2006

Algorithms For Haplotype Inference And Block Partitioning

Satya Ravi Vijaya
University of Central Florida

STARS Citation

Vijaya, Satya Ravi, "Algorithms For Haplotype Inference And Block Partitioning" (2006). *Electronic Theses and Dissertations, 2004-2019*. 1104.
https://stars.library.ucf.edu/etd/1104

Algorithms for Haplotype Inference and Block Partitioning

by

Ravi Vijaya Satya
B. Tech. Jawaharlal Nehru Technological University, 1999
M.S. University of Central Florida, 2001

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2006

Major Professor: Amar Mukherjee

Abstract

The completion of the human genome project in 2003 paved the way for studies to better understand and catalog variation in the human genome. The International HapMap Project was started in 2002 with the aim of identifying genetic variation in the human genome and studying the distribution of genetic variation across populations of individuals. The information collected by the HapMap project will enable researchers in associating genetic variations with phenotypic variations.

Single Nucleotide Polymorphisms (SNPs) are loci in the genome where two individuals differ in a single base. It is estimated that there are approximately ten million SNPs in the human genome. These ten million SNPS are not completely independent of each other - blocks (contiguous regions) of neighboring SNPs on the same chromosome are inherited together. The pattern of SNPs on a block of the chromosome is called a haplotype. Each block might contain a large number of SNPs, but a small subset of these SNPs are sufficient to uniquely identify each haplotype in the block. The haplotype map or HapMap is a map of these haplotype blocks. Haplotypes, rather than individual SNP alleles are expected to effect a disease phenotype.

The human genome is diploid, meaning that in each cell there are two copies of each chromosome - i.e., each individual has two haplotypes in any region of the chromosome.

With the current technology, the cost associated with empirically collecting haplotype data is prohibitively expensive. Therefore, the un-ordered bi-allelic genotype data is collected experimentally. The genotype data gives the two alleles in each SNP locus in an individual, but does not give information about which allele is on which copy of the chromosome. This necessitates computational techniques for inferring haplotypes from genotype data. This computational problem is called the haplotype inference problem.

Many statistical approaches have been developed for the haplotype inference problem. Some of these statistical methods have been shown to be reasonably accurate on real genotype data. However, these techniques are very computation-intensive. With the international HapMap project collecting information from nearly 10 million SNPs, and with association studies involving thousands of individuals being undertaken, there is a need for more efficient methods for haplotype inference.

This dissertation is an effort to develop efficient *perfect* phylogeny based combinatorial algorithms for haplotype inference. The perfect phylogeny haplotyping (PPH) problem is to derive a set of haplotypes for a given set of genotypes with the condition that the haplotypes describe a perfect phylogeny. The perfect phylogeny approach to haplotype inference is applicable to the human genome due to the block structure of the human genome.

An important contribution of this dissertation is an optimal $O(nm)$ time algorithm for the PPH problem, where n is the number of genotypes and m is the number of SNPs involved. The complexity of the earlier algorithms for this problem was $O(nm^2)$. The $O(nm)$ complexity was achieved by applying some transformations on the input data and by making

use of the FlexTree data structure that has been developed as part of this dissertation work, which represents all the possible PPH solution for a given set of genotypes.

Real genotype data does not always admit a perfect phylogeny, even within a block of the human genome. Therefore, it is necessary to extend the perfect phylogeny approach to accommodate deviations from perfect phylogeny. Deviations from perfect phylogeny might occur because of recombination events and repeated or back mutations (also referred to as homoplasy events). Another contribution of this dissertation is a set of fixed-parameter tractable algorithms for constructing near-perfect phylogenies with homoplasy events. For the problem of constructing a near perfect phylogeny with q homoplasy events, the algorithm presented here takes $O(nm^2 + m^{q+1}(n + m))$ time. Empirical analysis on simulated data shows that this algorithm produces more accurate results than PHASE (a popular haplotype inference program), while being approximately 1000 times faster than phase.

Another important problem while dealing real genotype or haplotype data is the presence of missing entries. The Incomplete Perfect Phylogeny (IPP) problem is to construct a perfect phylogeny on a set of haplotypes with missing entries. The Incomplete Perfect Phylogeny Haplotyping (IPPH) problem is to construct a perfect phylogeny on a set of genotypes with missing entries. Both the IPP and IPPH problems have been shown to be NP-hard. The earlier approaches for both of these problems dealt with restricted versions of the problem, where the root is either available or can be trivially re-constructed from the data, or certain assumptions were made about the data. We make some novel observations about these problems, and present efficient algorithms for unrestricted versions of these problems. The

algorithms have worst-case exponential time complexity, but have been shown to be very fast on practical instances of the problem.

To the unfortunate millions that are denied the right to basic education

Acknowledgments

First and foremost, I would like to thank my advisor Prof. Amar Mukherjee. This dissertation would not have been possible without his support and guidance. I thank him for allowing me to pursue topics in bioinformatics and computational biology. He gave me a lot of freedom in selecting the problems that I wanted to work on. He encouraged to me to take an independent approach to these problems. All these factors were critical in obtaining the results that I have achieved throughout the course of this dissertation. His expectations for the highest quality of work pushed me to do my best.

I thank all the committee members, Dr. Chris Parkinson, Dr. Annie Wu, Dr. Mark Heinrich, Dr. Guy Schiavone, and Dr. Eric Hoffman for their support and encouragement. Special thanks are due to Dr. Parkinson for clarifying many concepts in molecular biology and evolutionary biology. Special thanks are also due to Dr. Hoffman for closely working with me on a grant proposal and bringing to my notice related papers that I had previously overlooked. I thank my cousin, Dr. Uday Kumar Ranga, for introducing me to the codon optimization problem, which led to my first publication in bioinformatics.

I acknowledge funding from NSF grant no. IIS 0312724 which supported my initial research in data compression, and continued to support my early work in bioinformatics.

I thank Dr. Ajay Royyuru of IBM T. J. Watson Research Center for giving me the opportunity to work on the genographic project as a summer intern. Above all, this internship gave me the opportunity to work with Dr. Gabriela Alexe, Dr. Gyan Bhanot, and Dr. Laxmi Parida, all of whom helped me in understanding many concepts in phylogenetics and population genetics. The experience and exposure I obtained while working at IBM helped me look into new problems, and ultimately led to some very good publications. Gabriela and Gyan have proved to be very good friends. They continue to provide encouragement, guidance, and valuable advice about critical career decisions.

I would not have been able to pursue a Ph.D. without the help of Dr. Guy Schiavone. I am indebted to him for funding me for more than three years. He always believed in me and encouraged me to pursue a Ph.D. when I had no intentions to do so. The experience I gained while working under him is invaluable. I also thank Mr. Art Cortes of IST and Dr. David Hagan of CREOL, for funding me at different stages of my graduate career.

My aunt Ranganayakamma and my uncle B.R. Bapuji have been a great inspiration to me in my life. My aunt was my first teacher. She taught me everything from English grammar to basic science and math. She taught me critical thinking and problem solving skills that help me to this day in my work. I could not have reached this far without her training.

Acknowledgements are also due to all the friends, colleagues and lab mates that provided encouragement and support. I thank fellow graduate students Nan Zhang, Tao Tao and Weifeng Sun for valuable discussions and the publications that I co-authored with them. Friends Sridher Sathi and Sameer Joshi have provided invaluable support and encouragement. I could always count on them for help, and I thank them for their support during rough times in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Since ancient times, it was common knowledge that a child inherits features from its parents. Beginning with Mendel's experiments, we have been trying to understand how inheritance works. There are many aspects to inheritance, and we now know quite a lot about some of them. For example, we know that the genetic information is carried through DNA. We know that there are 23 pairs of chromosomes. We even know the DNA sequence for all the chromosomes to an acceptable level of accuracy.

Though all humans have almost the same DNA sequence, each person is unique. The exact DNA sequence of a person is different from that of any other person (except from that of an identical twin). These differences arise mostly due to mutation and recombination events that occur in the germ line of the individual. We understand that genetic variation is the basic source of diversity in the phenotype - the color of the skin, the shape of the nose, how the body reacts to stimuli in the environment, the risk of developing a cancer, etc. Studying genetic variation will help in understanding, diagnosis and treatment of many diseases. The human genome is three billion base pairs long, so there are uncountable number of ways in which the DNA sequence of two people can differ. So how do we correlate genetic variation with variation in the phenotype? A good place to start will be the genetic polymorphisms that are common to many individuals. The variations in the phenotype caused by these common polymorphisms will be much easier to identify and analyze.

Throughout history of mankind, infectious diseases had the most devastating effect on human populations. In the past hundred or so years, rapid progress in medicine has helped lessen the impact of infectious diseases on human populations. The mankind has succeeded in controlling many infectious diseases like polio and smallpox. Infectious diseases like AIDS and malaria continue to kill millions of people every year in developing countries. However, failure in combating these diseases is more due to socio-economic factors than due to the lack of technological capability to combat these diseases. Effective treatments and preventive measures are available for these diseases, and some of these diseases have either been wiped out or contained in the developed world.

Genetic factors are believed to play a major role in most common non-infectious diseases today, like cancer, diabetes and obesity. Identifying the genetic variations associated with these diseases will eventually revolutionize the treatment for many of these genetically inherited diseases. In the near future, it might be possible to develop customized treatments for at least some of these diseases that are targeted for the specific phenotype of the individual, and hence are more likely to succeed in combating the disease.

Due to change in life styles and increase in life expectancy, genetically inherited diseases are increasingly becoming a major concern in both developed and the developing parts of the world. Evolution, via natural selection, is successful in purging genetically inherited diseases that affect the early (pre-reproductive) stages of life of an organism. However, late-onset genetically inherited diseases are largely unaffected by evolution. With rapid increase in life expectancy, an increasingly larger percentage of the population are having to face these genetically inherited diseases that were mostly unaffected by evolution.

There are many instances where a specific genetic variation has been associated with a particular genetic risk. The following are a couple of instances.

- A gene in chromosome 17, named *Breast Cancer 1* (BRCA1) was discovered in 1994 through genetic analysis of families affected by hereditary breast cancer. Two mutations in BRCA1 gene were later associated with increased risk of breast cancer. A mutation in the gene BRCA2 in chromosome 13 has also been associated with increased risk of breast cancer. However, it is not yet completely understood how these mutations lead to breast cancer. For example, only 15% of the women with the mutations in the BRCA1 gene actually develop breast cancer. The mutations in BRCA1 appear in only 5% of the individuals that develop breast cancer [Law06].

- An extremely rare mutation in the CDH1 gene has been associated with a high risk of stomach cancer. According to a news article in Washington Post that appeared on June 18th, 2006 [CR06], this mutation was first discovered in 1998 in a large New Zealand family with a history of stomach cancer. The article reports about an extended family in United States that is affected by stomach cancer. The mutation could be traced to a woman in the family who died in 1960 due to stomach cancer. Seven of her children inherited the mutation, six of whom died with stomach cancer in their 40s and 50s. One of the eighteen grand children of the woman died of stomach cancer in 2003. The remaining 17 grand children got tested for the mutation and 11 of them tested positive. All 11 of them chose to have their stomachs removed to avoid the risk of stomach cancer.

1.1 Building A Haplotype Map of the Human Genome

The completion of the human genome project in 2003 paved the way for studies to better understand and catalog polymorphisms in the human genome. The International HapMap Project (www.hapmap.org) was started in 2002 with the aim of identifying polymorphisms in the human genome and studying the distribution of these polymorphisms both within the genome of an individual, and across populations. The information collected by the HapMap project will enable researchers in finding the polymorphisms that are the source for phenotypic variation.

The most common types of genetic variations are Single Nucleotide Polymorphisms (SNPs). SNPs are sites in the genome where individuals differ in a single base. It is estimated [Int05] that there are as many as 10 million SNPs in the human genome, which translates to a density of one SNP every three hundred base pairs of DNA. Blocks of neighboring SNPs on the same chromosome are inherited together. The pattern of SNPs on a block or continuous region of the chromosome is called a *haplotype*. Each block might contain a large number of SNPs, but a small subset of those SNPs are sufficient to uniquely identify each haplotype in the block. The haplotype map (or *HapMap*) is a map of these haplotype blocks. The SNPs that uniquely identify a block are called as haplotype *tag* SNPs, also referred to as htSNPs.

