} + 4\alpha(1-\alpha)n\sum_k \frac{P_k^2+P_kQ_k}{\sigma_k^4} + 4(1-\alpha)^2 n\sum_k \frac{P_kQ_k}{\sigma_k^4} \\
&= 4n\sum_k P_k \frac{\alpha^2 P_k + \alpha(1-\alpha)(P_k+Q_k) + (1-\alpha)^2 Q_k}{\sigma_k^4} = 2n\sum_k \frac{P_k}{\sigma_k^2} \\
V[\sum_{i:i\neq aa}\mathcal{S}(i,aa)] &\approx 4\alpha^2 n\sum_k \frac{P_kQ_k}{\sigma_k^4} + 4\alpha(1-\alpha)n\sum_k \frac{Q_k^2+P_kQ_k}{\sigma_k^4} + 4(1-\alpha)^2 n\sum_k \frac{Q_k^2}{\sigma_k^4} \\
&= 4n\sum_k Q_k \frac{\alpha^2 P_k + \alpha(1-\alpha)(P_k+Q_k) + (1-\alpha)^2 Q_k}{\sigma_k^4} = 2n\sum_k \frac{P_k}{\sigma_k^2}
\end{aligned}$$

$$V[\sum_{i:i\neq ab}\mathcal{S}(i,ab)]$$

$$\begin{aligned}
&\approx n\sum_k \frac{2\alpha^2(P_k^2+P_kQ_k) + 2\alpha(1-\alpha)(Q_k^2+P_k^2+2P_kQ_k) + 2(1-\alpha)^2(Q_k^2+P_kQ_k)}{\sigma_k^4} \\
&= n\sum_k \frac{Q_k+P_k}{\sigma_k^2}
\end{aligned}$$

Hence, the squared distances are given by

$$\begin{aligned}
E[\sum_{i:i\neq bb}\mathcal{S}(i,bb)^2] &= V[\sum_{i:i\neq bb}\mathcal{S}(i,bb)] + \sum_{i:i\neq bb}E[\mathcal{S}(i,bb)]^2 \\
&\approx n\left(2\sum_k \frac{P_k}{\sigma_k^2} + 8\alpha(1-\alpha)^3W^2\right) \\
E[\sum_{i:i\neq ab}\mathcal{S}(i,ab)^2] &= V[\sum_{i:i\neq ab}\mathcal{S}(i,ab)] + \sum_{i:i\neq ab}E[\mathcal{S}(i,ab)]^2 \\
&\approx n\left(\sum_k \frac{P_k+Q_k}{\sigma_k^2} + 2\alpha(1-\alpha)(1-2\alpha)^2W^2\right) \\
E[\sum_{i:i\neq aa}\mathcal{S}(i,aa)^2] &= V[\sum_{i:i\neq aa}\mathcal{S}(i,aa)] + \sum_{i:i\neq aa}E[\mathcal{S}(i,aa)]^2 \\
&\approx n\left(2\sum_k \frac{Q_k}{\sigma_k^2} + 8\alpha^3(1-\alpha)W^2\right)
\end{aligned}$$

Asymptotically, when n is large enough, the pivot i^* will be from A_2A_2 if $E[\sum_{i:i\neq bb}\mathcal{S}(i,bb)^2] > max(E[\sum_{i:i\neq ab}\mathcal{S}(i,ab)^2], E[\sum_{i:i\neq aa}\mathcal{S}(i,aa)^2])$. After simplifying the above expressions, we get that the requirement is that

$$\sum_k \frac{P_k - Q_k}{\sigma_k^2} + 4\alpha(1-\alpha)(1-2\alpha)W^2 \;>\; 0$$

The last inequality holds if the distance between the populations (W) is large enough. In that case, by Equation 8.9, the ordering of the individuals according to their similarity to the pivot should give the correct clustering asymptotically.

8.3 Accuracy of the window length and the majority vote

For a given window, the analysis in Section 8.1.4 shows that the expected fraction of individuals with no breakpoints is $1-\gamma$. Here, we strengthen this analysis under the assumption that the errors in the predictions of the different windows are independent.

It is easy to see that the expected fraction of individuals with two or more breakpoints in a window is smaller than γ^2. For a given individual with a breakpoint in position i, we denote the ancestry by $(A_{s_1}, A_{s_2}, i, A_{s_3})$, where A_{s_1} is the ancestry of the non-recombinant chromosome and A_{s_2} and A_{s_3} are the ancestries of the recombinant chromosome. We assume that the probability to classify such an individual as $A_{s_1}A_{s_2}$ is $\frac{i}{l}$, and the probability to classify it as $A_{s_1}A_{s_3}$ is $1-\frac{i}{l}$. There are l windows that overlap with any SNP. Consider a SNP which is a distance d away from a breakpoint. Let X be the number of times that

the SNP is incorrectly classified as $A_{s_1}A_{s_3}$. Clearly,

$$E[X] = \sum_{i=1}^{l-d} \frac{i}{l} \approx \frac{(l-d)^2}{2l}.$$

Using a Chernoff bound [17], the probability to incorrectly classify this SNP after the majority vote is

$$\Pr(X > \frac{l}{2}) = \Pr(X > (1 + \frac{d}{l-d})^2 E[X]) < e^{-\frac{(\frac{d}{l-d}(2+\frac{d}{l-d}))^2 E[X]}{2}} = e^{-\frac{(d(2+\frac{d}{l-d}))^2}{4l}} < e^{-\frac{d^2}{l}}.$$

In the case that there are no other breakpoints within distance l from the breakpoint considered, the expected number of errors around the breakpoint for the individual is

$$\int_0^l e^{\frac{-x^2}{l}} dx = \int_0^{\sqrt{2l}} e^{\frac{-x^2}{2}} \sqrt{l/2} dx \approx \sqrt{l/2}\sqrt{2\pi} = \sqrt{l/\pi}$$

If there are breakpoints within distance l of each other, we take the worst-case assumption that all windows containing the two breakpoints make erroneous predictions over their entire length l. Since the expected fraction of breakpoints in an individual is $\frac{\gamma}{l}$, and the expected fraction of pairs of breakpoints that are of distance smaller than l is at most γ^2, we can bound the expected number of errors as $\frac{\gamma}{\sqrt{\pi l}} + \gamma^2 = 4\sum_{i<j} \alpha_i\alpha_j(g-1)r(\sqrt{l/\pi} + 4(\sum_{i<j} \alpha_i\alpha_j)(g-1)rl^3)$. Based on this analysis, a sufficient condition to achieve a desired error rate of ϵ is to have $\frac{1}{\pi}(\frac{8(g-1)r(\sum_{i<j} \alpha_i\alpha_j)}{\epsilon})^2 < l < \frac{1}{4(g-1)r\sum_{i<j} \alpha_i\alpha_j}\sqrt{\frac{\epsilon}{2}}$. For typical values of g and r, the lower bound on l is small enough to be negligible.

8.4 Estimate of window length

The window length derived in Equation 8.8 bounds the classification errors within each window to a desired error rate ϵ. Since all SNPs within a window are assigned the same ancestry, *any* algorithm that is used within this window will incur some errors in the presence of breakpoints. Hence the window length was calculated under the assumption that the classification algorithm within the window was the most accurate possible *i*.e., any errors in the classification were only a result of breakpoints within a window. Here we empirically show that, for the window lengths computed using Equation 8.8, the average classification error for a most accurate classification is bounded by the error rate ϵ which is set to 0.10.

Within each window, the most accurate ancestry assignment is inferred assuming that the true ancestries are known. The most accurate assignment consists of assigning to an individual the ancestry found in a majority of the SNPs in that window. Thus, an individual who has no breakpoints is always correctly classified while an individual with a breakpoint at position $i < \lfloor \frac{l}{2} \rfloor$ in a window of length l and ancestries A_{s_1} and A_{s_2} on either side of the breakpoint will have errors in positions $\{1, \ldots, i\}$. The error rate for a window is the fraction of positions that are incorrectly classified in the window.

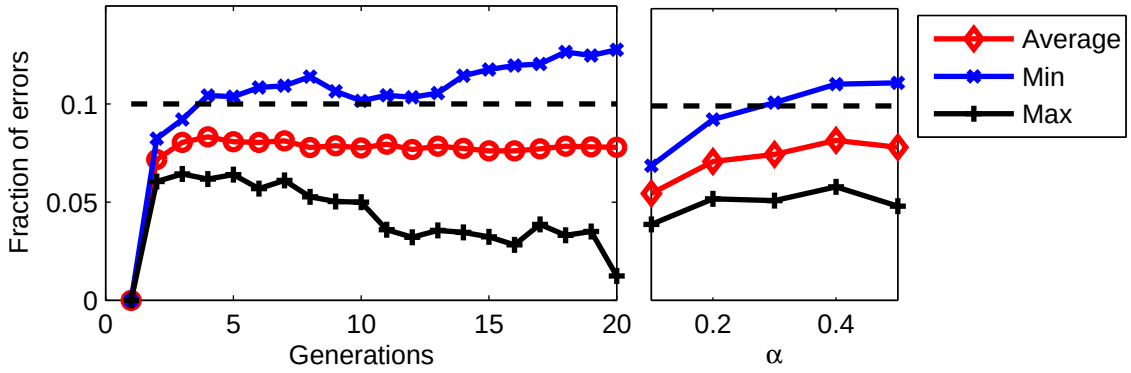


Figure 8.2: Empirical validation of the window length estimates. The window length is estimated in the Methods Section (Section 8.1.4). These estimates are based on a parameter ϵ, which represents the average desired fraction of errors incurred by the most accurate classification algorithm that can only return one of A_1A_1, A_1A_2, A_2A_2 for the entire window. The figure presents the actual average error rates for different values of g and α, run on the CEU-YRI dataset, with $\epsilon = 0.1$. Evidently, the actual average error rate falls within the desired error bound. The maximum and the minimum fraction of errors in a window are also shown.

We computed the average of these errors in overlapping windows that span chromosome 1 of the YRI-CEU dataset for different values of g and α. We see in Figure 8.2 that the average error is below ϵ. However, the variance of the estimates (indicated by the minimum and the maximum fraction of errors) increases with larger g or with $\alpha \to 0.5$. The window size estimates seem to provide a good bound on the average fraction of errors due to breakpoints.

8.5 Results

We empirically evaluated `LAMP` on various data-sets and compared its performance with other tools that infer ancestry in admixed populations. When comparing to previous methods, it is important to note that the inputs needed for the different methods are different. In particular, in `SABER` [114], the genotypes from the pure ancestral populations are assumed to be known, while in `LAMP`, we do not need this extra information. On the other hand, similar to `SABER`, `LAMP` assumes that the recombination rates across the genome and the admixture fraction $(\alpha_1, \ldots, \alpha_k)$ are known; the latter can be found with reasonable accuracy using existing methods such as `STRUCTURE` or `EIGENSTRAT`, while the former can be obtained from the previous estimates of recombination rates based on the HapMap data [85]. We also provide `LAMP` with an estimate of the number of generations g of admixture which can be approximated if the history of the admixed populations is known. We show below that our method is robust to deviations in the estimate of g. For `SABER`, we set the parameter τ, which roughly corresponds to the number of generations since admixing, to g. We found that allowing `SABER` to estimate the values of τ yielded much poorer estimates

of ancestry.

Simulated Datasets. We simulated admixed populations from the HapMap data in the following manner. We used the SNPs of chromosome 1 from the 500K Affymetrix GeneChip assay ® from each of the four HapMap populations: Yoruba people from Ibadan, Nigeria (YRI); Japanese from the Tokyo area (JPT); Han Chinese from Beijing (CHB); and Utah residents with ancestry from northern and western Europe (CEU). Overall, these span 38,864 SNPs for 60 unrelated individuals from CHB and YRI and 45 unrelated individuals from CHB and JPT.

For each pair of HapMap populations, we simulated admixed populations by random mating of individuals from the two populations across several generations. We started by joining a random set of αn individuals from the first population, and $(1-\alpha)n$ individuals from the second population. For the next generation, we repeatedly picked a random pair of individuals from the combined set of individuals, and generated a child for this pair by transmitting one chromosome from each individual. We repeated this process for g generations. We set the recombination rate to be 10^{-8} per base pair per generation consistent with previous studies [86]. We note that this model is a worst case scenario in the sense that in practice the populations are expected to mix in a slower rate, since individuals tend to mate with individuals from a similar ancestral background. We simulated admixture for various values of g and α; in the rest of this manuscript, the values of g and α are 7 and 0.2, unless stated otherwise. These parameters roughly match the nature of admixing in African-American populations [20, 35, 43, 49, 116].

`LAMP`'s performance and accuracy. We evaluated the accuracy of the ancestry estimates inferred by `LAMP`. We consider the two versions of `LAMP`, i.e., the de-novo inference of the local ancestries, as well as the inference of the local ancestries based on genotype data of the original ancestral populations. We refer to the latter method as `LAMP-ANC`. For each individual i and SNP j, `LAMP` finds an estimate $\hat{a}^p_{ij} \in \{0, 0.5, 1\}$ for the true ancestry a^p_{ij} by a majority vote across the windows overlapping with position j. We measure the accuracy of `LAMP` as the fraction of triplets (i, j, p) for which $a^p_{ij} = \hat{a}^p_{ij}$.

We compared `LAMP` to two state of the art methods: `STRUCTURE` [90] and `SABER` [114]. `SABER` requires the input genotypes, admixture fraction α, physical location of the SNPs and the ancestral sequences that were used in the simulation (i.e., the original HapMap populations) and was also provided the number of generations g. For `STRUCTURE`, we only needed to provide the genotypes. We did not compare `LAMP` to methods such as AdmixMap [71] and AncestryMap [43], since the high density of markers made these methods infeasible.

Table 8.1 summarizes the prediction accuracies of `LAMP`, `LAMP-ANC`, `SABER`, and `STRUCTURE`. `LAMP` and `LAMP-ANC` were run on the set of 38864 SNPs of chromosome 1. `SABER` and `STRUCTURE` were run on non-overlapping windows of 4000 SNPs that included 36000 of the original 38864 SNPs. This was done because `STRUCTURE` got into numerical instabilities when a large number of SNPs were used, and `SABER` crashed for an unknown reason when run on all the SNPs over the set of 500 individuals. For `STRUCTURE` the linkage model was

Dataset	Distance	LAMP	LAMP-ANC	SABER	STRUCTURE
YRI-CEU	0.055	0.94	0.95	0.87	0.84
CEU-JPT	0.036	0.87	0.93	0.82	0.47
JPT-CHB	0.0045	0.48	0.72	0.68	0.40
Time (sec)		394 (38864 SNPs)	246 (38864 SNPs)	7681 (4000 SNPs)	2.57×10^5 (4000 SNPs)

Table 8.1: A summary of the comparison between LAMP, LAMP-ANC, SABER, and STRUCTURE. The accuracy across all positions on chromosome 1 is shown for the three admixed populations. The distance between the admixing population (measured by the mean squared distance between the allele frequency vectors) is also shown indicating the difficulty in separating alleles from the populations. The time taken to run each of the methods is shown. LAMP and LAMP-ANC were run on the entire set of 38864 SNPs while SABER and STRUCTURE were run on non-overlapping blocks of 4000 SNPs due to issues with scaling them to the entire dataset. For SABER and STRUCTURE the accuracies reported here are obtained by averaging the accuracies across the blocks while the running time is the time to run a single block. Each of these methods was run on a single computer.

used with 10,000 burn-in and 50,000 MCMC iterations. SABER was also seen to crash on some of the 4000 SNP blocks and these were excluded from the analysis. The accuracy of the ancestry estimates were obtained on the SNPs for which all methods returned a result. From Table 8.1, it is clear that LAMP achieves considerable improvement over the YRI-CEU and the CEU-JPT datasets, when compared to SABER or STRUCTURE. For the JPT-CHB dataset, LAMP is worse than SABER, but LAMP-ANC achieves a higher accuracy than SABER.

The accuracy of each of the methods varies across the population. We therefore measured the average accuracy in predicting the ancestries for each of the individuals. Figure 8.3 shows the cumulative distribution function of the accuracies achieved by each of the methods across the set of 500 individuals. As can be seen from the figure, the improvement of LAMP compared to STRUCTURE and SABER is quite significant. For the YRI-CEU dataset, when measuring the percentage of individuals that are predicted with an accuracy of at least 90%, LAMP achieves 90% while SABER and STRUCTURE achieve less than 10%. In general, the accuracy in the predictions that STRUCTURE makes has a higher variance than the predictions made by SABER and LAMP. On the CEU-JPT dataset, LAMP is more accurate than SABER. On the JPT-CHB dataset, SABER performs considerably better than LAMP; this is probably due to the fact that the ancestral populations, which are given to SABER but not to LAMP, are too similar to distinguish within a window; since LAMP-ANC uses the allele frequencies of the ancestral individuals as input while still inferring ancestries over entire windows, it is more accurate than SABER.

Table 8.1 also shows that LAMP achieves a gain in running time of at least two orders of magnitude. We found that, on a single computer, LAMP and LAMP-ANC take less than 30 seconds to run on a 4000 SNP block and less than 7 minutes to run on the entire set of 38864 SNPs.

These experiments suggest that LAMP is especially useful when the ancestral populations

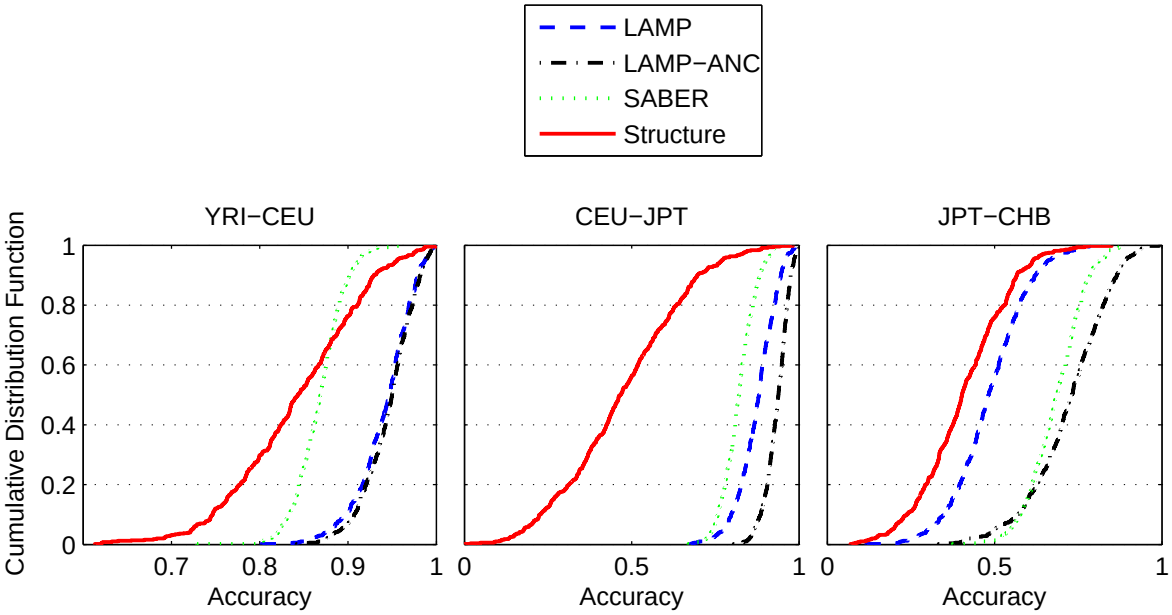


Figure 8.3: Comparison of the accuracies of LAMP, LAMP-ANC, SABER, and STRUCTURE on 3 admixed populations - YRI-CEU (left), CEU-JPT (middle), and JPT-CHB (right). The cumulative distribution function (CDF) is obtained from the accuracy of ancestry predictions for each individual. Note that the scales differ across the plots. CDFs that are to the right side correspond to higher accuracy. The graph on the left, for instance, shows that LAMP achieves an accuracy of at least 92% on 90% of the individuals. LAMP achieves an improved accuracy over SABER and STRUCTURE in the YRI-CEU and CEU-JPT populations while performing worse on the JPT-CHB population. LAMP-ANC performs consistently well on all three populations. Also notice the decrease in accuracy across all methods as we move from left to right as the ancestral populations become more similar.

are sufficiently different from each other (e.g., CEU and YRI). In those cases, it is actually not essential to genotype the ancestral individuals, as we observe that LAMP-ANC and LAMP achieve similar accuracy levels. When the populations are closer (e.g., CHB-JPT), even for a modest number of generations of mixing (in our case, 7 generations), none of the methods performs well even when the ancestral populations are given.

Inferring individual admixture. Current studies often use the individual admixture of each individual across the genome to correct for population stratification [40, 49, 66, 119]. The individual admixture of an individual is defined by the proportion of ancestors of the individual from each of the ancestral populations. For instance, for an individual with a mother from CEU and a father from YRI, the individual admixture would be 50% YRI, and 50% CEU.

Even though LAMP is designed to estimate the locus-specific ancestry, we can use it to find the individual admixture. We compare the estimates of the individual admixture obtained from LAMP with those from STRUCTURE. We used the YRI-CEU dataset with $g = 7$

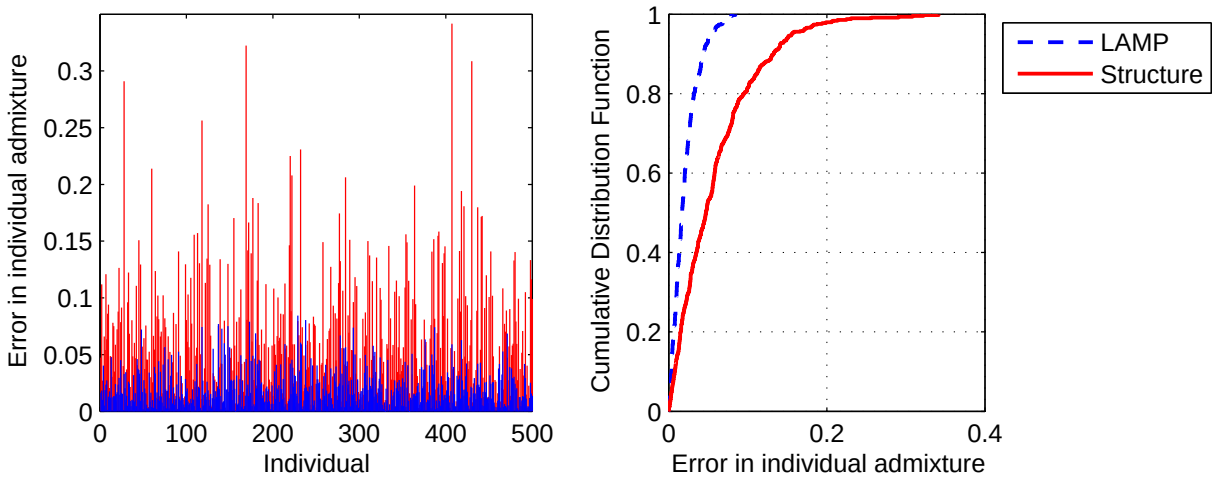


Figure 8.4: Comparison of the accuracy of methods for predicting individual admixture. a) The errors in the individual ancestries for each of the 500 YRI-CEU individuals. b) Errors in a) plotted as a Cumulative Distribution Function (CDF). The top-left region of the curve corresponds to higher accuracy. `LAMP` predicts the individual admixture with an error of less than 3% in 80% of the cases.

and $\alpha = 0.20$. We picked 4318 equally spaced SNPs from chromosome 1. This roughly matches the number of SNPs required to distinguish non-admixed individuals from even very closely related subpopulations [109]. We ran `STRUCTURE` on this set of SNPs with 10000 burn-in iterations and 50000 iterations using the NOLINKAGE model and the NOADMIX mode option set to 0. We ran `LAMP` on the entire chromosome and then calculated the locus-specific ancestry of each individual by averaging the ancestries predicted across the same set of 4318 SNPs given to `STRUCTURE`. As shown by Figure 8.4, `LAMP` consistently achieves considerably better estimates for the individual admixture. In particular, the average error rate for `LAMP` is 2.1%, while the average error rate for `STRUCTURE` is 5.4%. The difference in the performance between the methods is statistically significant (Wilcoxon signed rank test p-value of 9.9×10^{-51}). This experiment suggests that since `LAMP` is more than 600 times faster than `STRUCTURE` (see Table 8.1), it would be better to use `LAMP` across the entire genome to infer the individual admixture, than to use `STRUCTURE` across a smaller set of AIMs. We also inferred the individual admixture using the LINKAGE model in `STRUCTURE` but found that this gave a significantly higher average error rate of 9.0%.