The objective of the HapMap project is not to draw associations between genetic polymorphisms and disease phenotypes, but to make these association studies feasible. Identifying the blocks and tag SNPs is essential for making the disease association studies feasible. According to the HapMap web site (www.hapmap.org), the number of tag SNPs for the 10 million SNPs in the human genome is expected to be around 500,000. This twenty-fold reduction in the number of SNPs will reduce the cost of disease association studies. This will also enable the association studies to be more comprehensive, since the association studies can cover all regions of the genome due to reduced costs.

The fundamental approach of association studies is to compute the haplotype frequencies in individuals with a specific phenotype and individuals without the specific phenotype (controls). The 'phenotype' can be a disease, response to a drug or susceptibility to an infection, among other things. The association studies are based on the assumption that the genetic variations that have some effect on the phenotype occur more frequently in individuals with the phenotype than in the individuals without the phenotype. Using just the tag SNPs, biomedical researchers will be able to identify regions within each chromosome that exhibit different haplotype frequencies in individuals with the phenotype and individuals without the phenotype. These regions can then be examined more closely to identify the specific genetic variations that cause the phenotype. This in turn, will help in developing tests and drugs that are targeted for the individuals with a specific genetic variation.

1.1.1 Populations studied by the HapMap project

The HapMap project is collecting SNP data from 270 individuals belonging to four different populations/ethnicities. The 270 individuals are distributed among the following four populations.

1. Ninety Yoruba individuals from Ibadan, Nigeria (YRI). This dataset consists of 30 trios. Each trio consists of three related individuals - two parents and an adult child. All these individuals belong to a single community in Ibadan, Nigeria. All the individuals selected had four Yoruba grand parents.

2. Ninety individuals of European origin (CEU). This dataset consists of 30 trios from Utah with northern and western European ancestry. These samples were collected by the Centre d'Etude du Polymorphisme Humain (CEPH) in 1980.

3. Forty-five individuals from Tokyo, Japan (JPT). These are forty-five unrelated individuals from Tokyo. Each individual selected had all the four Japanese grand parents.

4. Forty-five Han Chinese from Beijing (HCB). These are forty-five unrelated individuals living in the residential community of Beijing Normal University. These are all in-

dividuals who described themselves as having at least three out of four Han Chinese grandparents.

The first phase of the HapMap project was completed in March 2005. The first phase covered approximately one million (1,007,329) SNPs. The SNPs were evenly spaced across the whole genome, except on chromosome Y and mtDNA. The second phase of the HapMap project, which covers an additional 5.6 million SNPs, is currently underway. As of June 2006, data from approximately 4 million of these SNPs is available for download from the HapMap web site.

1.2 SNPs and Haplotypes in the Human Genome

1.2.1 SNPs

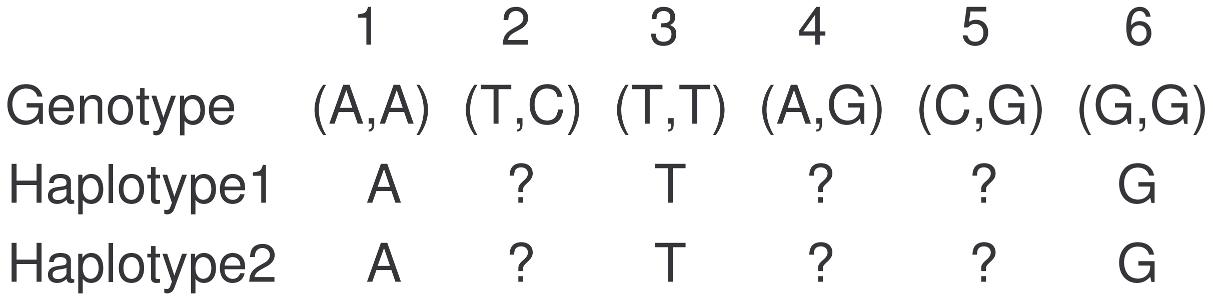
Genetic polymorphisms can be of many different types. They can be anything from a single nucleotide being different to having an extra copy of an entire chromosome. The most common, and the most significant, type of genetic variation is a Single Nucleotide Polymorphism (SNP). A SNP(pronounced 'snip') is a location in the human genome where a significant percentage (at least 2%, 5%, or 10%, depending on what is considered 'significant') of the population has a different nucleotide base than the rest of the population. For instance, 2% of all people might have a 'C' in a certain SNP location, whereas the rest have some base other than ‘C’ in that location. Each individual variation at a particular location is called an *allele*. Most of the SNPs are *bi-allelic*, meaning there are only two possible variations at that particular location. If more than two variations are possible in a particular location, the location is called *multi-allelic*. SNPs are quite common in the human genome - it is estimated that on average, there is one SNP for every 1200 base pairs [Hap03]. That comes to approximately 10 million SNPs in the whole genome.

1.2.2 Haplotypes in the human genome

The human genome is diploid, meaning that in each cell there are two copies of each chromosome. Due to the bi-parental nature of heredity in diploid organisms, one of these copies is derived from the mother and the other is derived from the father. Understandably, the two copies are not completely identical, as they are derived from two different individuals. In a given SNP location, the two copies of the chromosome in an individual may or may not have the same allele. If the two copies do have the same allele in an individual, the SNP location is said to *homozygous* in that individual. If the two copies have different alleles, the location is said to be *heterozygous*.

A single SNP variation may not be responsible for any given phenotype. Rather, it might be a particular pattern over multiple SNPs that causes the phenotype. Therefore, we are interested in knowing the state of all the SNPs in a region of the chromosome. As described before, a Haplotype is the pattern of SNPs on a single copy of the chromosome. i.e., in any region of a chromosome, each individual will have two haplotypes - one haplotype on each of the two copies of the chromosome.

Obtaining the haplotype information is essential in associating a haplotype with a disease/variation in the phenotype. However, obtaining the haplotype information involves isolating each copy of the chromosome, which is an expensive procedure, especially when thousands and thousands of individuals need to be analyzed. Therefore, the *conflated* information about the two copies of the chromosome is collected. This gives us an un-ordered pair of alleles at each location. We will know the two alleles at each SNP site in the individual, but we will not know which allele comes from which copy of the chromosome. For example, if the two possible alleles at a particular SNP location are A and C, we will know if the two alleles for that SNP in an individual are (A,A), (A,C) or (C,C). This information is called the *genotype* of the individual. Obtaining the haplotype information from the genotype information at a particular site is easy if the site is homozygous. However, if the site is heterozygous, we cannot tell which allele comes from which copy of the chromosome. For example, refer to Figure 1.1. Loci 1, 3 and 6 in the genotype are homozygous, and hence the two alleles in each haplotype must be as shown. However, for positions 2, 4, and 5, it is not clear from the genotype data which allele belongs to which haplotype.



	1	2	3	4	5	6
Genotype	(A,A)	(T,C)	(T,T)	(A,G)	(C,G)	(G,G)
Haplotype1	A	?	T	?	?	G
Haplotype2	A	?	T	?	?	G

Figure 1.1: Ambiguity in phasing a genotype

1.2.3 Block structure of the human genome

Recent studies [DRS01, PBH03, GSN02, WP03] have shown that the human genome can be divided into blocks of limited diversity. The haplotypes within each block can be represented by a subset of the SNPs that are covered by the block. It has been observed that there are regions within which there is strong association among the SNPs. This association is assessed as the degree of *linkage disequilibrium* (LD) between pairs of SNPs. There are various measures such as D' and r^2 [Hud01] for calculating the LD between a pair SNPs. Due to these regions of high LD, the number of haplotypes within a block is much smaller than the number of possible haplotypes in the block. Identifying these blocks of high LD, or *Block partitioning*, reduces the dimensionality of problems in disease association, and hence is essential in making many of the disease association studies feasible. It is important to correctly assess these linkage (haplotype) blocks because they may be tightly associated with regions of the genome influenced by positive selection (such as selective sweeps) or negative selection and disease association [Cla04]. Further more, block-partitioning enables identification of a smaller set of representative SNPs (haplotype tag SNPs or 'htSNPs') that describe a block unambiguously. Therefore, collecting genotype data for these representative SNPs will be sufficient for any association study. To minimize the costs of these association studies, it is necessary to identify the minimal set of tag SNPs for each block. Statistical and combinatorial methods are then used to associate the haplotypes with diseases.

1.3 The Haplotype Inference Problem

If k SNP sites are heterozygous in a given genotype, 2^{k-1} distinct pairs of haplotypes are possible that result in the same genotype. In other words, the genotype can have 2^{k-1} possible *explanation*s. Each explanation can be called a *phasing* of the genotype. The question is - which one of these explanations is the most 'accurate' for the given genotype? If we have a single genotype to deal with, all the 2^{k-1} haplotype pairs are equally likely, and we have no way of telling which one of these haplotype pairs is an 'accurate' explanation of the given genotype. However, if we have multiple genotypes, we can use information from the other genotypes to limit the possibilities for this genotype. The *Haplotype Inference* (HI) problem deals with finding the 'correct' explanation out of all these possible explanations.

Studies [GW02, RCB01, HSN05] have shown that the actual observed diversity within any region of a chromosome is much less than what we can expect from the number of SNPs covered by that region. Therefore, we expect many haplotypes to be common to many of the individuals. i.e, if there are 100 genotypes, we expect to see a lot fewer than 200 distinct haplotypes in the set of haplotypes that explain all the given genotypes. Therefore, given a population, we should obtain the smallest set of distinct haplotypes that explain all the individuals in the population. However, this problem was proven to be NP-complete [Gus01].

Sometimes, parent-child relationships between the individuals are available. This is called the *pedigree* of the individuals. When available, the pedigree data helps in disambiguating, (or phasing) some SNP locations. But the problem is NP-hard even when the pedigree data is available [LJ03].

1.3.1 The Coalescent Model

It is possible to obtain a more reasonable and efficient solution by assuming a biological model. Hence, Gusfield [Gus02] proposed application of the *coalescent* model to the haplotype inference problem. The coalescent model assumes that the evolutionary history of all the haplotypes in the population can be explained by a rooted tree, where each haplotype labels a vertex in the tree. The *infinite sites* model is also assumed, which stipulates that the number of sites is so large, and the frequency of mutation so small, that it is impossible for the same site to mutate more than once in the recent evolutionary history under consideration. This formulation of the haplotype inference problem is called as the Perfect Phylogeny Haplotyping (PPH) problem.

This dissertation solves many open problems related to the perfect phylogeny approach to the haplotype inference problem.

1.4 Contributions of this dissertation

The primary focus of this dissertation is to develop new, efficient combinatorial approaches for haplotype inference, based on the perfect phylogeny approach. Different statistical and combinatorial algorithms and approaches do exist for haplotype inference, but there are situations in which the nature of true genotype data renders some of these techniques inadequate, and in some cases, not applicable. One of the challenges when dealing with real genotype data is that as high as 10% of the data might be missing [HK04]. The existing combinatorial techniques for dealing with the missing data [KS05, HK04, PPS04] are applicable only in specific scenarios, and in some cases entirely rely on the availability of a large number of genotypes without any missing data. A drawback of the existing block partitioning approaches is that they rely heavily on the empirical observations like those made in [DRS01] that each block has no more than 4 or 5 common haplotypes. The approach presented in [HK04] rejects a block if the block has more than 5 common haplotypes. These parameters are highly sensitive to the size and ethnicity of the population considered. As the number of genotypes in the study increases, many blocks might not fit into the rather restrictive definition of block described above. Hence, there is a necessity to develop robust techniques that are not sensitive to the input sample size or ethnicity.

One of the fundamental contribution of this dissertation is extending the applicability of the perfect phylogeny approach to haplotype inference by incorporating *imperfect* and incomplete perfect phylogenies. The ability to construct imperfect phylogenies and the ability to handle missing data are essential in making the combinatorial approaches applicable to real genotype data. The ultimate goal is to enable incorporation of combinatorial methods into statistical approaches for haplotype inference. Empirical analysis on simulated data demonstrates that the combinatorial algorithms are faster and highly accurate when compared to statistical algorithms. Therefore, incorporating these combinatorial algorithms into statistical approaches is expected to improve their accuracy and performance.

1.4.1 Necessity for faster algorithms

Faster algorithms for haplotype inference are not just of theoretical interest. The HapMap project will ultimately collect data from 10 million SNPs in 270 individuals. The existing algorithms will be inadequate for dealing with problems of this magnitude. For example, the PHASE program [SSD01] takes nearly 15 minutes to phase 100 genotype over 100 loci. Even if the program were to scale linearly with n and m, this means that the program will take nearly three hundred days to phase the entire HapMap database. Any improvement in speed is certainly desirable.

The problems are expected to get bigger in the near future, when data from genome-scale association studies starts becoming available. For instance, the Framingham heart study plans to collect data from nine thousand individuals [Nat06] covering 500,000 SNPs in each individual. If the exponential growth of sequence databases over the past decade is an indication, the SNP databases will also experience an exponential growth in the coming years.

1.4.2 Significant results obtained

It is important to note that the contributions of this dissertation are not limited to the haplotype inference problem. This dissertation also presents some fundamental results in phylogenetic reconstruction.

The following are the major contributions of this dissertation:

- **Linear algorithm for the PPH problem.** An optimal algorithm for the PPH problem presented in this dissertation is among the first linear-time (in terms of the input) solutions for the PPH problem. This dissertation introduces the FlexTree data structure, which allows the representation of all the perfect phylogenies for a given PPH instance. The optimal algorithm utilizes the properties FlexTree data structure to achieve the linear-time complexity.

- **Algorithms for constructing near-perfect phylogenies.** Polynomial time algorithms for constructing near-perfect phylogenies with homoplasy events on both haplotype and genotype data. The algorithms on genotype data can be used to infer haplotypes when the input data does not admit a perfect phylogeny.

- **Algorithms for constructing incomplete perfect phylogenies.** Necessary and sufficient conditions for a given set of incomplete haplotypes or genotypes to allow a perfect phylogeny. These conditions, which were previously unknown, greatly simplify phylogenetic analysis on haplotypes or genotypes with missing data. New, efficient algorithms for constructing perfect phylogenies on both haplotype and genotype data with missing entries have been presented based on these conditions.

- **A new algorithm for optimal block-partitioning.** A new algorithm for selecting optimal block partitioning has been developed as part of this dissertation work. This new block-partitioning algorithm can be used with various optimization criteria like minimizing the number of tagSNPs, maximizing coverage, etc.

- **Experimental results on simulated data.** Empirical analysis on simulated data shows that the haplotype inference algorithms above are faster than existing methods. Comparison of the near-perfect phylogeny and incomplete perfect phylogeny algorithms with existing statistical algorithms shows that these algorithms have better accuracy in spite of being orders of magnitude faster than the statistical algorithms.

1.4.3 Organization of this dissertation

Chapter 2 introduces some relevant terminology, and presents some fundamental results in phylogeny construction. A formal description of the haplotype inference problem and the perfect phylogeny haplotyping problem are introduced.

Chapter 3 deals with perfect phylogeny haplotyping problem on complete genotype data. A review of the previous work on the perfect phylogeny haplotyping problem is provided. The FlexTree data structure and an optimal algorithm for the HI problem and the PPH problem are presented. A performance of analysis of the optimal algorithm with the previous algorithms is presented.

Chapter 4 deals with constructing near-perfect phylogenies on haplotypes and genotypes. Previous work on these problems is presented. New practical formulations of the problems are introduced, and fixed parameter tractable algorithms are presented for these problems. Performance of these algorithms on simulated data is compared with that of the PHASE program.

Chapter 5 introduces the problem of constructing perfect phylogenies on incomplete haplotype and genotype data. Previous results on these problems are presented. New algorithms that do not make any assumptions about the input data are presented.

Chapter 6 discusses some issues in applying the perfect phylogeny based algorithms on real genotype data. Some open problems in this area that need to be solved are presented, along with possible future directions of research.

CHAPTER 2

CONCEPTS AND TERMINOLOGY

This chapter introduces some fundamental concepts and terminology that will be used throughout this dissertation. A detailed, formal description of the haplotype inference problem is presented.

2.1 Molecular Biology Basics

The genetic information of an organism is given by the DNA (DeoxyriboNucleic Acid) of the organism. The DNA is a *polymer*, a long chain of molecules connected to each other. Each unit of DNA is called a *nucleotide*, and consists of two parts - a sugar called as deoxyribose, and a base. There are four different bases - Adenine(A), Guanine(G), Cytosine(C) and Thymine(T). Each nucleotide in the DNA is connected to the next through a *covalent* bond, called the phospho-diester bond. Such a chain of nucleotides is called a *strand* of DNA. However, DNA seldom exists in single-stranded form. Most of the DNA exists in a double-stranded form - two strands of the DNA are connected to each other through a series of weak hydrogen bonds. The hydrogen bonds that connect the two strands are only possible between two pairs of bases: either between A and T or between C and G. Hence, the two strands in any double stranded DNA are complementary two each other. If we know the sequence of bases on one strand, the other strand can be deduced through the complementarity between the (A,T) and (G,C) pairs.

The double-stranded DNA that is present in each cell of the organism is called as the *genome* of the organism. Through evolution, higher organisms have found it convenient to break up their genomes into smaller pieces, rather than having the entire genome as a single contiguous string. Each piece is called a chromosome. Most of the cells in an organism actually have two copies of each chromosome - and are called *diploid.* One copy of the chromosome is derived from the mother, while the other is derived from the father. The two copies of each chromosome are mostly similar, but not exactly identical to each other.

The human genome is divided into 23 pairs of chromosomes. Out of these, 22 pairs are called *autosomes* - both men and women have two copies of these chromosomes. The autosomes are numbered from 1 to 22. There are two other chromosomes, X and Y. The 23rd pair in women consists of a two copies of the X chromosome. The 23rd pair in men consists of a copy of the X chromosome and a copy of the Y chromosome. X and Y are called as the sex chromosomes. In each pair, the two copies are attached to each other through a protein complex. The length of each chromosome is fixed and is different from that of any other chromosomes. A schematic representation of the 24 different chromosomes is shown in Figure 2.1.

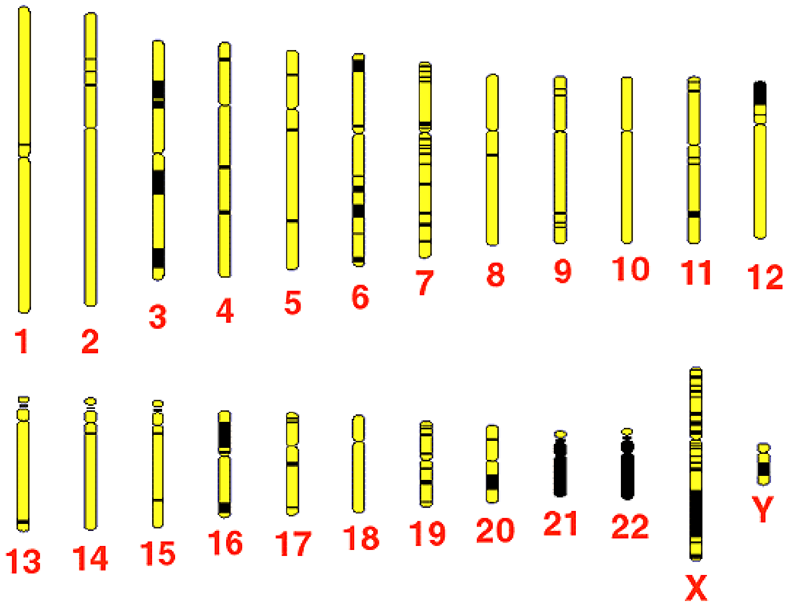


Figure 2.1: The relative lengths of the twenty four chromosomes

2.1.1 Meiosis

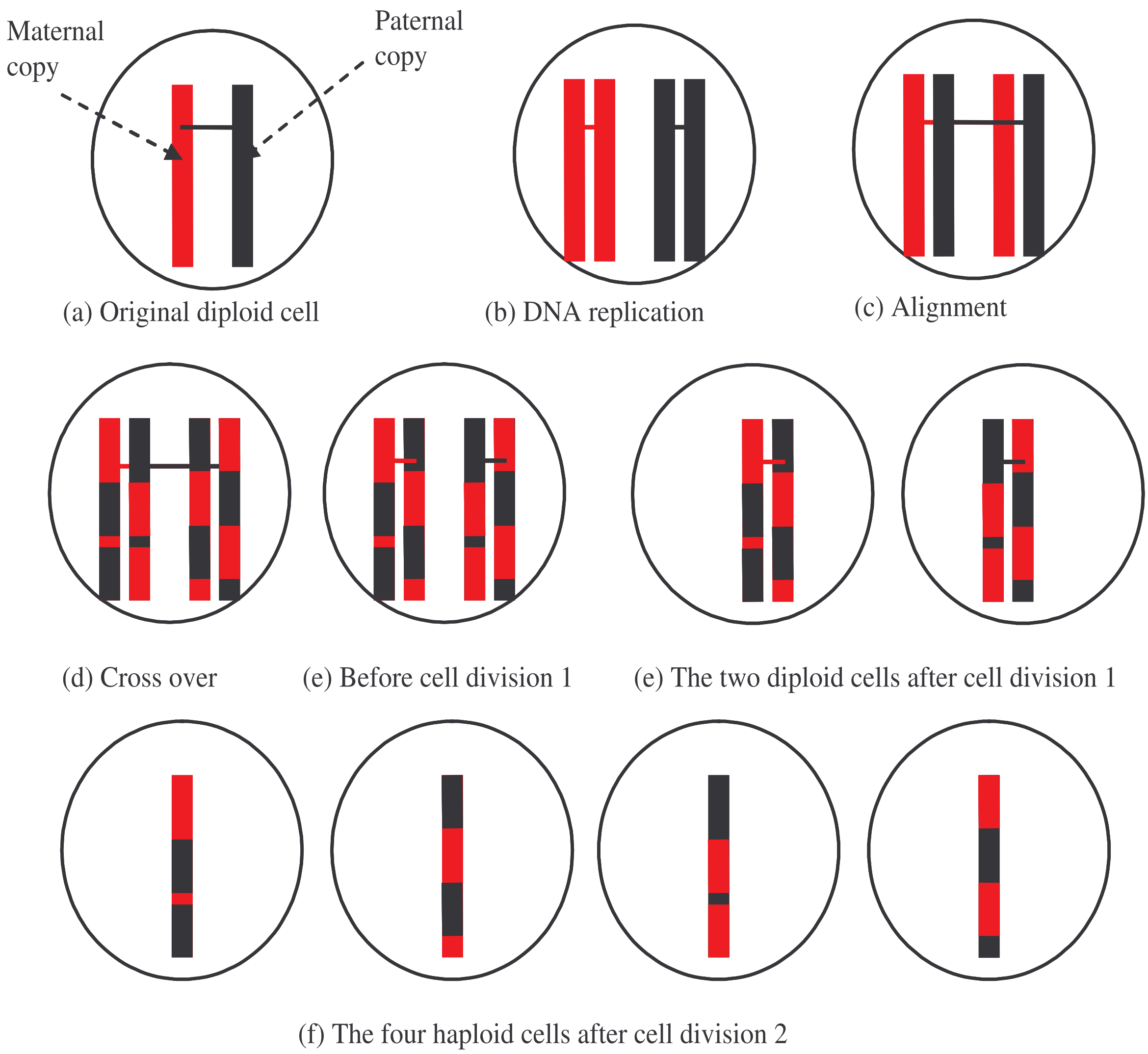
During reproduction, the diploid cells undergo a special process of cell division, and produce cells that have a single copy of each chromosome. This process is called *meiosis*. The cells produced through this process are called *haploid* cells, since they have only a single copy of each chromosome. These cells participate in reproduction, and are also called *germ* cells. Meiosis consists of two rounds of cell division. In the first cell division, the two copies of the chromosome (henceforth called *paternal* and *maternal* chromosomes) are first duplicated, producing two copies of the paternal chromosome and two copies of the maternal chromosome, as shown in Figure 2.2. The four copies of each chromosome then arrange them selves in such a way that each paternal copy is lined up against a maternal copy. Then some thing called *chromosomal crossing over* takes place, and parts of the maternal copy will be swapped with parts of the paternal copy.