Another method to infer the individual admixture is `EIGENSTRAT`. We ran `EIGENSTRAT` on the SNPs used above and chose the largest eigenvector. The ancestries of the individuals were obtained by scaling the entries of the eigenvector to the interval $[0, 1]$. We found this procedure to result in individual admixtures with an average error rate of 13.4%. When we included 10 ancestral individuals each from the Hapmap YRI and CEU populations reduced the average error to 4.1% (Wilcoxon signed rank test p-value of 1.3×10^{-83}). Using all 38864 SNPs decreased the average error to 11.1% and 3.6% respectively.

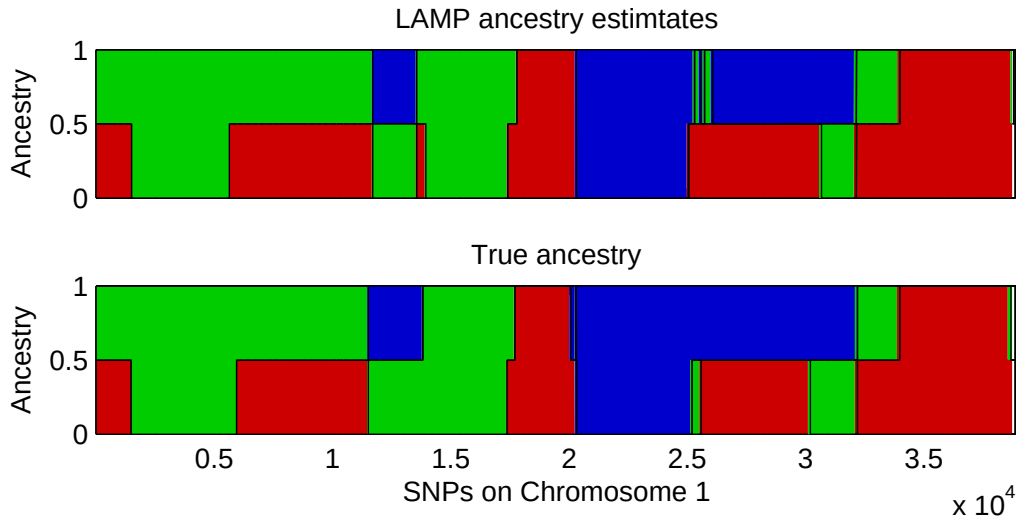


Figure 8.5: Ancestry estimates for an individual in an admixture of YRI-CEU-JPT in the ratio $0.4, 0.4, 0.2$. The top panel shows the LAMP ancestry estimates and the bottom panel the true ancestries. Red, green and blue represent YRI, CEU and JPT respectively.

LAMP's performance across three admixed populations. When more than two populations are mixed, de-novo inference of the locus-specific ancestry is a more challenging task. We therefore compare LAMP-ANC, which uses the genotypes from the ancestral populations, to SABER, on a dataset generated by the mixing of three populations (YRI, CEU and JPT). We mixed these populations in the ratio $0.4 : 0.4 : 0.2$ for seven generations. Figure 8.5 shows the ancestry estimates of LAMP-ANC for one of the individuals. LAMP-ANC accurately infers the ancestry over most of the chromosome, and it is clear that qualitatively the estimates are very close to the true ancestry. To give a more quantitative measure for the accuracy of LAMP-ANC, we calculated the cumulative distribution function of the accuracies for each individual of LAMP-ANC and of SABER (see Figure 8.6). Evidently, LAMP-ANC achieves a significantly better accuracy than SABER across the population (average accuracies of 92% and 74% respectively).

Empirical Robustness of LAMP. The performance of LAMP clearly depends on the nature of the data, on the number of generations g, and on α. We varied g for a simulated YRI-CEU admixed population with the fraction of CEU $\alpha = 0.20$. As shown in Figure 8.7, even when g is as large as 20, LAMP reaches an accuracy of 88%, and LAMP-ANC reaches an accuracy of 93%. For more realistic values of g, (i.e. $g < 10$) the accuracy of LAMP is above 93%.

To measure the effect of α on the performance of LAMP, we measured the performance for simulated data with $g = 7$ for different values of α (see Figure 8.7). We observe that LAMP performs well for values of α of up to 0.40 with it accuracy remaining above 90% and its performance drops sharply to a little above 50% accuracy for $\alpha = 0.5$.

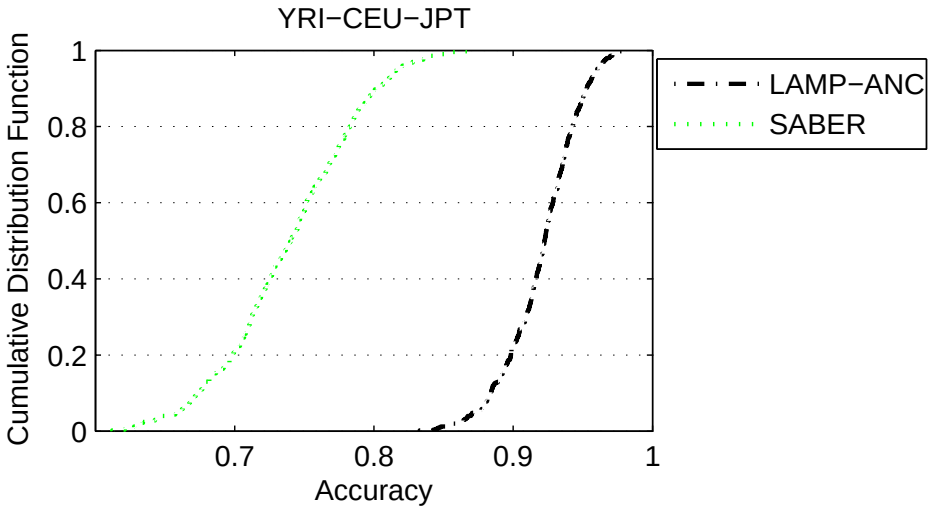


Figure 8.6: Cumulative Distribution Function (CDF) of the accuracy achieved per individual. The methods compared are `LAMP-ANC` and `SABER` for the YRI-CEU-JPT admixture. `LAMP` achieves an accuracy of at least 80% on all the individuals.

Finally, we measured the effect of the distance between the ancestral populations, by comparing the accuracy of `LAMP` across the YRI-CEU, CEU-JPT and the JPT-CHB datasets. As shown in Table 8.1 (see also Figure 8.7), `LAMP` is quite accurate on the CEU-JPT and the YRI-CEU datasets, but its performance is quite poor on the JPT-CHB dataset. In such a situation, the availability of allele frequencies is essential for accurate inference, as we observe that `LAMP-ANC` maintains an accuracy of around 70%.

Robustness to Parameter Settings. Since `LAMP` requires as an input the values of α and g, it is important to verify that inaccurate estimates of these parameters do not affect the results significantly. We tested `LAMP` by benchmarking it over the simulated YRI-CEU dataset, with true values of $g = 7$ and $\alpha = 0.2$. We ran `LAMP` on this dataset with different erroneous input values of g and α. In Figure 8.9 we observe that if the number of generations g is mistakenly given to `LAMP` as 4 or larger, than the accuracy of `LAMP` is kept reasonably high, and in particular it is at least 90%. On the other hand, it seems that if the input α is very different from the true α, `LAMP` can perform quite poorly. For instance, when the input α is set to 0.4 instead of 0.2, the accuracy level is about 85%. However, since α is a single parameter across all individuals, standard methods such as `STRUCTURE` [90] give reasonable accuracy for α (e.g. the estimates for the YRI-CEU dataset are between 0.17 and 0.24 across 10 runs) , we can safely assume that the error in the prior estimate of α is within a factor of 0.5, in which case `LAMP` maintains a very good performance.

The model used in `LAMP` requires the SNPs to be independent. To ensure this, we discard SNPs with r^2 above a threshold. We empirically chose a threshold of 0.10 for r^2 so

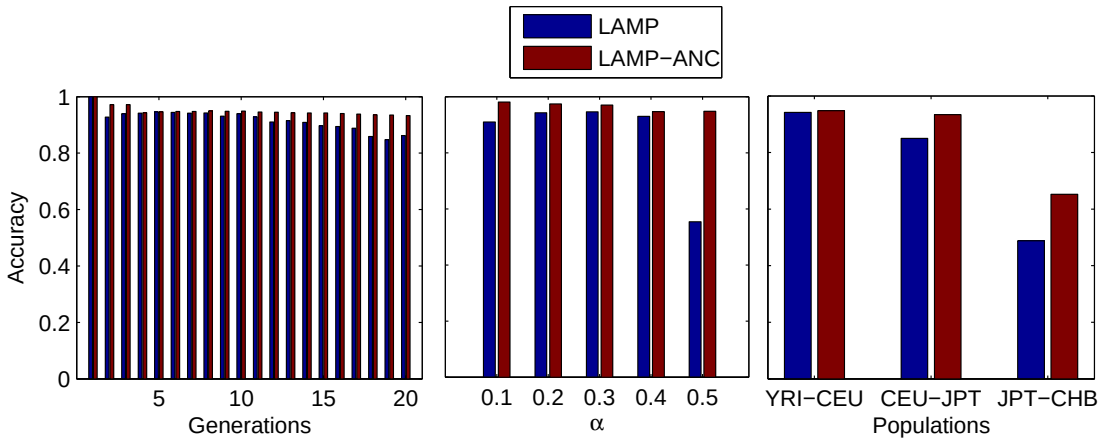


Figure 8.7: Accuracy of `LAMP` and `LAMP-ANC` with varying number of generations g, fraction of admixture α and populations. In each figure, inferring the ancestries becomes increasingly harder as we move from left to right. The difference in the accuracies between `LAMP` and `LAMP-ANC` shows the advantage conferred by a knowledge of the ancestral allele frequencies.

that we retain a majority of the SNPs. However, as shown in Figure 8.8 the accuracy of `LAMP` does not change much even when this threshold is raised so that the SNPs retained are no longer independent. The accuracy begins to decrease only at stringent thresholds below 0.005 due to a tendency to discard informative SNPs. We also examined the impact of the sample size on the ancestry estimates. While an increase in sample size might lead to SNPs being significantly linked even when the mutual r^2 is small, for practical purposes, such SNPs are essentially uncorrelated. Thus, `LAMP` is also robust to the sample size as shown in Figure 8.8.