The chromosomal crossover is generally called *recombination*. Recombinations are quite common. According to [AJL03], 2-3 recombination events take place in each chromosome during meiosis. After the chromosomal crossover, the first cell division takes place, and two diploid cells are created. Each such diploid cell then undergoes a second round of cell division, this time without any DNA replication, thus producing four haploid cells. Each haploid cell has only one copy of each chromosome. Due to the crossover, none of the chromosomes in the haploid cells are exact copies of the paternal or the maternal chromosomes in the original diploid cell.

During fertilization, the DNA from a haploid cell (sperm cell) of the father is delivered into the a haploid cell (the egg) of the mother. The fertilized egg has two copies of each chromosome again, and is a complete diploid cell. In the child, exactly one of the two copies of each chromosome is derived from the father and the other is derived from the mother. A schematic representation of this inheritance is shown in Figure 2.3.

2.2 Phylogenetics

Phylogenetics is the study of evolutionary relationships between organisms. A *phylogeny* is typically a tree whose leaves are a set of *taxa*. The taxa for a given phylogeny is a set of



(a) Original diploid cell (b) DNA replication (c) Alignment

(d) Cross over (e) Before cell division 1 (e) The two diploid cells after cell division 1

(f) The four haploid cells after cell division 2

Figure 2.2: Different stages of meiosis

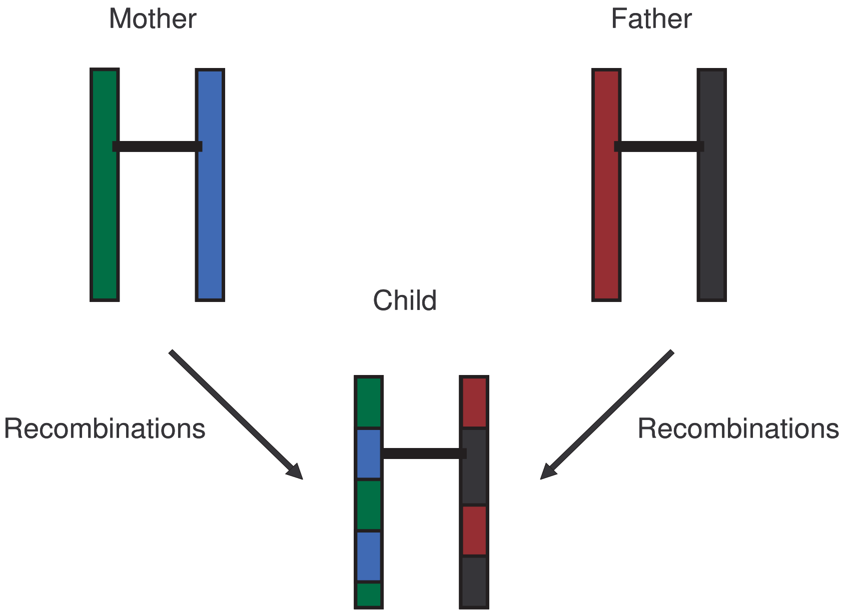


Figure 2.3: Inheritance of the chromosomes

species or set of individuals of the same species. Figure 2.4 shows a phylogeny for a set of taxa. The taxa in this particular case are a set of species.

A rooted phylogeny gives the direction of evolution. The root represents the common ancestor of all the taxa in the phylogeny. Taxa that have a more recent common ancestor are more closely related. Phylogenetic reconstruction methods generally construct un-rooted trees. This is because of the difficulty in determining the root without external information. Un-rooted trees are generally rooted with the help of an *outgroup*, a taxon that is known to have diverged from the rest of taxa before they diverged from each other.

In a rooted tree, a *monophyletic group* is a group of taxa so that all the descendants of the most recent common ancestor of the group are in the group. A monophyletic group is

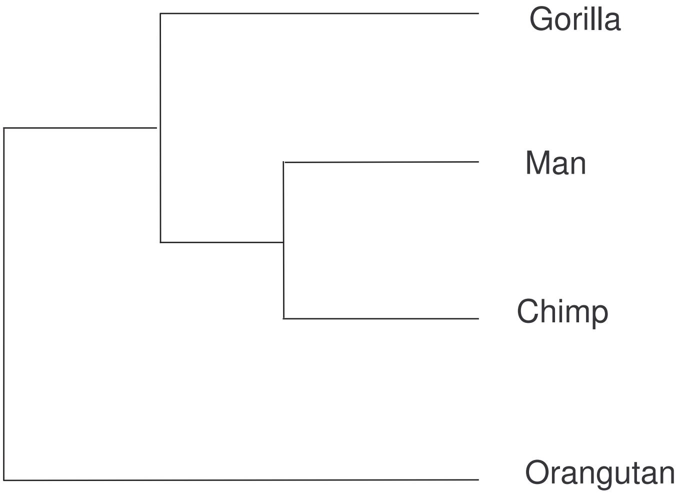


Figure 2.4: An un-rooted phylogenetic tree.

some times referred to as a *clade*. A *paraphyletic group* is a group of taxa that does not include all the descendants of the most recent common ancestor of the group.

2.2.1 Phylogenies on characters

Phylogenies are constructed on a set of *character*s. A character is an encoding of any kind of polymorphism - it can be a SNP site, a microsatellite, a restriction fragment length polymorphism (RFLP) site, or a morphological feature, among other things. These characters can be broadly classified into molecular characters and morphological characters. Molecular characters represent polymorphisms in DNA, RNA, or amino acid sequences, where as morphological characters represent external, observable features like the presence of fins, the color of eyes, etc. Molecular characters are generally discrete in that they have finite set of states in practice. Morphological characters, on the other hand can be continuous - for example, the length of the tail of a peacock.

This dissertation deals with phylogenies constructed on molecular characters. Each variation of the character is called an *allele*. If only two variations (or states) are possible for a given character, the character is said to be *biallelic*. If more than two states are possible for the character, the character is said to *multiallelic*.

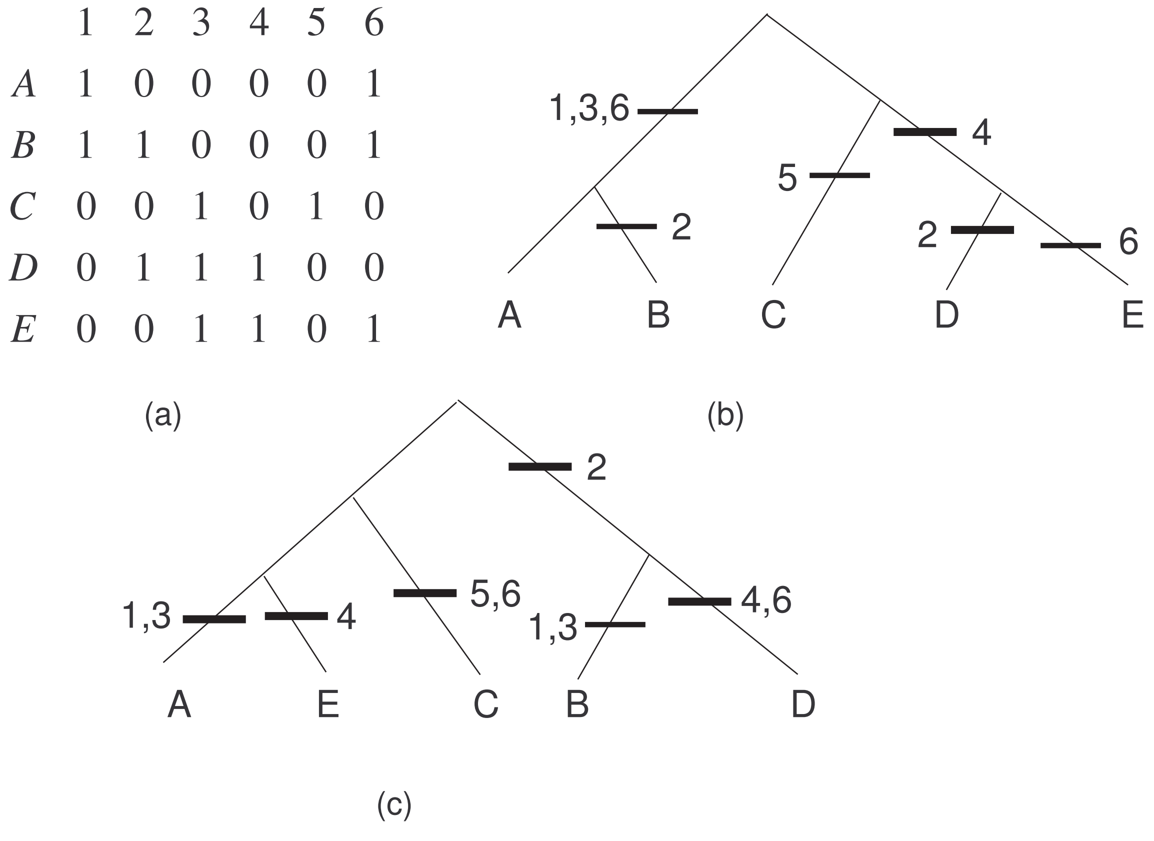
The data for constructing a phylogeny is generally provided as a two-dimensional matrix. The rows of the matrix represent the taxa, and the columns represent the characters. Figure 2.5-(a) represents five species over six bi-allelic characters. For simplicity, the two alleles in each character are encoded using the symbols '0' and '1'.

2.2.2 Parsimony

Each taxon can be represented as a character vector, where each position in the vector represents a character, and consists of the state of that character in the taxon. Given the character vectors of each taxon, the edges in any phylogeny on a set of taxa can be labeled to indicate where the state changes occur in the phylogeny. Obviously, any given phylogeny can be labeled in multiple ways for the same set of taxa. However, we are generally interested only in those labelings that incur the least number of state changes. Such a labeling is called as the most *parsimonious* labeling of the given phylogeny. Algorithms to determine the most parsimonious labeling of a give phylogeny have been developed by Fitch [Fit71] and Sankoff [San75, SR75]. These algorithms are included in the book by Dr. Felsenstein [Fel04].

A phylogeny is the most parsimonious phylogeny if it involves the minimum number of states changes among all the possible phylogenies for the given taxa. For example, the phylogeny in Figure 2.5-(b) is the most parsimonious phylogeny for the taxa described by the matrix in Figure 2.5-(a). The phylogeny in Figure 2.5-(b) requires eight state changes. The state changes are indicated using the horizontal bars on the edges. These bars are, in effect, the edge-labels of the edges. Because of the edge labels, we can determine the state of each character at each internal node of the phylogeny. i.e., each internal node can be labeled by a character vector.

Parsimony is generally accepted as the criterion for obtaining the best tree for the given taxa. It is generally agreed upon that evolution takes the 'shortest path', and hence parsimony is a biologically relevant criterion. However, building the most parsimonious tree for the given set of taxa is an NP-hard problem [DJS86]. Several heuristics have been developed for obtaining the parsimonious tree. Implementations of these heuristics are available through the popular phylogeny reconstruction packages PHYLIP [phy06] and PAUP [pau06]. Even with these heuristics, parsimony criterion can only be used with data sets with small number of taxa. The number of possible trees increases quickly (faster than exponential) with the increase in the number of taxa. This makes parsimony based approaches impractical for data sets involving a large number of taxa.



	1	2	3	4	5	6
A	1	0	0	0	0	1
B	1	1	0	0	0	1
C	0	0	1	0	1	0
D	0	1	1	1	0	0
E	0	0	1	1	0	1

Figure 2.5: A character-based phylogeny. (a) A character matrix with five taxa and six characters. (b)The most parsimonious phylogeny for the taxa in (a). (c) A non-parsimonious phylogeny for the taxa in (a)

2.2.3 Perfect Phylogeny

A phylogeny is called a *perfect phylogeny* if every character state is generated only once in the phylogeny. In other words, the phylogeny is a perfect phylogeny if for each character state, the taxa that have that character state form a sub-tree of the phylogeny. Perfect phylogeny is relevant since mutation events are generally unique within a population. If a set of taxa admit a perfect phylogeny for a given set of characters, the perfect phylogeny will be the most parsimonious phylogeny for the given taxa. This, combined with the fact that perfect phylogenies can be constructed in linear time makes them very useful in practice.

The problem of determining if a given set of taxa admit a perfect phylogeny is NP-complete [Ste92, BFW92]. However, the problem can be solved in polynomial time when each character has a constant number of alleles. Many polynomial time algorithms have been developed for this fixed character state problem [AF94, KW97]. The complexity of the best algorithm so far is $O(2^{2r}m^2n)$, where n is the number of taxa, m is the number of characters, and r is the maximum number of states of a character, which is assumed to be a constant.

However, the problem is much simpler when $r = 2$, i.e., when each character has only two alleles. In this case, the problem has a linear time ($O(nm)$) solution, as presented in [Gus91]. As most SNPs in the human genome are bi-allelic, this version of the perfect phylogeny problem will be frequently encountered in the rest of this dissertation.

2.3 Haplotype Inference

As explained in Chapter 1, diploid organisms have two copies of each chromosome. To obtain the haplotype data, the two copies need to be isolated, and analyzed separately. As described by Niu [Niu04], many methods for obtaining haplotype data empirically do exist [SRR90, MTB96, DBG01, WGC00, OHE02, MKE02, ZLH01]. However, all these methods are currently prohibitively expensive.

Hence, the practical approach is to analyze both the copies of a chromosome from an individual simultaneously. Through this approach, the two alleles at each SNP locus are obtained. The data thus obtained is referred to as un-phased genotype data, or simply the genotype data. Computational methods are necessary to obtain the haplotype data from the genotype data.

Given a set of genotypes, the haplotype inference problem is to compute a pair of haplotypes for each input genotype. This dissertation will assume that all the loci in the input genotypes are bi-allelic. This assumption is justified because of the fact that more than 99% of the SNPs (with Minimum Allele Frequency (MAF) $> 2\%$) in the human genome are bi-allelic [Int05].

2.3.1 Formal Statement of the Haplotype Inference Problem

Since we are dealing with bi-allelic data, the two alleles in each SNP locus can be represented using the symbols '0' and '1'. Using this notation, a haplotype over m loci is a vector of length m over the alphabet {0,1}. i.e, a haplotype $h = \{0, 1\}^m$. Homozygous loci in a genotype can be represented by the corresponding allele. Heterozygous loci in a genotype are represented using the symbol '2'. i.e., a genotype can be represented by a vector $g = \{0, 1, 2\}^m$.

The input to the haplotype inference problem consists of n genotype vectors, each of length m. A pair of haplotypes $< h, k >$, each of length m, are called an *explanation* of the a genotype g if $h[i] = k[i] = g[i]$ $\forall$ i such that $g[i] \neq 2$, and $h[i] \neq k[i]$ $\forall$ i such that $g[i] = 2$. In other words, h and k are a pair of haplotypes that *explain* or *resolve* the genotype g. The vectors h and k are said to be *compatible* with g. Given any two out of g, h and k, it is easy to deduce the third. If the genotype g has p heterozygous sites ($p > 0$), there will be 2^{p-1} distinct pairs of haplotypes that can explain g (if $p = 0$, $h = k = g$). However, the problem is to determine which one of these 2^{p-1} haplotype pairs is the 'true' explanation for g. Without additional information, each one of the 2^{p-1} haplotype pairs is equally likely to be the 'true' explanation of g. Therefore, we need additional information in order find the 'true' explanation, or even to limit the number of possible explanations.

There could be many ways of defining the 'correct', or most likely explanation for the given set of genotypes. As in many other problems, parsimony is generally used as the criterion to define the correct solution. One possible way of defining parsimony in this case is in terms of the number of distinct haplotypes. With respect to this definition of parsimony, the correct explanation will be the one requires the fewest number of distinct haplotypes.

2.3.2 The Maximum Resolution Haplotype Inference Problem

Clark [Cla90] introduced the haplotype inference problem. He proposed a practical, parsimony based version of the Haplotype inference problem, and provided a heuristic for the problem. The following paragraph introduces some terminology necessary to explain Clark's algorithm.

Any genotype vector that has less than two heterozygous sites has a unique explanation, and therefore is called *unambiguous*. A genotype vector that has two or more heterozygous sites is called *ambiguous*. Given a set G of genotype vectors that contain both ambiguous and unambiguous genotype vectors, the unambiguous genotype vectors can be resolved directly, as they have a unique solution. Clark's approach first resolves these unambiguous genotype vectors, and adds the resulting haplotypes to a set of haplotype vectors H, which is initially empty. Once a genotype vector is resolved, it is removed from G. Clark defined the following inference rule:

Inference Rule:

> Select a genotype $g \in G$ and a haplotype $h \in H$ such that h is compatible with G. Deduce the haplotype k such that the pair h, k resolves g. Now set $G \leftarrow G - g$, and $H \leftarrow H \bigcup k$.

Clark's heuristic algorithm works by applying the inference rule repeatedly until the set G is empty or until none of the haplotypes in H are compatible with any of the genotypes in G. At any step in the procedure, there might be multiple options for selecting h and g, and each option might lead the procedure in a different direction. Therefore, in each run of the algorithm, depending on the series of steps taken, we might end up with a different solution, or might get stuck after resolving a different set of genotypes. The goal is to find a series of applications of the inference rule that resolves the maximum number of genotypes. This problem is known as the maximum resolution (MR) problem. Formally stated:

MR Problem:

> Given a set of genotypes G and a set of haplotypes H what is the size of the minimum cardinality set G achievable by repeatedly applying the inference rule?

The problem was shown to be NP-hard [Gus01]. Clark's original approach was to perform thousands of runs, randomly selecting h and g whenever the inference rule has to be applied. The best solution obtained during these runs is adopted as the solution for the instance. The complexity of the MR problem is due to the fact that it does not assume any biological model. Assuming a biological model might simplify the algorithm, and result in an efficient solution.

2.3.3 Block structure of the human genome and the perfect phylogeny haplotyping problem

Linkage Disequilibrium (LD) is defined as the non-random association between two or more loci on a chromosome. Two or more loci are said to be in *linkage equilibrium* when the observed frequencies of haplotypes covering the loci agree with the haplotype frequencies predicted by multiplying the individual frequencies of the allele at each locus. The loci are said to be in linkage disequilibrium when they deviate from linkage equilibrium. Different measures like D' and r^2 have been developed to measure LD. A detailed description of different measures for LD was presented by Hudson [Hud01].

Many studies [DRS01, PBH03, RCB01] have shown that the human genome can be divided into regions with high LD. The regions with high LD are called *blocks*. Few recombinations are expected to have occurred within regions of low LD. This, combined with the fact that repeated mutations are rare in the human genome, leads to the possibility that the phylogeny of the haplotypes within each block is close to a perfect phylogeny.

As described earlier, the perfect phylogeny model assumes that the evolutionary history of all the haplotypes in a given population can be described by a rooted tree, also known as a *coalescent*. Since the haplotypes in a population evolve on phylogeny, applying a phylogenetic model to the haplotype inference problem is biologically relevant.

The *Perfect Phylogeny Haplotyping* (PPH) problem is to determine if there is a set of haplotypes H that is an explanation of a given set of genotypes G so that H admits a perfect phylogeny.

Even though the PPH model imposes severe restrictions on the phylogenetic network, it is practically applicable because of the block structure of the human genome. The perfect phylogeny formulation of the problem was first presented by Gusfield [Gus02].

2.3.4 Formal statement of the perfect phylogeny haplotyping problem

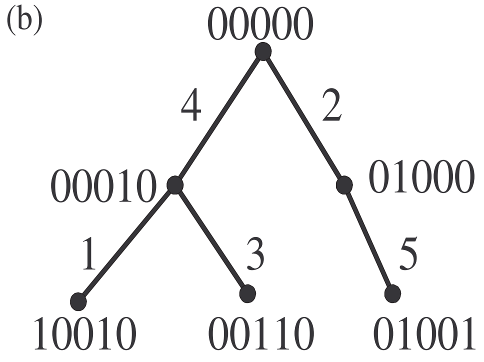
We are given a $n \times m$ matrix A over the alphabet $\{0, 1, 2\}$, in which the rows represent the genotype vectors, and the columns represent the SNP sites. The problem is to find a $2n \times m$ binary matrix B which has the following properties:

1. Every row in the matrix A is explained by a pair of rows in the matrix B.
2. There is a rooted perfect phylogeny T for the matrix B:
    (a) The root of the tree is labeled by an all-zero vector.

(a)

$$A = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{bmatrix} 2 & 0 & 2 & 1 & 0 \\ 2 & 2 & 0 & 2 & 0 \\ 0 & 2 & 2 & 2 & 2 \end{bmatrix} \end{array}$$

(b)



(c)

$$B = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Figure 2.6: (a) A genotype matrix A (b) A PPH tree T for the matrix A (c) The haplotype matrix B that explains A

(b) Each node label in T is a compact representation of the sites that label the edges in the unique path from the node to the root. i.e, the sites that label the edges in the path from the node to the root are '1' in the node label, and all the other sites are '0'.

(c) Every edge in T is labeled by a site.

Each row in the matrix B represents a haplotype. Since all the rows in B label the nodes in T, the evolutionary history of the haplotypes is a perfect phylogeny. A genotype matrix A that covers five SNP sites is shown in Figure 2.6(a). A PPH tree T for the matrix A is shown in Figure 2.6(b). The corresponding haplotype matrix B is shown in Figure 2.6(c).

2.3.5 Utilizing Pedigree Data for Haplotype Inference

Pedigree data is the information about the relationships between individuals in the population. In some cases, limited pedigree data might be available. The pedigree data provides valuable information that can be used in order to infer haplotypes that are more accurate.

Representation of the pedigree data

Pedigree data is generally represented by a pedigree graph. The pedigree graph is a directed acyclic graph $G =< V, E >$, where $V = M \bigcup F \bigcup N$. M represents the male nodes, F represents the female nodes, and N represents the mating nodes. The edges in E connect a mating node to a male or female node, or connect a male or female to a mating node. The in-degree of a each individual is at most 1, and the in-degree of a mating node is 2. A mating node defines a parent-child relationship between the individual nodes that are adjacent to it. The individuals that have edges from the mating nodes are called as the *children* of the individuals that have edges to the mating node. The individuals that have no parents are called the founders.

Practically speaking, in general, the pedigree data consists only of mother-father-child *trio*s. i.e., each connected component in the pedigree graph consists of two parents, a mating node and a child. The pedigree data provides additional information in the following ways:

- **Inferring the alleles at some loci:** Two rules can be applied at some locations in order to obtain the two alleles at that loci - (1) If at least one of the parents is homozygous, we can determine which allele comes from which parent (also called as the parental source of each allele) in the child. (2) If the child is homozygous and one of the parents is heterozygous, we can determine which allele in the parent was inherited by the child.

- **Inferring missing data:** Using the rules mentioned above, we can infer the missing data at some alleles.

- **Detecting errors in the data:** If none of the alleles in the child match any of the alleles in the parent, we can infer that there is some error in the data in the child or in the parent, or both. For example, if the child is homozygous with the 0-allele and a parent is homozygous with the 1-allele, at a particular loci, it indicates that there is definitely some error with the data at that loci.

- **Detecting recombination sites:** It might be possible to determine that a recombination has taken place between two loci during the process of inheritance from the parent. However, the recombinations that occur in evolutionary history of the parent can not be determined.

The MRHC problem

The Minimum Recombinant Haplotype Configuration problem can be stated as follows:

> Given a set of genotypes and the pedigree data, find the haplotype assignment that results in the minimum number of recombination events within the pedigree.

Unfortunately, the problem was shown to be NP-hard in [LJ03]. However, a heuristic, called the block extension algorithm, was presented in [LJ03]. The authors claim that the algorithm performs well in practice. Though their approach seems to perform better than some of the earlier approaches for the same problem [QB02], the effectiveness of the algorithm on real life genotype data has not been established.

2.3.6 Limitations of the pedigree data

The main limitation of algorithms based on pedigree data is the availability of pedigree data. As far as humans are concerned, very little or no pedigree data is available in general. For example, in the four sample populations that are being analyzed by the HapMap project [Hap03], the pedigree data is available only for two of the populations. Even in those two populations, the pedigree data comprises of a set of un-related trios. However, the algorithms in [LJ03, QB02] assume that all the individuals in the population are related by a single, connected pedigree graph. When the pedigree consists of unrelated trios, there is very little useful information that can be obtained from the algorithms in [LJ03, QB02]. A comparison of different pedigree-based programs was presented in [LZH04]. The comparison was done using simulated data for nuclear families with 1-5 children. The study concluded that mis-assignments were unacceptably high for small nuclear families. It was concluded that the programs needed at least 4-5 children in a family to predict haplotypes with acceptable levels of accuracy.