Finally, we measured the effect of the method used to simulate the data on the different algorithms. To achieve this, we amplified the Hapmap haplotypes for YRI and CEU populations using the model of Li and Stephens [79]. Briefly, the Li and Stephens model generates additional haplotypes based on the ones already observed. The newly generated haplotypes are composed from previous ones, assuming mutation and recombination. The recombination rate in this model depends on the number of observed haplotypes, such that the rate is higher when less haplotypes are observed. This reduces the recurrent sampling of haplotypes, and as was shown by Li and Stephens, mimics more accurately the generation of haplotypes. This resulted in a set of 10000 ancestral individuals which then underwent admixture with $g = 7$ and $\alpha = 0.20$ as described earlier. On this new dataset, the accuracies obtained by `LAMP`, `LAMP-ANC` and `SABER` were 94.72%, 94.70%, and 89.09% respectively. The accuracies are close to the accuracies obtained on the YRI-CEU dataset described in Table 8.1.

Since the ancestral allele frequencies used in `LAMP-ANC` were estimated from the same data that was used to generate the admixed datasets, there is a potential risk of overfitting. To make sure that this is not the case, we partitioned the founding YRI and CEU

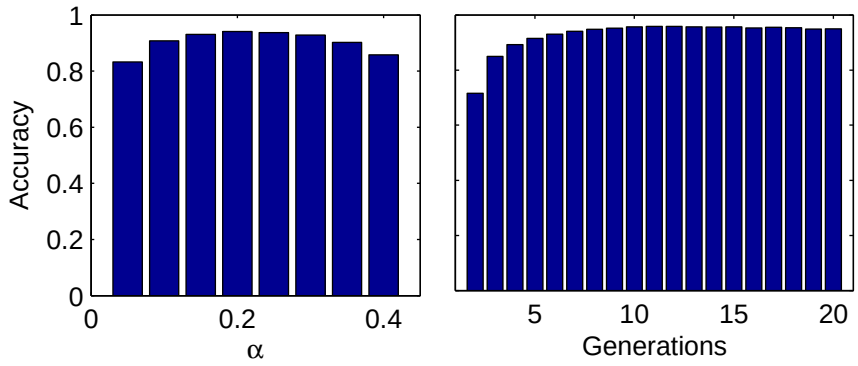


Figure 8.8: Robustness of LAMP estimates to the r^2 threshold used to discard SNPs and the sample size. The accuracy of LAMP has been shown on the YRI-CEU dataset for different values of g and α with true values of $g = 7$ and $\alpha = 0.20$.

populations into two equal-sized sets. We chose one of the two sets from each population to generate a YRI-CEU dataset with parameters $g = 7$ and $\alpha = 0.20$. Ancestral allele frequencies were estimated from the other set. The accuracy of LAMP-ANC in this setting was 94.06% which is very close to the previous estimates obtained. Running the same procedure on the amplified datasets gave an accuracy of 94.44%, and thus we conclude that the results were not due to over-fitting.

8.6 Discussion

We have presented a new method, LAMP, for *de-novo* estimation of locus-specific ancestry in recently admixed populations. Unlike previous methods for locus-specific ancestry (e.g., SABER), LAMP does not use any information about the ancestral populations (i.e., it estimates the ancestries de-novo). We show that LAMP is analytically justified and that it achieves significant improvements over existing methods both in terms of accuracy of prediction and speed. In particular, LAMP can easily be applied to whole genome datasets, and the resulting locus-specific ancestries can be estimated within a few hours.

De-novo estimation of the locus-specific ancestries is sometimes infeasible, especially when the ancestral populations are very close to each other (e.g., CHB and JPT). We therefore extended LAMP to a method called LAMP-ANC, which uses additional genotypes from the ancestral populations as priors. This approach has been shown to be useful before by methods such as SABER.

When compared to previous methods, LAMP is shown to achieve significantly better accuracy than other methods (SABER and STRUCTURE). The increase in accuracy may be crucial when trying to correct for population stratification in studies that involve recently admixed populations, as well as in studies that are based on admixed mapping. Furthermore, improved accuracy in the locus-specific ancestry estimation has potential applications in finding better signals for selection and other events across the genome.

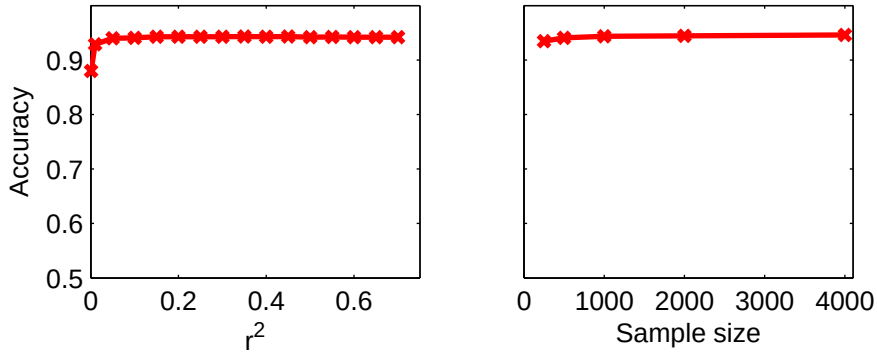


Figure 8.9: Robustness of LAMP estimates to uncertainty in the parameters - g, α. The accuracy of LAMP has been shown on the YRI-CEU dataset for different values of g and α with true values of $g = 7$ and $\alpha = 0.20$.

While LAMP relies on a knowledge of the parameters g and α, we have shown the robustness of the ancestry estimates to inaccuracies in these parameters. These parameters control the window size. As the window size is decreased, each window may contain fewer informative SNPs. On the other hand, errors in classifying individuals who have breakpoints within a window are reduced. This trade-off is illustrated in Figure 8.9 where we see that the ancestry estimates are robust when g is overestimated. In practice, we would therefore recommend using an upper bound on g when g cannot be estimated accurately. Furthermore, g may actually be a more complex parameter. For example, if some portions of the admixed population have admixed for g_1 generations and other portions have been admixed for only g_2 generations, where g_2 is smaller than g_1. In this case, g is set to be g_1, and more accurate results are expected than if the whole population has admixed for exactly g1 generation.

The fact that the LAMP algorithm performs better on the unbalanced case ($\alpha << 0.5$) than on the balanced case, seems counterintuitive at first. The reason for this phenomenon is the fact that a small α helps to break the symmetry. Even if all windows were perfectly clustered, combining the solutions of the different windows into one integrated solution is not a simple task when $\alpha = 0.5$ due to symmetry. That is, after clustering the individuals in every given window, we are still left with the problem of deciding which cluster is population 1, and which one is population 2. If $\alpha < 0.5$, then this decision is easier since the smaller cluster could be labeled as population 1, and the larger cluster as population 2.

Further, it is interesting to note that even though SABER models the LD structure while LAMP does not, it appears that LAMP performs better than SABER. This could be attributed to several possible reasons. First, it may be that the LD structure only adds slightly to the information captured by the independent SNPs. Second, it may be that optimizing the model in SABER is a harder task than optimizing the model in LAMP due to the larger number of parameters, and thus SABER may potentially not converge to the global optimum of its parameter space.

A simple extension to LAMP can be used to infer the individual admixture. As we show here, the resulting estimates of the individual admixture are considerably better than the estimates achieved by STRUCTURE or EIGENSTRAT. A number of recent studies have produced panels of Ancestry Informative Markers (AIMs) in admixed populations [30, 31, 41, 46], which are SNPs that have differing frequencies in the ancestral populations. It is possible that the AIMs may be used to improve the accuracy of individual admixture prediction done by STRUCTURE or other methods including LAMP. However, the AIMs have disadvantages since there is a risk of over-fitting, and the studied population may be somewhat different than the population for which the AIMs were found. As we show here, in an era where the genotyping technology is getting cheaper, it is useful to use the entire set of genotyped SNPs in the analysis of population stratification.

There are many possible improvements to this work, and in particular it would be important to improve the current methods in the case of very similar ancestral populations, or when more than two populations are involved. Furthermore, removing the dependency of the method on the input parameters (e.g., the number of generations g, or the admixture fraction α) may be quite useful in order to generate a rigorous statistical test for admixing. Additional improvements in the running time can be achieved by parallelizing the LAMP algorithm. This is straightforward in our case since each window could be run independent of the others and the results for windows overlapping a SNP could be aggregated in a final step.

8.7 Practical issues in implementing LAMP

In this section, we describe some of the issues that we faced while implementing LAMP. One of the issues that we needed to address was how to determine the degree of overlap between adjacent windows. An extreme degree of overlap would require adjacent windows to differ in a single SNP. In practice, we found that a smaller degree of overlap, where consecutive windows overlapped in a fraction $c = 80\%$ of their length, did not significantly change the accuracy while resulting in faster running times. The overlap between adjacent windows can be exploited to further improve the running time. While using the MAXVAR algorithm to obtain an initial classification, each window requires a computation of the similarity score between all pairs of individuals. The similarity score is computed using an inner product of the normalized genotypes as described in Section 8.1.3. Instead of computing these similarity scores over entire windows of length l, we can compute these scores over chromosomes of length $(1 - c)l$. The similarity score in a new window can then be computed from that of the previous window by adjusting for the non-overlapping regions.

As we mentioned at the end of Section 8.1.4, the window length calculation should take into account the distance between the two populations. This is feasible when the ancestral genotypes are known. In this scenario, the accuracy of the classification for a given window length can be obtained by running LAMP-ANC on the ancestral genotypes. With an increase in the window length, this accuracy is expected to increase. On the other hand, the errors due to breakpoints, as given in Equation 8.8, increases with window length. We can then

search for the window length that maximizes the product of the fraction of individuals who do not have breakpoints and the fraction of these individuals who are accurately classified. For populations that are well-separated such as YRI-CEU and CEU-JPT, we find that the number of SNPs needed to accurately classify a non-admixed individual is much smaller than the length of the window obtained from Equation 8.8, so that it suffices to simply set the window length to the latter estimate.

See the Appendix for obtaining the software package.

Part IV

Conclusions

Chapter 9

Discussion and Conclusions

In this thesis work, we primarily focused on methods to analyze SNP variation data within humans. To reconstruct the mutational history of these sequences we use the maximum parsimony objective, a particularly appropriate measure for short time scales. Although, the problem of maximum parsimony phylogeny reconstruction has been studied for several decades, typically heuristics are used to solve the problem in practice. Reconstructing 'perfect phylogenies' showed great promise in converging theoretical optimal solutions to practical fast algorithms. However, the assumption was too weak, leading to methods that heuristically tackled the case when the input was imperfect. Prior near-perfect phylogeny reconstruction algorithms were theoretical and so extremely involved that no researcher even attempted an implementation. We showed that near-perfect phylogenies can be constructed extremely efficiently with a simple and elegant algorithm. Our theoretical result showed the problem to be fixed parameter tractable (FPT), thus solving a previously open problem. Moreover, we showed that the algorithm can be implemented and runs very fast in practice. As another way to bridge optimality with practical run-time constraints, we developed extensive theoretical pre-processing techniques that do not change the problem (in terms of optimality) while typically greatly reducing the problem sizes. We then used an integer linear program (ILP) that solved the problem extremely efficiently in practice – in fact typically faster than the most widely used heuristic.

Maximum parsimony phylogeny reconstruction from genotypes has been more closely studied because of its applications to phase inference (haplotyping). Again, perfect phylogeny reconstruction was immensely useful in providing a theoretical (optimal) solution under the special case, while being efficient in practice. Researchers made progress by solving for 1-near-perfect phylogenies and those for which at most 1 site is allowed to recurrently mutate. In our work, we solve the general problem efficiently by developing a polynomial time algorithm for the q-near-perfect phylogeny problem, when q is any constant. Like in the previous case, we also developed integer linear programming (ILP) based solution that was faster in practice. We showed that popular phase inference packages such as `phase` and `haplotyper` that have been previously shown to be excellent methods for 'switch error' statistic, introduce significant errors when phylogenies are constructed from the inferred haplotypes. Instead, direct reconstruction of phylogenies from genotypes yields significantly better results.

Finally, we applied these methods to the whole genome by reconstruction maximum parsimony phylogenies around every locus of the HapMap data-set. We developed an online tool that can provide a phylogeny and the number of recurrent mutation around any position of the human genome. This is the only tool available today that can provide this information. We hope that this would be a great asset to researchers interested in specific genes and in understanding their mutation history and the influence of selective bias.

Mutational history focuses on exploiting differences in observed phenotypes based on haplotype information. To use population specific segregation due to random mating, we worked on clustering SNP sequences. Our first result was under the simple model that each individual belongs to exactly one of several ancestral population groups. Under this model, we developed an algorithm that runs in polynomial time when the length of the sequences (number of SNP sites examined) is large enough. Furthermore, the algorithm guarantees correctness of the returned clusters. The algorithm is very simple to implement and was highly efficient and accurate in practice. In particular the algorithm was more accurate than the leading software package while being 1000-fold faster.

Lastly, we studied a more general model where individuals could be admixed. For the computational problem, this means that the individuals could be a mixture of 2 or more ancestral clusters. Under this setting, we developed an algorithm that was several orders faster than the leading package while being even more accurate.

9.1 Future work

There are several avenues to extend the work that has been discussed. Some specific possibilities are outlined below.

Evolutionary Tree Reconstruction. There are several interesting theoretical directions that are still largely unsolved. One problem is that of reconstructing near-perfect phylogenies for input genotypes efficiently without using the assumption that the number of solutions to the perfect phylogeny problem is bounded by a polynomial. Another problem is to reconstruct evolutionary trees when the number of states is large, i.e., each individual's sequence is a string over s states. While the extension is not useful for analysis of SNP data, other kinds of polymorphisms, such as copy number variation, require the larger number of states.

Understanding the topology of evolutionary trees observed in reality over the whole genome is very interesting. Is a star topology more common or a path? Are there topologies that are rarely observed and, if so, what inferences can we draw about their ancestry? We can attempt to answer such questions using the algorithms described in the thesis by reconstructing piece-wise evolutionary trees for the whole genome. Using simulation and well-known loci under selective pressure (such as the lactase gene), we can therefore understand how selection influences the tree structure. It would also be interesting to figure out how the trees look at various structural positions of the genome – around genes, around intergenic regions, around specialized gene families such as those responsible for

the immune system etc. Finally, it would be of incredible utility for all researchers if we can compile the set of piece-wise phylogenies over the HapMap data into a grand unified evolutionary tree (network) that can be treated as a gold standard for reference.

Population Substructure. In prior work we were able to distinguish closely related populations (such as Chinese-Japanese) if there was no admixture and also compute admixture fractions when the ancestral sub-populations are not too similar (such as African-Americans). There is much that is unsolved. The first is to compute the approximate time (number of generations) that separates any two sub-populations. This requires new algorithms as well as extensive testing with simulations to show that the method is accurate and efficient. The second is to come up with a method that is very fine-scale and can actually find admixture within a single population if there are enough SNPs (such as different villages within a Chinese population). This is a very important problem since at the rate at which SNPs are typed, within a few months, it would be easy to obtain one million SNPs for every individual in a single experiment. This is a lot of information from which to deduce the population ancestry.

Disease Association Testing. This is the biggest application of genotyping SNPs. Given a set of SNP sequences for a set of diseased individuals (cases) and a set of healthy individuals (controls), we would like to find one or more SNPs that can be used as an explanation for the disease. The problem is significantly harder than it might seem at first glance, since the amount of signal is very tiny and statistical significance drops quickly with the number of hypotheses tested. However, it is interesting to see how our newly established methods on evolutionary tree reconstruction and population substructure can help with disease association testing.

Algorithms. As stated above, developing theoretical algorithms for the problems mentioned is exciting. However, there are a couple of very interesting paradigms that will likely be important.

First is the relationship between discrete algorithms and statistical machine learning methods. As has been observed by many, discrete algorithms use very few parameters and find optimal solutions with efficient running times. In contrast, machine learning algorithms model the problem far more accurately but require many additional parameters which are typically estimated heuristically with no good run-time bounds. Recently, several papers, including some developed as part of this thesis work [65, 95, 105, 109] have used hybrid algorithms that have proved to be very useful. Such methods find the optimal solution using discrete algorithms for a simplified problem formulation and then use that as a starting solution for the machine learning based methods. An interesting question is to find the relationship between the two objective functions for specific problems. For example, does maximizing the cut-weight also maximize the likelihood function of the cuts in the problem of detecting population substructure? Although some of the preliminary theoretical results discussed in the thesis point in this direction, much remains open. Such hybrid paradigms will be very useful in the future as well, since they combine the strengths of discrete algorithms and machine learning algorithms.

Second, it is obvious that the amount of data is increasing at a very fast rate. Therefore, there is an interesting trade-off between running time, information, and accuracy to be made. Ten years back, the data sizes were so small that it was critical to have a very good model even if the running time for that model was large. Such was the case with `STRUCTURE` [90]. However, if one needs to use such older algorithms on the current data-sets, the running time is prohibitive. Therefore, the trade-off will be to use a smaller sampled set so that the run-time is faster, thereby compromising on the information. Instead, if we can develop new methods that are significantly faster, they can use all of the data available. This was the main idea behind the new graph-based algorithm to detect sub-structure [109] and the algorithm to detect admixture [95]. These results have provided some nice examples for future work to generalize and use similar techniques to solve other applied problems.

Appendix A

A.1 Simple definitions for some genetic terms

- Allele: One of the possibly many variations in a polymorphism.
- Bi-allelic: Polymorphism that results in only two possible variations.
- Copy number polymorphism (CNP): Variation in the genome arising due to different copies of a sequence (string) of DNA.
- Genotype: Conflated combination of haplotypes. Combined allele counts from the pair of homologous chromosomes.
- Haplotype: Sequence information from a single chromosome.
- Haplotype inference (haplotyping): Finding a pair of haplotypes for each input genotype.
- Homologous pair: All chromosomes in the human genome, except X and Y are in pairs. This term refers to one such pair.
- Homozygous: Two alleles of the same kind present, one each in a (homologous) pair of chromosomes.
- Heterozygous: Different alleles present at a site in a (homologous) pair of chromosomes.
- Linkage disequilibrium (LD): Correlation of the genotypes on a pair of SNP sites due to close proximity of their physical location.
- Locus: See Site.
- Major allele: The allele that is more common in the population.
- Minor allele: The allele that is less common in the population.
- Phasing (phase inference): See Haplotyping.
- Phenotype: Physical (observable) traits such as hair-color.
- Phylogenetic tree (phylogeny): An evolutionary tree that explains the observed sequences. Vertices (points) represent current or ancestral sequences and edges (lines) represent lineage contain possible mutation events.
- Point mutation: Mutation on a single base which results in an SNP.

- Recombination: Shuffling of chromosome (during meiosis) resulting in a chromosome that contains one part from the maternal chromosome and one part from the paternal.
- Recurrent mutation: Two mutation events on the same site over time. Typically an SNP site is a result of a single ancestral mutation, rarely a recurrent mutation can occur.
- Substructure: Underlying sequences having additional structure primarily due to the samples being drawn from multiple population groups.
- Single nucleotide polymorphism (SNP): Variation in the DNA due to a single base change. They are defined to be bi-allelic.
- Site: A region of the genome. In the context of this article a region that exhibits polymorphism. In the case of a single nucleotide polymorphism, this refers to a single position (one nucleotide) of the DNA sequence.

A.2 Web Resources

SCIMP

This tool can reconstruct maximum parsimony phylogenies from a given set of haplotypes or given set of marker ids using built-in HapMap phase II sequences. It can also display the union of all MP phylogenies. More importantly, the website can scan any region of the genome that the user specifies to identify regions of high or low imperfection. The website can be accessed at: `www.cs.cmu.edu/~imperfect`

We provide some instructions on how to use the tool below. Figures A.1 and A.2 provide some screenshots of the output.

Input options. Users can either enter their own SNP-haplotype data (see format below) or use the data-set published by the International HapMap consortium.

User-specified SNP Data. The file is white-space and endline (\n) delimited. The first two numbers (white-space and/or endline delimited) should be the number of haplotype sequences and the number of SNPs respectively. Each subsequent line corresponds to one haplotype sequence, which is white-space delimited. Each site should either take the value 0 (one allele) or 1 (other allele), with no missing data allowed. In short, the set of haplotypes should form an $n \times m$ binary matrix, where n is the number of rows (number of sequences) and m is the number of columns(number of sites).

HapMap Data. Currently we support analysis of two populations: CEU and YRI. The set of contiguous SNPs that the user is interested in can be specified in three ways as described below.

Genotyped SNP Number. This corresponds to the i^{th} SNP as genotyped by the HapMap project. Notice that these numbers are consecutive, starting from 1 and ending

at the number of SNPs genotyped for that chromosome.

Chromosome Position. The physical location of the site. If the site was not genotyped, the first genotyped SNP larger than the starting value and the last SNP smaller than the ending value are used. The easiest way to browse the imperfection is to first use the genotyped SNP number option, with a pre-processed window size. Then, using the output as a guide, to focus on the SNPs physical chromosome positions.

Analysis. The user has two options to analyze the data. The first is to scan a region of the genome to identify regions of high imperfection (recurrent mutations/recombinations).

Scan. The user specifies a window size. A sliding window of SNPs is used to compute the most parsimonious phylogeny and therefore the imperfection (number of recurrent mutations) of the region. The output produces an imperfection scan for the region under study. An imperfection of k indicates that the region must have undergone k recurrent mutations (in the absence of recombinations).

Imperfection. This simply reconstructs the most parsimonious phylogeny for the specified sequences. Note that unlike heuristics that attempt to maximize parsimony, our method is guaranteed to return the most parsimonious phylogeny. In otherwords, our results should be similar to exhaustive search (brute-force) or branch-and-bound but significantly faster.

Phylogeny. Each green vertex corresponds to one or more input rows, which are indicated by the number(s) within the vertex (see below for HapMap). Each blue vertex corresponds to a Steiner (ancestral) vertex, that is simply used to link the input rows. Each edge denotes one or more mutations and is annotated with the site(s) that mutate. If the HapMap is used for scan, then in the resulting phylogenies, the individuals are numbered from 0 through 119. The individual IDs (in order) are given in these files. The IDs appear twice since there are two haplotypes per individual.

Direct Phylogeny Inference from Genotypes

This tool can find a set of haplotypes consistent with the given input genotypes such that the size of the maximum parsimony phylogeny is minimized. Note that the tool by itself does not display the phylogeny, but it is easy to use the output as the input to the previous tool to determine the phylogeny if needed. The output is a simple text interface that lists the set of haplotypes.