However, the pedigree data provides a lot of information which can be quite helpful in arriving at the 'true' haplotype configuration. The best way to use pedigree data is to use the pedigree to validate the results obtained without using the pedigree data. The pedigree data can be used to infer missing data, and detect errors. Apart from that, the pedigree data can be used to restrict the number of solutions, if there are multiple valid haplotype resolutions for some genotype data.

2.4 Haplotype Inference on Real Genotype Data

Applying any of the PPH algorithms to actual genotype data is another challenge. The genotype data (like any other biological data) is often populated with missing data. In addition, real genotype data may be very deviate from a perfect phylogeny quite often. Therefore more robust approaches are necessary that can handle missing data and/or deviations from perfect phylogeny. Pedigree data, when available, might assist in the imputation of some missing entries.

2.4.1 Homoplasy Events

Violations of the infinite sites assumption, i.e, repeated or back mutations, are called as homoplasy events. One reason why true genotype data deviates from perfect phylogeny is because of homoplasy events. Though homoplasy events are generally rare in the human genome, mutation rates vary from site to site, and is possible to have loci that have mutated multiple times in the evolutionary history of a given population.

2.4.2 Recombinations

Recombination events are quite common. According to [AJL03], between two and three recombination events occur in each chromosome during meiosis. If we consider the evolutionary history of the mankind, millions of recombination events would have accumulated in each chromosome. Therefore, any algorithm that can handle data with recombinations, even a limited number of them, will be very helpful. It was shown in [LZ01] that the problem of finding a phylogenetic network for data with recombinations is NP-hard in general. However, [LZ01, GEL03] attempted to introduce some recombinations into the perfect phylogeny model, calling it a *phylogenetic network*. An algorithm was presented in [GEL03] that can deal with phylogenetic networks in which the recombination cycles are node-disjoint. Such a phylogenetic network is called a *galled tree*. If there is a galled tree for the genotype data, the galled tree can be obtained in $O(nm + n^3)$ time using the algorithm presented in [GEL03]. It was also established that any set of sequences that can be derived on a galled tree can also be derived on a PPH tree that allows one back mutation per site. A lower bound on the minimum number of recombinations required in a phylogenetic network was obtained in [BB04, GH04].

2.4.3 Block partitioning on real genotype data

Block partitioning of the human genome is the ultimate goal of the HapMap project. The ability to divide the chromosomes into blocks with limited diversity is absolutely essential for high-throughput genotyping. The motivation is to come up with a few ‘tag’ SNPs for each block. The tag SNPs have to be selected in such a fashion that a high percentage (80-100) of the haplotypes can be distinguished by just knowing the states of the tag SNPs. Therefore, it will be sufficient to collect data about just the tag SNPs. For example, Patil et. al. [PBH03] could divide a region covering 24,047 SNPs from chromosome 21 into 4,135 blocks containing just 4,563 tag SNPs. Most of the current techniques for block partitioning rely on linkage disequilibrium(LD) measures (see glossary).

A major drawback of the current block partitioning techniques is that they assume that the haplotypes are directly available. The fact that it is the genotype data that is experimentally obtained, and that the haplotype data has to be computationally derived from the genotype data is conveniently ignored in most approaches. The inaccuracies that creep in during haplotype inference result in errors in the haplotype data. The block-partitioning techniques are themselves heuristics or statistical approaches, and introduce new inaccuracies. On the whole, the quality of the block partitioning achieved by this two-step procedure might not be very reliable.

The coalescent model is very promising in that it allows block partitioning directly from the genotype data. The resulting blocks will be such that all the SNPs within in each block label the edges of a PPH tree. Determining the tag SNPs becomes a trivial problem, as the edges incident on the extant leaves of the tree become the tag SNPs for the block.

2.5 Constructing Perfect Phylogenies on Haplotypes

The input to this problem is a set of n haplotypes. Each haplotype is a binary vector of length m, where 0 and 1 represent the two alleles. These n haplotypes can be represented by an $n \times m$ matrix M over the integer alphabet {0,1}. Each row in M represents a haplotype, and each column represents a character. Through out this dissertation, the terms ‘character’, ‘column’ and ‘site’ are used interchangeably. The problem is to construct a perfect phylogeny T for the matrix M, or to determine that the matrix M does not admit a perfect phylogeny. The following definitions will be necessary.

A column i is said to be *polymorphic* in M if there is at least one row r_0 in M with $M[r_0, i] = 0$ and at least one row r_1 so that $M[r_1, i] = 1$. An underlying assumption about M is that all columns in M are polymorphic. This is because no mutation events will be necessary in a non-polymorphic site in any phylogeny for the matrix M. Hence the matrix M can be pre-processed to remove all non-polymorphic columns from M. Two columns i and j are said to *equivalent* in M if one of the following two conditions are satisfied:

1. $M[r, i] = M[r, j]$ for every row r in M.

2. $M[r, i] = 1 - M[r, j]$ for every row r in M.

If two columns i and j are equivalent, they both label the same edge in any phylogeny for M. It is sufficient to consider only one column out of the two columns i and j for the purpose of constructing a phylogeny. Therefore, M can be pre-processed to ensure that there are no two columns that are equivalent to each other.

An ordered pair (a, b), $a \in \{0, 1\}$, $b \in \{0, 1\}$, is said to be *induced* by a pair of ordered columns (i, j) if there is a row r in M such that $M[r, i] = a$ and $M[r, j] = b$. The set of ordered pairs induced by a pair of columns (i, j) is denoted by $I(i, j)$.

2.5.1 Necessary and sufficient conditions for M to admit a perfect phylogeny

The following theorem has been stated many times before, using different terminology:

Theorem 2.1 *The matrix M admits a perfect phylogeny* iff $|I(i, j)| \leq 3$ *for every pair of columns* (i, j).

Proof Let us first consider the *only if* part of the theorem. This part of the theorem implies that the matrix M does not admit a perfect phylogeny if $|I(i, j)| = 4$ for any pair of columns (i, j). To see why this is true, consider the matrix restricted to just the two columns i and j. We denote this matrix by $M[*, ij]$. There is a unique topology for a perfect phylogeny with two sites, as shown in Figure 2.7. Since the phylogeny has just two edges, it will have three vertices U, V and W, as shown. Let the state of site i at vertex U be a,

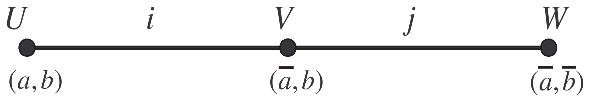


Figure 2.7: The only possible topology for perfect phylogeny with two sites

where $a \in \{0, 1\}$. Similarly, let the state of site j at vertex U be b, where $b \in \{0, 1\}$. Since the site i labels the edge (U, V). The states of the sites i and j at vertex V must be $\overline{a}$ and b, respectively. Similarly, the states of sites i and j at vertex W will be $\overline{a}$ and $\overline{b}$, respectively.

If $M[*, ij]$ admits a perfect phylogeny, each row in $M[*, ij]$ must have been derived from the vertices U, V or W. Therefore, the set of ordered pairs induced by the pair of columns (i, j) will be $I(i, j) = \{(a, b), (\overline{a}, b), (\overline{a}, \overline{b})\}$. i.e., irrespective what the actual values of a and b might be, $|I(i, j)|$ can never be greater than 3. Therefore, the matrix $M[*, ij]$ admits a perfect phylogeny *only if* $|I(i, j)| \leq 3$.

Now, consider the *if* part of Theorem 2.1. We need to show the matrix M admits a perfect phylogeny if $|I(i, j)| \leq 3$ for every pair of columns (i, j). Consider any column x. The following discussion will demonstrate that there will be a phylogeny for M in which there is only one edge labeled with the site x. Divide the rows of the matrix M into two non-overlapping sets S_0 and S_1 using the following criterion - a row $r \in S_0$ if $M[r, x] = 0$, and $r \in S_1$ if $M[r, x] = 1$. In any phylogeny for T for M, let (U, V) be the edge labeled with the site x, with the state of x being 0 at vertex U and 1 at vertex V, as shown in Figure 2.8. All the haplotypes in S_0 form a subtree T_0 rooted at U and the haplotypes in S_1 form a subtree T_1 rooted at S_1, as shown.

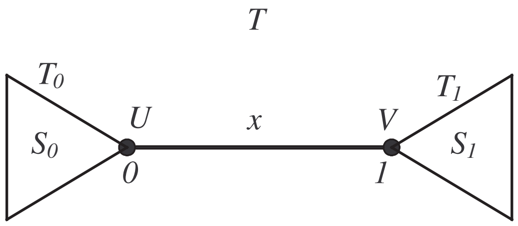


Figure 2.8: Illustration of the proof for Theorem 2.1

By definition, the site x is non-polymorphic in both the sets S_0 and S_1. Hence, there will be no edge labeled with x in either T_0 or T_1. Let C_0 be the set of columns that are polymorphic in S_0 and C_1 be the set of columns that are polymorphic in S_1. We know that, for every column $i \neq x$, $I(i, x) \leq 3$. Therefore, no column $i \neq x$ can be polymorphic in both S_0 and S_1. i.e., any column $i \neq x$ in M will either be in C_0 or in C_1, but not in both. Since $|I(i, j)| \leq 3$ for all pairs of columns in M, $|I(i, j)| \leq 3$ for all pairs of columns in C_0 and C_1. Therefore, similar to x in T, it is possible to construct T_0 and T_1 so that any site $c_0 \in C_0$ will label a single edge in T_0, and any site $c_1 \in C_1$ will label a single edge in T_1. Therefore, M admits a perfect phylogeny if $|I(i, j)| \leq 3$.

This completes the proof for Theorem 2.1. $\diamondsuit$

2.5.2 Rooted Perfect Phylogenies

It is often easier to construct a rooted perfect phylogeny than to construct a un-rooted perfect phylogeny. A rooted phylogeny is sometimes also referred to as a *directed* phylogeny.

Correspondingly, an un-rooted phylogeny is referred to as an *undirected* phylogeny. An un-rooted phylogeny can be converted into a rooted phylogeny by designating any vertex in the phylogeny as the root. By convention, the root is generally assumed to be an all-zero vector. For any given matrix M, ensuring that every column is majority-zero guarantees that there will be a vertex labeled with an all-zero vector in any phylogeny T for M. We can then root the phylogeny at the all-zero vector. In other words, ensuring that each column in M is majority-zero guarantees that the root is an all-zero vector. If any column in M has more '1's than '0's, it can be converted to a majority-zero column by simply complementing each entry in the column.

The *only if* part of Theorem 2.1 is also known as the four-gamete test [HK85]. Traditionally, the perfect phylogeny problem is dealt with as a rooted, or directed perfect phylogeny problem with the root being an all-zero vector. In this context, with the all-zero vector as the root, Theorem 2.1 is stated as in the following statement - *The haplotype matrix M does not admit a perfect phylogeny if any sub-matrix formed by two columns in M contains the three rows* $\{01, 10, 11\}$.

2.5.3 Algorithms for the perfect phylogeny problem on binary characters

Many $O(nm)$ algorithms have been presented for the perfect phylogeny problem on binary characters. One of the simplest algorithms is presented in [Gus97]. The following is a high level description of the algorithm:

1. Treat the columns in M as binary strings, and sort the columns according to their numerical value.
2. For each row in the sorted matrix, construct the string of characters, in sorted order.
3. Build the keyword tree T for the n character strings formed in step 2.
4. Test if T is a perfect phylogeny.

All the steps except for the sorting step can be trivially completed in $O(nm)$ time. The sorting step can be implemented to run in $O(nm)$ time using radix sort.

One other $O(nm)$ time algorithm has been presented in [SM97].