Input Format. The input data is white-space and endline delimited. The first two numbers (white-space and/or endline delimited) should be the number of genotype sequences and the number of SNPs, respectively. Each subsequent line corresponds to one genotype sequence, which is white-space delimited. Each site should either take the value 0 (homozygous), 1 (heterozygous) or 2 (homozygous for the second allele) with no missing

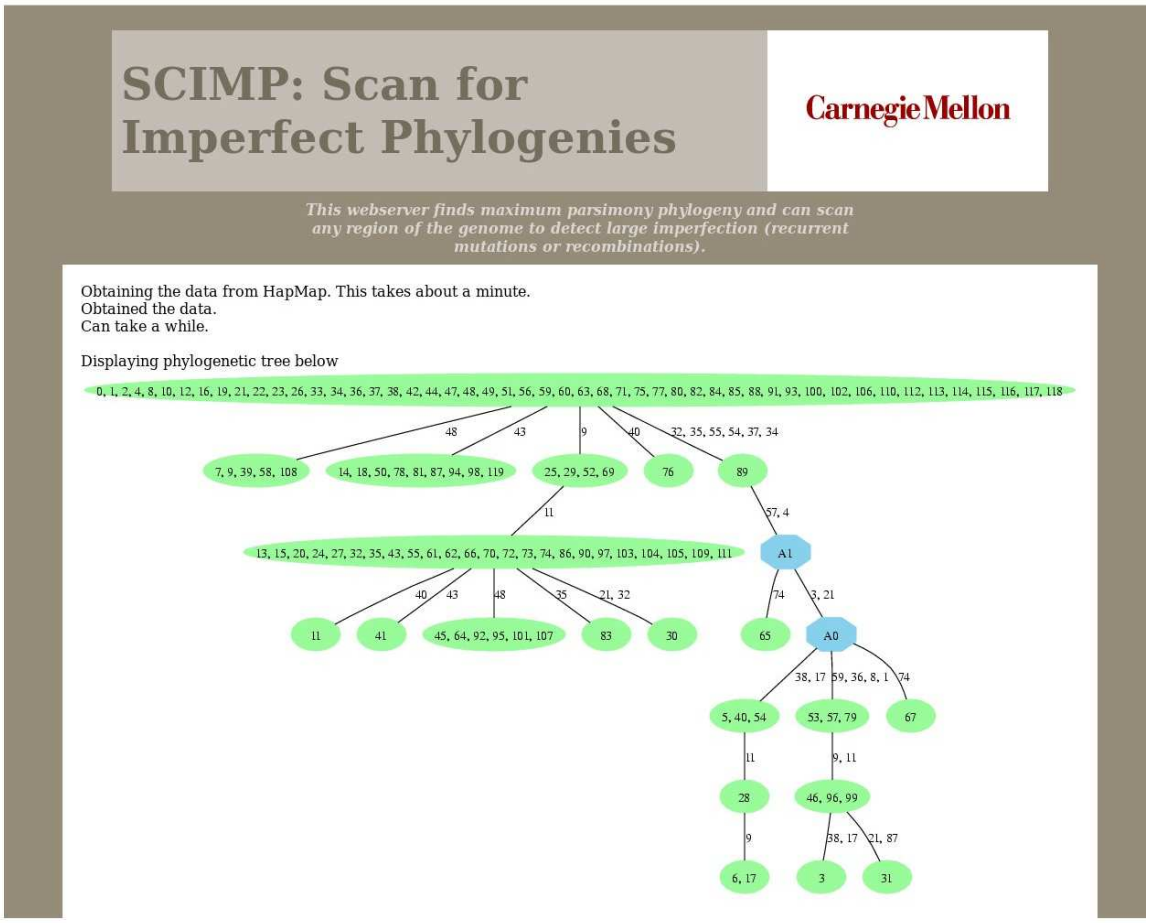


Figure A.1: Screenshot of an imperfect phylogeny reconstructed by the online tool using a set of contiguous SNPs from HapMap chromosome 21

data allowed. In short, the set of genotypes should form an $n \times m$ binary matrix, where n is the number of rows (number of sequences) and m is the number of columns (number of sites)

The website can be accessed at: `www.cs.cmu.edu/~imperfect/direct`

Population substructure (pure populations)

Given a set of genotypes, the tool can cluster them into two sub-populations. It can also provide a p-value for the strength of the clusters found.

Input Format. The webserver uses the same input format as the software package `STRUCTURE`. Each row represents the genotype information from one individual. A row begins with an individual ID and is followed with a binary representation of the genotype, which is space de-limited. A homozygous allele is represented as 0 0 or 1 1 while a heterozygous site is represented as 0 1 or 1 0. Missing data are represented by the digit 9.

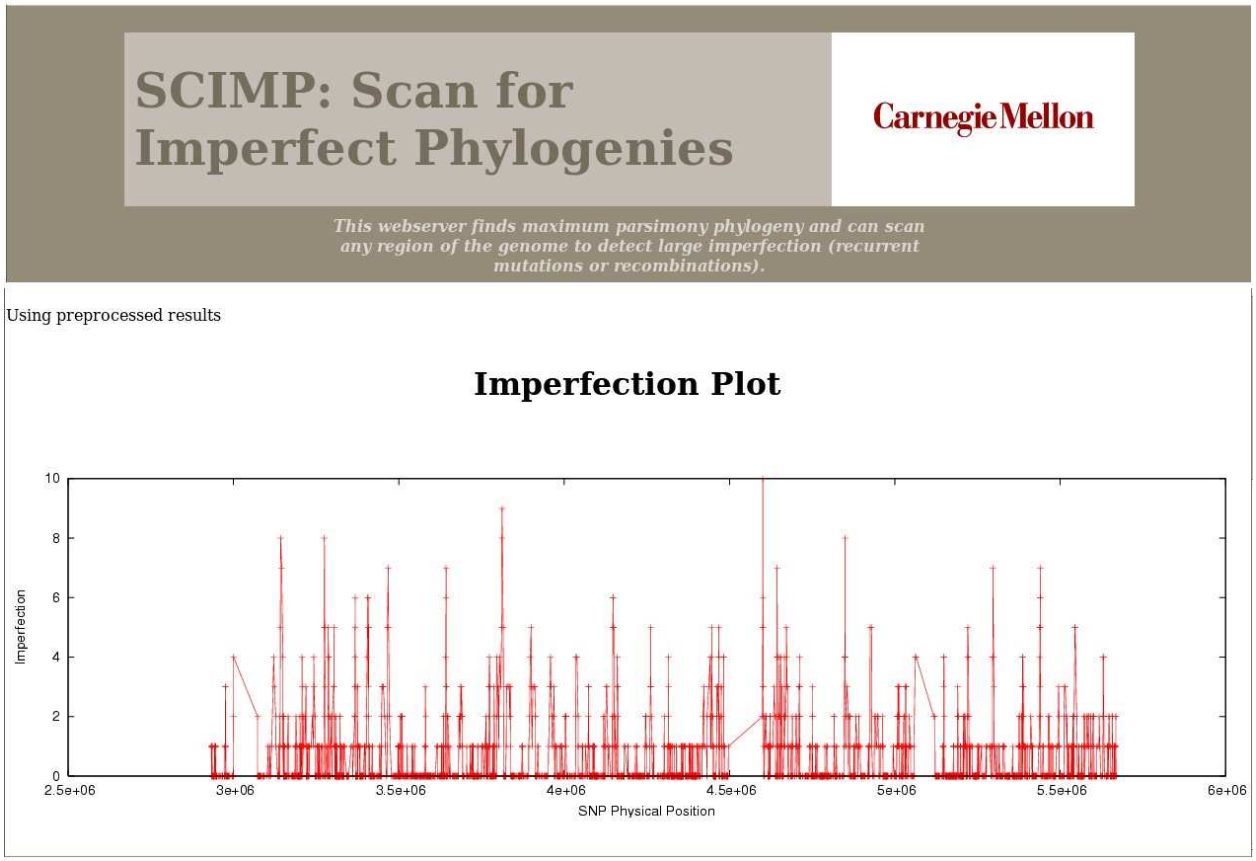


Figure A.2: Screenshot of scan for detecting phylogenetic imperfection using a window size of 5 from a part of Chromosome 21.

Therefore if there are n individuals and m SNPs, then the input should consist of n rows and $2m+1$ columns. The first column is alpha-numeric and subsequent columns are binary (unless they contain missing data).

Output Format. The output format is fairly simple. It consists of a set of lines, each of which contains an individual ID followed by a label: either a 0 or a 1. Therefore two individual IDs with the same label belong to the same sub-population and individuals with different IDs belong to different sub-populations. If the p-value option is used, then the webserver also prints the significance.

The website can be accessed at: `www.cs.cmu.edu/~triplets`

Population substructure (admixed populations)

Given a set of genotypes, the tool computes ancestries for each individual as a mixture of two or more populations. The prediction is more accurate if the allele frequencies from the ancestral populations are provided as well. The tool will soon be available online for download at: `http://lamp.icsi.berkeley.edu/lamp`

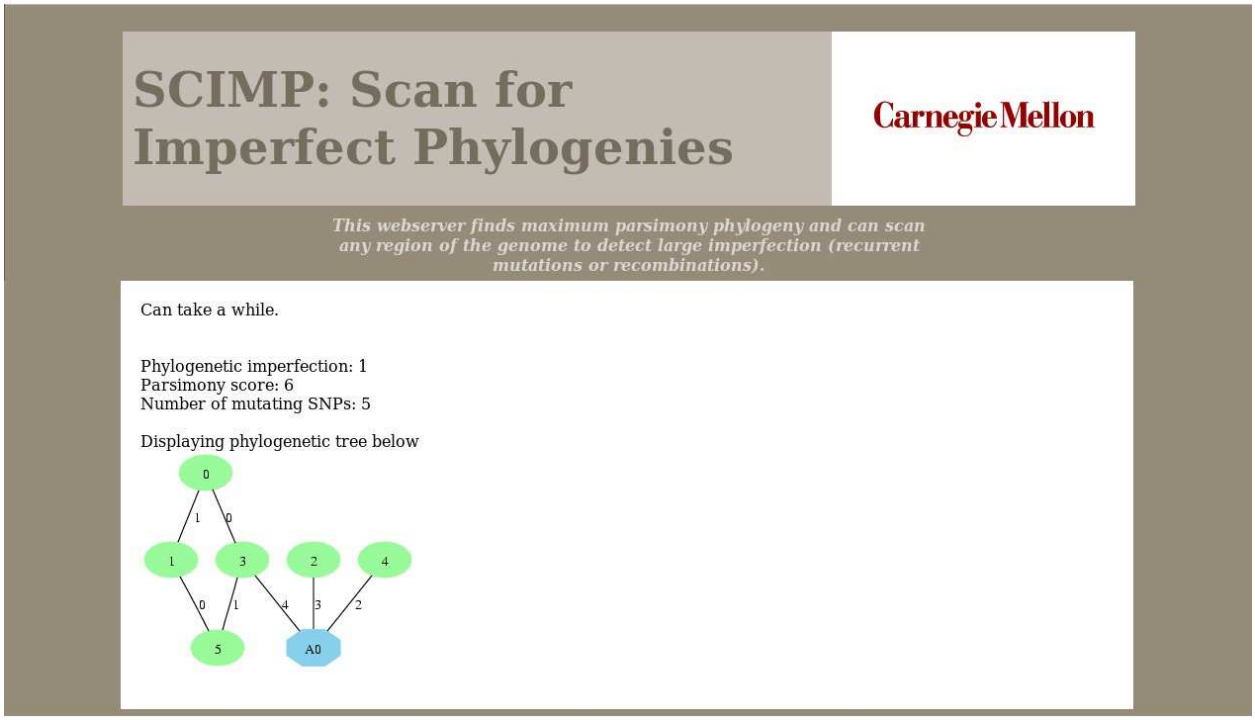


Figure A.3: Screenshot of the output containing a phylogenetic network produced as a union of all edges of every maximum parsimony phylogeny for a user input.

Bibliography

[1] H. S. Pedersen E. Angulalik E. D. Gunnarsdottir B. Yngvadottir A. Helgason, G. Palsson and K. Stefansson. mtDNA variation in Inuit populations of Greenland and Canada: migration history and population structure. *American Journal of Physical Anthropology*, 130:123–134, 2006. 3.6, 3.6

[2] R. Agarwala and D. Fernandez-Baca. A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM Journal on Computing*, 23:1216–1224, 1994. 2, 3

[3] V. Bafna, D. Gusfield, G. Hannenhalli, and S. Yooseph. A note on efficient computation of haplotypes via perfect phylogeny. *Journal of Computational Biology*, 11: 858–866, 2004. 4, 5

[4] V. Bafna, D. Gusfield, G. Lancia, and S. Yooseph. Haplotyping as perfect phylogeny: A direct approach. *ournal of Computational Biology*, 10:323–340, 2003. 13, 4, 5

[5] H. J. Bandelt, P. Forster, B. C. Sykes, and M. B. Richards. Mitochondrial portraits of human populations using median networks. *Genetics*, 141:743–753, 1989. 3, 3.2.1, 5, 5.1.2

[6] J. Barthélemy. From copair hypergraphs to median graphs with latent vertices. *Discrete Math*, 76:9–28, 1989. 3.2.1, 5.1.1

[7] J.E. Beasley. An algorithm for the Steiner problem in graphs. *Networks*, 14:147–159, 1984. 3.3

[8] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, 48(3):259–302, 1986. 8.1.3

[9] G. E. Blelloch, K. Dhamdhere, E. Halperin, R. Ravi, R. Schwartz, and S. Sridhar. Fixed parameter tractability of binary near-perfect phylogenetic tree reconstruction. *International Colloquium on Automata, Languages and Programming*, 2006. 2, 2.3, 3, 3.6, 4, 5, 6

[10] M. Bonet, M. Steel, T. Warnow, and S. Yooseph. Better methods for solving parsimony and compatibility. *Journal of Computational Biology*, 5:409–422, 1992. 5

[11] P. E. Bonnen, I. Pe'er, R. M. Plenge, J. Salit, J. K. Lowe, M. H. Shapero, R. P. Lifton, J. L. Breslow, M. J. Daly, D. E. Reich, et al. Evaluating potential for whole-genome studies in Kosrae, an isolated population in Micronesia. *Nat. Genet.*, 38: 214–217, 2006. 7

[12] R. Boppana. Eigenvalues and graph bisection: An average case analysis. *In proc IEEE Symposium on Foundations of Computer Science*, 1987. 7

[13] P. Buneman. The recovery of trees from measures of dissimilarity. *Mathematics in the Archeological and Historical Sciences, F. Hodson et al., Editors*, pages 387–395, 1971. 3.2.1, 5.1.1

[14] W. Buntine and A. Jakulin. Applying discrete pca in data analysis. *In proc Uncertainty in AI*, 2004. 7

[15] B. Lizarraga C. J. Lewis, R. Tito and A. Stone. Land, language, and loci: mtDNA in Native Americans and the genetic history of Peru. *American Journal of Physical Anthropology*, 127:351–360, 2005. 3.6, 3.6

[16] X. Cheng and D.Z. Du (Editors). *Steiner Trees in Industry*. Springer, 2002. 3.1

[17] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statist.*, 23:493–507, 1952. 8.3

[18] A. G. Clark. Inference of haplotypes form PCR-amplified samples of diploid populations. *Molecular Biology and Evolution*, 7:1111–122, 1990. 4, 5

[19] A. G. Clark, K. M. Weiss, D. A. Nickerson, S. L. Taylor, A. Buchanan, J. Stengård, V. Salomaa, E. Vartianinen, M. Perola, E¿ Boerwinkle, and C. F. Sing. Haplotype structure and population genetic inferences from nucleotide-sequence variation in human lipoprotein lipase. *Am J Hum Genet*, 63:595–612, 1998. 5.1.3, 5.2.3

[20] H. E. Collins-Schramm, B. Chima, D. J. Operario, L. A. Criswell, and M. F. Seldin. Markers informative for ancestry demonstrate consistent megabase-length linkage disequilibrium in the african american population. *Hum Genet*, 113(3):211–219, August 2003. 8.5

[21] International Human Genome Sequencing Consortium. *Nature*, 409:860–921. 1

[22] T. E. P. Consortium. The encode (encyclopedia of dna elements) project. *Science*, 306:636–640, 2004. 2

[23] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *International Workshop on Parameterized and Exact Computation*, 2004. 2

[24] W. H. Day and D. Sankoff. Computational complexity of inferring phylogenies by compatibility. *Systematic Zoology*, pages 224–229, 1986. 2

[25] Z. Ding, V. Filkov, and D. Gusfield. A linear time algorithm for perfect phylogeny haplotyping. *Research in Computational Molecular Biology*, 2005. 4, 4.2, 4.2, 5

[26] R. G. Downey and M. R. Fellows. Parameterized complexity. *Monographs in Computer Science*, 1999. 2.3.3

[27] M. Ward E. M. Smigielski, K. Sirotkin and S. T. Sherry. dbsnp: a database of single nucleotide polymorphisms. *Nucleic Acids Research*, 28:352–355, 2000. 2

[28] E. Eskin, E. Halperin, and R. M. Karp. Efficient reconstruction of haplotype structure via perfect phylogeny. *Journal of Bioinformatics and Computational Biology*, pages

1–20, 2003. 2.1, 4, 4.1, 4.2, 4.3, 4.4, 5

[29] A. Kasprzyk et. al. Ensmart: A generic system for fast and flexible access to biological data. *Genome Res.*, 14(1):160–169, January 2004. URL `http://dx.doi.org/10.1101/gr.1645104`. 6.1.1

[30] A. L. Price et. al. A genomewide admixture map for latino populations. *Am J Hum Genet*, 80(6), 2007. 8.6

[31] C. Tian et. al. A genome-wide snp panel for mexican american admixture mapping. *Am J Hum Genet*, 80(6), 2007. 8.6

[32] D. C. Thomas et al. Recent developments in genome wide association scans: a workshop summary and review. *American Journal of Human Genetics*, 77, 2005. 7, 7.5

[33] D. Karolchik et. al. The ucsc genome browser database. *Nucleic Acids Res*, 31(1):51–54, January 2003. ISSN 1362-4962. URL `http://view.ncbi.nlm.nih.gov/pubmed/12519945`. 6.1.1

[34] D. Reich et. al. A whole-genome admixture scan finds a candidate locus for multiple sclerosis susceptibility. *Nature Genetics*, 37(10):1113–1118, September 2005. 8

[35] E. J. Parra et. al. Estimating african american admixture proportions by use of population-specific alleles. *Am J Hum Genet*, 63(6):1839–1851, December 1998. 8.5

[36] F. A. Kurreeman et. al. A candidate gene approach identifies the traf1/c5 region as a risk factor for rheumatoid arthritis. *PLoS Medicine*, 2007. 1

[37] J. C. Venter et. al. *Science*, 291:1304–1351. 1

[38] K. Lindblad-Toh et al. Genome sequence, comparative analysis and haplotype structure of the domestic dog. *Nature*, 438:803–819, 2005. 2

[39] Lindblad-Toh et. al. Large-scale discovery and genotyping of single-nucleotide polymorphisms in the mouse. *Nature Genetics*, pages 381–386, 2000. 2

[40] M. D. Shriver et. al. Ethnic-affiliation estimation by use of population-specific dna markers. *Am J Hum Genet*, 60(4):957–964, April 1997. 8.5

[41] M. W. Smith et. al. A high-density admixture map for disease gene discovery in african americans. *Am J Hum Genet*, 74(5):1001–1013, May 2004. 8.6

[42] N. Patil et al. Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. *Science*, 2001. 4, 4.4

[43] N. Patterson et. al. Methods for high-density admixture mapping of disease genes. *Am J Hum Genet*, 74(5):979–1000, May 2004. 8, 8.5, 8.5

[44] R. C. Hardison et. al. Covariation in frequencies of substitution, deletion, transposition, and recombination during eutherian evolution. *Genome Resarch*, 13:13–26, 2003. 6.3

[45] T. J. Hubbard et. al. Ensembl 2007. *Nucleic Acids Res*, 35(Database issue), January 2007. ISSN 1362-4962. URL `http://view.ncbi.nlm.nih.gov/pubmed/17148474`. 6.1.1

[46] X. Mao et. al. A genome-wide admixture mapping panel for hispanic/latino populations. *Am J Hum Genet*, 80(6), 2007. 8.6

[47] X. Zhu et. al. Admixture mapping for hypertension loci with genome-scan markers. *Nat Genet*, 37(2):177–181, February 2005. 8

[48] D. L. Wheeler etl. al. Database resources of the national center for biotechnology information. *Nucleic Acids Research*, 35(Database issue), 2007. URL `http://view.ncbi.nlm.nih.gov/pubmed/17170002`. 6.1.1

[49] D. Falush, M. Stephens, and J. K. Pritchard. Inference of population structure using multilocus genotype data: linked loci and correlated allele frequencies. *Genetics*, 164 (4):1567–1587, August 2003. 8, 8.5, 8.5

[50] J. Felsenstein. PHYLIP (Phylogeny Inference Package) version 3.6. distributed by the author, department of genome sciences, university of washington, seattle. 2005. 2.4, 3, 3.5, 5

[51] D. Fernandez-Baca and J. Lagergren. A polynomial-time algorithm for near-perfect phylogeny. *SIAM Journal on Computing*, 32:1115–1127, 2003. 2, 2.1, 3, 3.6, 4, 5, 6

[52] D. Field and C. Wills. Abundandant microsatellite polymorphisms in saccharomyces cerevisiae and the different distributions of microsatellites in eight prokaryotes and s. cerevisiae, result from strong mutation pressures and a variety of selective forces. *Proc Natl Acad Sci*, 95:1647–1652, 1998. 6.2.3

[53] L. R. Foulds and R. L. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3, 1982. 2, 3.1, 4.1, 5

[54] C. Fraumene, E. M. S. Belle, L. Castrì, S. Sanna, G. Mancosu, M. Cosso, F. Marras, G. Barbujani, M. Pirastu, and A. Angius. High resolution analysis and phylogenetic network construction using complete mtDNA sequences in Sardinian genetic isolates. *Mol Biol Evol*, 23:2101–2111, 2006. 5.1.3, 5.2.2

[55] G. Ganapathy, V. Ramachandran, and T. Warnow. Better hill-climbing searches for parsimony. *Workshop on Algorithms in Bioinformatics*, 2003. 5

[56] M. R. Garey and D. S. Johnson. *Computers and Intractability (A Series of books in the mathematical sciences)*. 1979. 2, 3, 7.1.2

[57] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991. 2, 2.1.1, 2.1, 2.2, 2.3.3, 3, 3.2.2, 6

[58] D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1999. 2

[59] D. Gusfield. Haplotyping as a perfect phylogeny: Conceptual framework and efficient solutions. *Research in Computational Molecular Biology*, 2002. 4, 4.3, 5, 6

[60] D. Gusfield. Haplotyping by pure parsimony. *Combinatorial Pattern Matching*, 2003. 3, 5

[61] D. Gusfield. An overview of combinatorial methods for haplotype inference. *Lecture Notes in Computer Science*, 2983, 2004. 4

[62] D. Gusfield and V. Bansal. A fundamental decomposition theory for phylogenetic networks and incompatible characters. *Research in Computational Molecular Biology*, 2005. 2.3.3, 3.2.2, 3.2.2, 3.2.5, 4, 4.2

[63] D. Gusfield, S. Eddhu, and C. Langley. Efficient reconstruction of phylogenetic networks with constrained recombination. *IEEE Computer Society Bioinformatics Conference*, pages 363–374, 2003. 10

[64] J. B. S. Haldane. The combination of linkage values and the calculation of distance between the loci of linked factors. *J. Genet.*, 8:299–309, 1919. 8.1.1

[65] E. Halperin and E. Eskin. Haplotype reconstruction from genotype data using imperfect phylogeny. *Bioinformatics*, 2004. 4, 5, 5.1.3, 5.2.3, 9.1

[66] C. L. Hanis, R. Chakraborty, R. E. Ferrell, and W. J. Schull. Individual admixture estimates: disease associations and individual risk of diabetes and gallbladder disease among mexican-americans in starr county, texas. *Am J Phys Anthropol*, 70(4):433–441, August 1986. 8.5

[67] I. Hellmann, I. Ebersberger, S. E. Ptak, S. Paabo, and M. Przeworski. A neutral explanation for the correlation of diversity with recombination rates in humans. *Am J Hum Genet*, 72:1527–1535, 2003. 6.3

[68] D. A. Hinds, L. L. Stuve, G. B. Nilsen, E. Halperin, E. Eskin, D. G. Ballinger, K. A. Frazer, and D. R. Cox. Whole genome patterns of common DNA variation in three human populations, `www.perlegen.com`. *Science*, 2005. 1.1

[69] J. N. Hirschhorn and M. J. Daly. Genome-wide association studies for common diseases and complex traits. *Nat. Rev. Genet.*, 6:95–108, 2005. 7, 7.5

[70] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association(JSTOR)*, 58, 1963. 7.1.2, 7.1.2

[71] C. J. Hoggart, M. D. Shriver, R. A. Kittles, D. G. Clayton, and P. M. McKeigue. Design and analysis of admixture mapping studies. *Am J Hum Genet*, 74(5):965–978, May 2004. ISSN 0002-9297. 8, 8.5

[72] R. R. Hudson. Generating samples under a Wright-Fisher neutral model. *Bioinformatics*, 18:337–338, 2002. 5.1.3

[73] D. Huson, T. Klopper, P. J. Lockhart, and M. A. Steel. Reconstruction of reticulate networks from gene trees. *Research in Computational Molecular Biology*, 2005. 4

[74] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*, volume 53. Annals of Discrete Mathematics, 1992. 3.1

[75] The International HapMap Consortium. The international hapmap project. `www.hapmap.org`. *Nature*, 426:789–796, 2005. 1, 1.1, 1.3.1, 2, 2.4, 3.6, 3.6, 3.5, 6, 6.1.1, 6.3, 7, 7.2

[76] S. Rao K. Chaudhuri, E. Halperin and S. Zhou. Separating populations with small data. *To appear in proc Symposium on Discrete Algorithms (SODA)*, 2007. 7, 7.1.1, 7.5

[77] S. Kannan and T. Warnow. A fast algorithm for the computation and enumeration of perfect phylogenies. *SIAM Journal on Computing*, 26:1749–1763, 1997. 2, 3

[78] J. W. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and A. D. Haussler. The human genome browser at ucsc. *Genome Res.*, 12(6):996–1006, June 2002. URL `http://dx.doi.org/10.1101/gr.229102.Articlepublishedonlinebeforeprintin May2002`. 6.1.1

[79] N. Li and M. Stephens. Modeling Linkage Disequilibrium and Identifying Recombination Hotspots Using Single-Nucleotide Polymorphism Data. *Genetics*, 165(4): 2213–2233, 2003. 8.5

[80] E. Heinaru J. Truu M. Merimaa, M. Liivak and A. Heinaru. Functional co-adaption of phenol hydroxylase and catechol 2,3-dioxygenase genes in bacteria possessing different phenol and p-cresol degradation pathways. *Eighth Symposium on Bacterial Genetics and Ecology*, 31:185–212, 2005. 2.4, 3.6, 3.6

[81] N. Maculan. The Steiner problem in graphs. *Annals of Discrete Mathematics*, 31: 185–212, 1987. 3.3, 3.3

[82] F. McSherry. Spectral partitioning of random graphs. *In proc IEEE Symposium on Foundations of Computer Science (FOCS)*, 2001. 7

[83] G. A. McVean, S. R. Myers, S. Hunt, P. Deloukas, D. R. Bentley, and P. Donnelly. The fine-scale structure of recombination rate variation in the human genome. *Science*, 304(5670):581–584, April 2004. ISSN 1095-9203. URL `http://dx.doi.org/10.1126/science.1092500`. 6.2.2

[84] J. Meigs, A. Manning, C. Fox, J. Florez, C. Liu, L.A. Cupples, and J. Dupuis. Genome-wide association with diabetes-related traits in the framingham heart study. *BMC Med Genet*, 8 Suppl 1, 2007. 1

[85] S. Myers, L. Bottolo, C. Freeman, G. McVean, and P. Donnelly. A fine-scale map of recombination rates and hotspots across the human genome. *Science*, 310(5746): 321–324, October 2005. 6.2.2, 8.5

[86] M. W. Nachman and S. L. Crowell. Estimate of the mutation rate per nucleotide in humans. *Genetics*, 156(1):297–304, September 2000. 8.5

[87] T. Niu, Z. S. Qin, X. Xu, and J. Liu. Bayesian haplotype inference for multiple linked Single Nucleotide Polymorphisms. *American Journal of Human Genetics*, 2002. 4, 4.4, 5.2

[88] K. S. Pollard, S. R. Salama, B. King, A. D. Kern, T. Dreszer, S. Katzman, A. Siepel, J. S. Pedersen, G. Bejerano, R. Baertsch, J. Rosenbloom, K. R.and Kent, and D. Haussler. Forces shaping the fastest evolving regions in the human genome. *PLoS Genet*, 2(10), 2006. 1

[89] A. L. Price, N. J. Patterson, R. M. Plenge, M. E. Weinblatt, N. A. Shadick, and D. Reich. Principal components analysis corrects for stratification in genome-wide association studies. *Nature Genetics*, 38(8):904–909, July 2006. 7, 7.2, 8

[90] J. K. Pritchard, M. Stephens, and P. Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155(2):945–959, June 2000. 7, 7.2, 7.3, 8, 8, 8.1.2, 8.1.3, 8.5, 8.5, 9.1

[91] H. J. Promel and A. Steger. The Steiner tree problem: A tour through graphs algorithms and complexity. *Vieweg Verlag*, 2002. 2.2, 2.3.3

[92] L. Rabiner. A tutorial on hmm and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989. 8

[93] A. Rambaut and N. C. Grassly. Seq-gen: An application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Comput Appl Biosci*, 13: 235–238, 1997. 5.1.3

[94] B. Rannala and Z. Yang. Bayes estimation of species divergence times and ancestral population sizes using DNA sequences from multiple loci. *Genetics*, 164:1645–1656, 2003. 5.1.3

[95] G. Kimmel E. Halperin S. Sankararaman, S. Sridhar. Estimating local ancestry in admixed populations. *To appear in American Journal of Human Genetics*, 2008. 9.1

[96] E. Rai A. Bhat S. Sharma, A. Saha and R. Bamezai. Human mtDNA hypervariable regions, HVR I and II, hint at deep common maternal founder and subsequent maternal gene flow in Indian population groups. *American Journal of Human Genetics*, 50:497–506, 2005. 3.6, 3.6

[97] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987. 3

[98] P. Scheet and M. Stephens. A fast and flexible statistical model for large-scale population genotype data: applications to inferring missing genotypes and haplotypic phase. *Am J Hum Genet*, 78:629–644, 2006. 5, 5.2

[99] J. Sebat, B. Lakshmi, D. Malhotra, J. Troge, C. Lese-Martin, T. Walsh, B. Yamrom, S. Yoon, A. Krasnitz, J. Kendall, A. Leotta, D. Pai, R. Zhang, Y. Lee, J. Hicks, S.J. Spence, A.T. Lee, K. Puura, T. Lehtimki, D. Ledbetter, P.K. Gregersen, J. Bregman, J.S. Sutcliffe, V. Jobanputra, W. Chung, D. Warburton, M. King, D. Skuse, D.H. Geschwind, T.C. Gilliam, K. Ye, and M. Wigler. Strong association of de novo copy number mutations with autism. *Science*, 2007. 1

[100] C. Semple and M. Steel. *Phylogenetics*. Oxford University Press, 2003. 2, 2.1, 2.2, 2.3.3, 2.3.3, 3.2.1, 3.2.2, 3.2.2, 5.1.2

[101] T. C. Sequencing and A. Consortium. Initial sequence of the chimpanzee genome and comparison with the human genome. *Nature*, 437:69–87, 2005. 2

[102] S. T. Sherry, M. H. Ward, M. Kholodov, J. Baker, L. Pham, E. Smigielski, and K. Sirotkin. dbSNP: The NCBI Database of genetic variation. *Nucleic Acids Research*, 29:308–311, 2001. 2, 6.1.1

[103] A. F. A. Smit, R. Hubley, and P. Green. Repeatmasker open-3.0, 1996-2004. URL `http://www.repeatmasker.org`. 6.1.1

[104] Y. Song, Y. Wu, and D. Gusfield. Algorithms for imperfect phylogeny haplotyp-

ing with a single homoplasy or recombination event. *Workshop on Algorithms in Bioinformatics*, 2005. 4, 4.2, 4.3, 4.4, 5

[105] S. Sridhar, G. E. Blelloch, R. Ravi, and R. Schwartz. Optimal imperfect phylogeny reconstruction and haplotyping. *In proceedings of Computational Systems Bioinformatics (CSB)*, 2006. 3, 4, 5, 6, 6.1.1, 9.1

[106] S. Sridhar, K. Dhamdhere, G. E. Blelloch, E. Halperin, R. Ravi, and R. Schwartz. Simple reconstruction of binary near-perfect phylogenetic trees. *International Workshop on Bioinformatics Research and Applications*, 2006. 2, 2.4, 3, 3.1, 3.6, 4, 4.4, 5.1.2, 6

[107] S. Sridhar, K. Dhamdhere, G. E. Blelloch, E. Halperin, R. Ravi, and R. Schwartz. Algorithms for efficient near-perfect phylogenetic tree reconstruction in theory and practice. *ACM/IEEE Transactions on Computational Biology and Bioinformatics*, 2007. 2, 3.5, 5, 5.1.3, 5.1.1, 6

[108] S. Sridhar, F. Lam, G. E. Blelloch, R. Ravi, and R. Schwartz. Efficiently finding the most parsimonious phylogenetic tree via linear programming. *In proceedings International Symposium on Bioinformatics Research and Applications (ISBRA)*, 2007. 5

[109] S. Sridhar, S. Rao, and E. Halperin. An efficient and accurate graph-based method to detect population substructure. *Proceedings of Research in Computational Molecular Biology (RECOMB)*, 2007. 8.5, 9.1

[110] M. Stephens, N. Smith, and P. Donnelly. A new statistical method for haplotype reconstruction from population data. *American Journal of Human Genetics*, 2001. 4, 4.4, 5, 6.1.1

[111] A. C. Stone, R. C. Griffiths, S. L. Zegura, and M. F. Hammer. High levels of y-chromosome nucleotide diversity in the genus pan. *Proceedings of the National Academy of Sciences*, pages 43–48, 2002. 2.4, 3.6, 3.6

[112] P. Parham M. Slatkin T. Wiehe, J. Mountain. Distinguishing recombination and intragenic gene conversion by linkage disequilibrium patterns. *Genetics Research*, 75:61–73, 2000. 6.3

[113] B. Linz R. P. Novick J. K. Lum M. Blaser G. Morelli D. Falush T. Wirth, X. Wang and M. Achtman. Distinguishing human ethnic groups by means of sequences from helicobacter pylori: Lessons from ladakh. *Proceedings of the National Academy of Sciences*, 2004. 2.4, 3.6, 3.6, 3.6, 3.5, 5.2.2

[114] H. Tang, M. Coram, P. Wang, X. Zhu, and N. Risch. Reconstructing genetic ancestry blocks in admixed individuals. *Am J Hum Genet*, 79(1):1–12, July 2006. 8, 8.1.2, 8.5, 8.5

[115] D. Tautz and C. Schlotterer. Simple sequences. *Curr Opin Genet Devel*, 4:832–837, 1994. 6.2.3

[116] C. Tian, D. A. Hinds, R. Shigeta, R. Kittles, D. G. Ballinger, and M. F. Seldin. A genomewide single-nucleotide-polymorphism panel with high ancestry information

for african american admixture mapping. *Am J Hum Genet*, 79(4):640–649, October 2006. 8.5

[117] S. A. Tishkoff and B. C. Verrelli. Patterns of human genetic diversity: Implications for human evolutionary history and disease. *Ann Rev Genom Hum G*, 4:293–340, 2003. 5

[118] R.T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984. 3.3, 3.3, 5.1.1

[119] E. Ziv and E. G. Burchard. Human population structure and genetic association studies. *Pharmacogenomics*, 4(4):431–441, July 2003. 8.5