CHAPTER 3

PERFECT PHYLOGENY HAPLOTYPING: THE FLEXTREE DATA STRUCTURE AND THE OPPH ALGORITHM

3.1 The Perfect Phylogeny Haplotyping Problem

In the case of the PPH problem, we are given an $n \times m$ genotype matrix A. The problem is to determine if there is an $2n \times m$ haplotype matrix B ssuch that:

1. Each row (i.e., genotype vector $A[r]$) can be produced by conflating the two haplotypes vectors $B[2r-1,*]$ and $B[2r,*]$ in B.

2. The haplotype matrix M admits a perfect phylogeny T.

Gusfield [Gus02] introduced the rooted version of the PPH problem (with the root being an all-zero vector) and made some important observations about the problem. Let h_r and k_r be the two haplotypes for the row r in the matrix A. The following three observations,

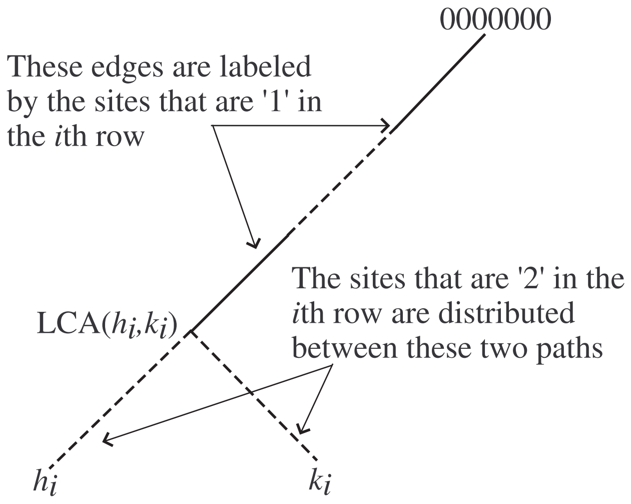

0000000
These edges are labeled by the sites that are '1' in the *i*th row
The sites that are '2' in the *i*th row are distributed between these two paths
LCA(h_i,k_i)
h_i
k_i

Figure 3.1: Properties of the two haplotypes of a genotype

first made in [Gus02], are used, directly or indirectly, in every solution to the problem (see Figure 3.1:

Observation 3.1 *The set of columns that are '1' in a row r of A specify the exact set of edge labels from the root to the lowest common ancestor of nodes labeled with h_r and k_r, in every perfect phylogeny for A.*

Observation 3.2 *Any column c that is '2' in the row r of A must be in the path from the root to* exactly *one of the nodes labeled with h_r and k_r in any perfect phylogeny for A.*

Observation 3.3 *Any column c that is '0' in the row r of A must not be in the path to either of the nodes labeled with h_r or k_r, in any perfect phylogeny for A.*

The concept of *column sums* was also noted in [Gus02]. The column sum η_i of a column i in A is the number of '1's in column i in any binary matrix B that is a explanation of A.

η_i is given by the following expression, where $A[*, i]$ denotes the ith column in matrix A:

$$\eta_i = (\text{\# of 1's in } A[*, i] \times 2) + (\text{\# of 2's in } A[*, i]) \tag{3.1}$$

The column sum gives the exact number of haplotypes that must be in the subtree under the edge labeled with i in any perfect phylogeny for the matrix A. The column sums impose an order on the columns in any perfect phylogeny for A - no column with a smaller column sum than η_i can label an edge in the path from the root to the edge labeled with the column i. Though the significance of the column sums was noted in [Gus02], the algorithm itself does not make complete use of the ordering imposed by the column sums. Other solutions for the PPH problem [BGL02, EHK03, Wiu04] have mainly ignored this property and failed to take advantage of it. The ordering imposed by the column sums plays a crucial role in the optimal $O(nm)$ opph algorithm that is presented in this chapter.

3.1.1 Solution via graph realization

In [Gus02], the PPH problem is solved by mapping the problem to a graph realization problem. The algorithm first builds an initial perfect phylogeny T_{11} for the columns that have at least one '1'. The algorithm then defines path sets for each row in A. Each path set for a row i consists of all the columns that are '1' in the row and a set of columns that are '2' in the row. The algorithm uses deep results in matroid theory and graph realization in order to arrive at a tree that realizes all these path sets. A complete algorithm is not presented in [Gus02], but the complexity of the approach was stated as $O(nm^2)$, based on the complexity of the underlying graph realization problem. It was mentioned in [Gus02] that the implementation of the algorithm is complicated. It was conjectured that a direct approach might provide a simpler solution to the problem, that is easier to implement and understand.

3.1.2 A direct approach for the PPH problem

Consequently, a direct approach for the PPH problem was presented in [BGL02]. The direct approach defines pair-wise relationships between the columns in the matrix A. The approach makes use of the standard four-gamete test, first presented in [HK85]. A pair of columns i and j are defined as *companion* columns if both of them are '2' in any row of A. All the rows in which both the columns i and j are '2' are called the companion rows for the columns i and j. Any companion row in the matrix A can be expanded in two ways in the matrix B, w.r.t the columns i and j: it can be expanded as the rows {00,11} or as the rows {10,01}. In the former case, the companion row is said to have been expanded *equally* w.r.to columns i and j. In the later case, the companion row is said to have been expanded *unequally* w.r.t. columns i and j. The approach taken in [BGL02] is based on the fact that unless all the companion rows of a pair of columns are expanded in the same way, the resulting matrix B will not be realizable by a perfect phylogeny. This is obvious, as the matrix B will fail the four gamete test for the columns i and j if one of the companion rows is expanded equally and the other expanded unequally. Therefore, some of the companion columns are forced to be expanded equally or unequally based on the state of the two columns in the non-companion rows in the matrix A. The solution in [BGL02] essentially constructs a graph G in which each site is represented by a vertex. Companion sites in the matrix A are connected by an edge in G. There are three types of edges: the companion sites are connected by an *equal* edge if they are forced to expand equally. They are connected by an *unequal* edge if they are forced to expand unequally. Finally, they are connected by a *neutral* edge if they are neither forced to expand equally, nor forced to expand unequally. Each neutral edge can be converted into an equal edge or an unequal edge. The matrix A is realizable by a perfect phylogeny if there is an assignment of equality or un-equality to each neutral edge such that there are no cycles in the graph that contain an odd number of unequal edges.

Clearly, the approach requires all the pairwise relationships between all pairs of companion columns. Since there are $O(m^2)$ pairs of companion columns and since collecting the equality/unequality relation ships between a pair of columns takes $O(n)$ time, the overall complexity of the algorithm is $O(nm^2)$. Out of all the algorithms presented for the PPH problem, this was the algorithm that came closest to an $O(nm)$ solution. However, the algorithm has completely ignored the relative ordering induced by the column sums, and hence could not achieve the $O(nm)$ bound.

3.1.3 Improvements to the direct approach

Wiuf [Wiu04] attempted to improve upon the approach taken in [BGL02], and made some interesting observations, among which is the observation that there will be no cycles in the graph G with odd number of unequal edges unless there is at least one row in the matrix that has three '2's. Consequently, the algorithm tries to establish transitive relationships between pairs of columns. However, the algorithm does not make use of the ordering induced by the column sums, and hence fails to achieve an $O(nm)$ bound.

3.1.4 Other solutions

The most significant among the other solutions for the PPH problem is presented in [EHK03]. One important contribution of [EHK03] is that it clearly establishes the fact that if the matrix A can be explained by an un-rooted tree T, then the matrix A can be explained by a rooted tree in which the root is an all-zero vector. It also presents a generalized concept of realizability for the binary matrix B. It states that a binary matrix B is realizable by a perfect phylogeny only if the number of distinct rows is less than four in the sub-matrix induced by each pair of columns. This statement might seem like a re-statement of the 4-gamete test, but actual significance of the statement is that it establishes a realizability criteria in which the root does not have to be an all-zero vector.

The algorithm works by defining a set of pair-wise relationships between the columns. Some of the relationships do impose an order on the columns. However, since the column sums are not utilized in building these pair-wise relationships, this ordering is not apparent until all the pair-wise relationships are built. However, building the pair-wise relationships takes $O(nm^2)$, and hence the overall complexity of the algorithm is $O(nm^2)$.

Eskin et.al. [EHK03] also provide some useful insights on how to tackle the problem of realizing an imperfect phylogeny. They present a criteria for quantifying the discrepancies in T induced by a pair of columns in B that fail the four-gamete test. This leads to a heuristic approach to realizing imperfect phylogeny which defines certain error thresholds. The approach makes it possible to determine if there is a tree T in which none of the pairs of columns in B exceed the error threshold.

As mentioned in [Gus02], the column sums induce an order on the columns. However, the algorithm in [Gus02] did not make complete use of this ordering - the ordering was only used in case of columns that have at least one '1'. The method failed to take advantage of the fact that the ordering applies even to the columns that do not have any '1's. Other algorithms [BGL02, EHK03, Wiu04] completely ignored this ordering, and mostly rely on building all pairwise relationships between the columns. The fact that the columns can be ordered leads to this idea - can the rows be ordered in some fashion, so that an algorithm can take advantage of the row ordering? Given the row ordering, can there be an algorithm that spends $O(m)$ time in each row, but collects all the information necessary to build a PPH tree?

In order to determine if a matrix is realizable and to represent all possible PPH trees for the matrix we need all pairwise relationships between the columns. However, since the columns are ordered, it might be possible to store only some of these pair wise relationships explicitly, and implicitly infer the rest. Hence, we will need a robust data structure that allows us to manage and maintain all these relationships. The FlexTree data structure presented that is introduced in this chapter is such a data structure that allows us to represent most of the pairwise relationships implicitly.

3.2 Some Lemmas and Properties

This section will introduce some lemmas and properties in order to simplify the presentation of the problem. Throughout this chapter, we assume that the root of the phylogeny is an all-zero vector. For any genotype matrix that admits a perfect phylogeny, if the number of '1's in every column is less than or equal to the number of '0's, the root for the phylogeny will be an all-zero vector. Though every column in the input matrix A might not always satisfy this condition, there is a simple transformation that guarantees that the root is an all-zero vector. The transformation is to invert all columns with column sums greater than m - the '1's in the column are changed to '0's, the '0's are changed to '1's, and the '2's are left unchanged.

3.2.1 Columns sums

The column sum η_j of a column j gives the exact number of haplotypes in B that are in the subtree under the edge labeled with j in any perfect phylogeny for A. Consequently, we can define certain properties with respect to the column sums.

Lemma 3.1 *If two columns i and j in A are such that $\eta_i > \eta_j$ then the site j cannot be in the path from the root to the site i in any perfect phylogeny T for A.*

Proof The proof is trivial. let T_i be the subtree under i and T_j be the subtree under j in T. If j is in the path from the root to i, T_j will include T_i. But, this is not possible since $\eta_i > \eta_j$. Hence the site j cannot be in the path from the root to the site i. $\diamondsuit$

Lemma 3.1, when combined with Observation 3.1 leads to the following property:

Property 3.1 *If there is a row r in A such that $A[r, i] = A[r, j] = 1$ for any two columns i and j such that $\eta_i > \eta_j$, then the site i must be in the path to site j in any perfect phylogeny T for the matrix A.*

3.2.2 Pre-processing the input matrix A

It is clear from Lemma 3.1 that the column sums of the matrix A impose an ordering on the sites in any perfect phylogeny T for A. Let A^c be a matrix derived by re-arranging the columns of A sorted left to right according to non-increasing columns sums η. In A^c, if we take any two columns i and j such that $i < j$, it implies that $\eta_i \geq \eta_j$. This means that if i and j appear in a path from the root to any node in T, then the site i must precede the site j. Only one column out of any set of identical columns is retained. Hence, two columns with $\eta_i = \eta_j$ can not lie in the path to each other. The column-sorted matrix A^c has the following property:

Property 3.2 *Each realization of the matrix A^c will be a realization of the matrix A.*

Proof The matrix A^c is just a re-arrangement of the columns in the matrix A. Therefore, every column i^c in A^c corresponds to a column i in A. Any realization T^c of the matrix A^c can be transformed into a realization T of the matrix A just by re-labeling every column i^c in A^c with the corresponding column i in A. $\diamondsuit$

To determine the realizability criteria for the matrix A^c, we will need to interpret the standard four-gamete test in the context of the column-sorted matrix A^c. Let B^c be a haplotype matrix for A^c. To begin with, since the root is always an all-zero vector, the pair (0,0) is induced by any pair of columns (i, j). i.e., (0,0) is always in $I(i, j)$. Consequently, we need not test for the presence of a 00 row in $M[*, ij]$, and the four gamete test reduces to testing just for the three rows {01,11,10}. Further, since the matrix B^c is column-sorted, the four-gamete test reduces to testing for just two rows:

2-gamete test: In any column-sorted binary matrix B^c, if any sub-matrix formed by a pair of ordered columns consists of both the rows 01 and 11, then the matrix B^c cannot be realized by a perfect phylogeny.

$$\begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ B^c = & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

(a)

P_{B^c}	1	2	3	4
1	–	0	1	0
2	–	–	1	0
3	–	–	–	1
4	–	–	–	–

(b)

Figure 3.2: (a) A matrix B^c with four columns (b) The phase matrix for B^c

Extending 2.1 to the column-sorted matrix B^c implies that the matrix B^c admits a perfect phylogeny *iff* the sub-matrix formed by any ordered pair of columns does not contain more than one row from the set {01,11}. Each column in B^c has at least one '1', and hence the sub-matrix formed by each pair of columns in B^c has at least a 01 row or a 11 row. A pair of columns (x,y), $x < y$ in B^c are said to be *in-phase* if $B^c[*, xy]$ has a 11 row. The columns x and y are said to be *out-of-phase* if $B^c[*, xy]$ has a 01 row. For any binary matrix B^c that has a perfect phylogeny, these phase relationships can be represented by a $m \times m$ phase matrix P_{B^c}, in which $P_{B^c}[x, y]$ gives the phase relationship between the columns x and y. $P_{B^c}[x, y] = 0$ if x and y are in-phase and $P_{B^c}[x, y] = 1$ if x and y are out of phase. If the matrix B^c is not realizable by a perfect phylogeny, $B^c[*, xy]$ can have both rows 01 and 11, in which case the $P_{B^c}[x, y] = \psi$. A haplotype matrix B^c and the corresponding phase matrix P_{B^c} are shown in Figure 3.2. As the columns x and y have to be ordered, $P_{B^c}[x, y]$ is defined only if $x < y$, and hence only the upper triangle of the matrix B^c is defined.

To use the 2-gamete test to determine the realizability of the column-sorted genotype matrix A^c, we need to be able to interpret the '2's in each column. Every row except a 22 row in a sub-matrix $A^c[*, ij]$ of A^c induces certain rows in the sub-matrix $B^c[*, ij]$ of any matrix B^c that is an explanation of A^c. A 00, 01, 10 or 11 row in $A^c[*, ij]$ induces itself in $B^c[*, ij]$, where as a 02, 20, 12 or 21 row in $A^c[*, ij]$ induces the rows {00,01}, {00,10}, {11,10}, or {01,11} in $B^c[*, ij]$, respectively. If the matrix A^c is to be realizable, both 01 and 11 rows should not be forced in sub-matrix $B^c[*, ij]$. A phase matrix P_{A^c} for A^c can be defined based on these forced rows. For the matrix A^c, $P_{A^c}[i, j] = 0$ if a 11 row is forced in $A^c[*, ij]$, $P_{A^c}[i, j] = 1$ if a 01 row is forced in $A^c[*, ij]$ and $P_{A^c}[i, j] = \psi$ if both 01 and 11 rows are forced in $A^c[*, ij]$. However, if a sub-matrix $A^c[*, ij]$ of A^c has only 00,22 and 20 rows, the columns i and j are *neither forced in-phase nor forced out-of-phase*, and $P_{A^c}[i, j]$ is then designated as ϕ. Extending the 2-gamete test to a column-sorted genotype matrix A^c, we can now state the 2-gamete test for a column sorted genotype matrix A^c as follows:

Extended 2-gamete test: The column sorted genotype matrix A^c is not realizable by a perfect phylogeny if there are two columns i and j, $i < j$, such that $P_{A^c}[i, j] = \psi$.

An interesting result from the extended 2-gamete test is that in some situations, we can deduce that the matrix A^c is not realizable just by looking at a single row in A^c. A 21 row in any sub-matrix of A^c induces both 01 and 11 rows in the corresponding sub-matrix in B^c, and hence:

Property 3.3 *The matrix A^c is not realizable if a '2' occurs to the left of a '1' in any row.*

Thus, a necessary condition for A^c to be realizable is that each row can be partitioned into two parts, the left part containing no '2's and the right part containing no '1's. Checking if a row satisfies Property 3.3 is a simple procedure that takes $O(m)$ time. In the rest of the discussion, we assume that each row in the matrix A^c satisfies Property 3.3.

Property 3.4 *If a column j in A^c has at least one '1', then $P_{A^c}[i,j] \neq \phi$ for every column $i < j$.*

Proof Let r be the row in A^c such that $A^c[r,j] = 1$. For any column $i < j$, there are three possibilities:

- **Case 1:** $A^c[r,i] = 0$. $A^c[r,ij] = 01$, and hence $P_{A^c}[i,j] = 1$.
- **Case 2:** $A^c[r,i] = 1$. $A^c[r,ij] = 11$, and hence $P_{A^c}[i,j] = 0$.
- **Case 3:** $A^c[r,i] = 2$. $A^c[r,ij] = 21$, and hence $P_{A^c}[i,j] = \psi$. $\diamondsuit$

3.2.3 Implied relationships

The in-phase and out-of-phase relationships described above are *direct* relationships. These relationships between any pair of columns i and j can be directly deduced from the submatrix $A^c[*,ij]$. However, a row in the matrix A^c might force some additional *implied* phase relationships on pairs of columns. The matrix A^c will be realizable by a perfect phylogeny only if the implied relationships forced by a row do not contradict the direct relationships or the implied relationships forced by other rows. The following discussion describes some relationships that are indirectly forced.

Theorem 3.1 *In any realizable matrix A^c, given three columns x, y and z, if $P_{A^c}[x, y]\varepsilon\{0, 1\}$, $P_{A^c}[x, z]\varepsilon\{0, 1\}$, and if $A^c[r, x] = A^c[r, y] = A^c[r, z] = 2$ in any row r, then $P_{A^c}[y, z] = P_{A^c}[x, y] \oplus P_{A^c}[x, z]$, where $\oplus$ is the exclusive-or operator.*

Proof Let r_1 and r_2 be the two rows in B^c corresponding to the row r. The following situations are possible:

- **Case 1:** $P_{A^c}[x, y] = P_{A^c}[x, z] = 0$. Since $P_{A^c}[x, z] = 0$, x and z have to be expanded as 00 and 11 rows in $B^c[*, xz]$. w.l.o.g., let $B^c[r_1, xz] = 00$ and $B^c[r_2, xz] = 11$. Since $P_{A^c}[x, y] = 0$, x and y have to be expanded as 00 and 11 rows in B^c, and hence $B^c[r_1, y] = 0$ and $B^c[r_2, y] = 1$. However, this results in $B^c[r_2, yz]$ being a 11 row. Hence, the relationship $P_{A^c}[y, z] = 0$ is indirectly forced by the row r. The situation is illustrated by the columns x_1, y_1 and z_1 in the Table 3.1.

- **Case 2:** $P_{A^c}[x, y] = 1$ **and** $P_{A^c}[x, z] = 1$. Since $P_{A^c}[x, z] = 1$, x and z have to be expanded as 01 and 10 rows in $B^c[*xz]$. w.l.o.g., let $B^c[r_1, xz] = 01$ and $B^c[r_2, xz] = 10$. Since $P_{A^c}[x, y] = 1$, x and y have to be expanded as 01 and 10 rows in $B^c[*, xy]$. Hence, $B^c[r_1, y] = 1$ and $B^c[r_2, y] = 0$. However, this results in $B^c[r_1, yz]$ being 11. Hence, the relationship $P_{A^c}[y, z] = 0$ is indirectly forced by the row r. The situation is illustrated by the columns x_2, y_2 and z_2 in the Table 3.1.

- **Case 3:** $P_{A^c}[x, y] = 1$ **and** $P_{A^c}[x, z] = 0$. Since $P_{A^c}[x, y] = 1$, x and y have to be expanded as 01 and 10 rows in $B^c[*, xy]$. w.l.o.g., let $B^c[r_1, xy] = 01$, and $B^c[r_2, xy] = 10$. Since $P_{A^c}[x, z] = 0$, x and z have to be expanded as 00 and 11 rows in $B^c[*, xz]$.

Hence, $B^c[r_1, z] = 0$ and $B^c[r_2, z] = 1$. However, this results in $B^c[r_2, yz]$ being 01. Hence, the relationship $P_{A^c}[y, z] = 1$ is indirectly forced by the row r. The situation is illustrated by the columns x_3, y_3 and z_3 in the Table 3.1.

- **Case 4:** $P_{A^c}[x, y] = 0$ **and** $P_{A^c}[x, z] = 1$. Identical to case 3. The implied relationship $P_{A^c}[y, z] = 1$ is forced on the columns y and z.

Therefore, $P_{A^c}[y, z] = P_{A^c}[x, y] \oplus P_{A^c}[x, z]$ in all the four cases. $\diamondsuit$ Note: The relative order of the columns x, y and z is insignificant in Theorem 3.1. The relative orders shown in Table 3.1 are only one of the many possible relative orders between the columns x, y and z.

B^c	x_1	y_1	z_1	x_2	y_2	z_2	x_3	y_3	z_3
r_1	0	0	0	0	1	1	0	1	0
r_2	1	1	1	1	0	0	1	0	1

Table 3.1: Illustration of Theorem 3.1.

The essence of Theorem 3.1 has been presented in [BGL02], using different terminology. However, the direct relationships and implied relationships were treated differently in [BGL02]. The input matrix is first checked to make sure that the matrix does not fail the 4-gamete test. The implied relationships are then built and checked to make sure that none of them contradict with each other or the direct relationships. In the opph (Optimal Perfect Phylogeny Haplotyping) algorithm presented in Section 4, both the direct and implied relationships are built and checked simultaneously.

Theorem 3.2 *In any realizable matrix A^c, given three columns x, y and z, $x < y < z$, if there is a row r in which $A^c[r,x] = A^c[r,y] = A^c[r,z] = 2$, then $P_{A^c}[x,y]$ will be in {0,1} if $P_{A^c}[x,z]$ is in {0,1}.*

Proof The proof is trivial if the column y or column z have at least one '1', due to Property 3.4. The proof is trivial also when $P_{A^c}[y,z]$ is {0,1}- $P_{A^c}[x,y]$ can be derived from $P_{A^c}[y,z]$ and $P_{A^c}[x,z]$ using Theorem 3.1. Let us consider the case when the columns y and z do not have any '1's, and $P_{A^c}[y,z]$ is not directly forced. i.e, the column y is '2' in every row in which column z is '2'. There are two possibilities:

- **Case 1:** The column z is directly forced out-of-phase or in-phase with x. This means that there is a row r in which $A^c[r,x] \in \{0,1\}$ and $A^c[r,z] = 2$. But, since the column y is '2' in every row in which column z is '2', $A^c[r,y] = 2$. This means that $A^c[r,xy]$ is either 02 or 12, and hence a 01 row or a 11 row is forced in $B^c[*,xy]$. Therefore, $P_{A^c}[x,y]$ is in {0,1} if $P_{A^c}[x,z]$ is directly forced.

- **Case 2:** The column z is forced out-of-phase or in-phase with x through an implied relationship. i.e, x is '2' in every column in which z is '2'. There must be at least one other column w such that all the three columns x, w and z are '2' in some row r, and the phase between the pairs (x,w) and (w,z) is directly forced. $P_{A^c}[x,z]$ must be derived by applying Theorem 3.1 on w, x and z. Again, there are two possibilities:

  - **$w < y$:** Since $P_{A^c}[w,z]$ is directly forced, there must be some row r_1 in which $A^c[r_1,w] \in \{0,1\}$ and $A^c[r_1,z] = 2$. But, since y must be '2' in every row in which z is '2', $A^c[r_1, y] = 2$, and $P_{A^c}[w, y]$ will be equal to $P_{A^c}[w, z]$. Hence, $P_{A^c}[x, y]$ can be obtained by applying Theorem 3.1 on columns x, y and w.

– $y < w$**:** Since $P_{A^c}[x, w]$ is directly forced, there must be some row r_2 in which $A^c[r_2, x] \in \{0, 1\}$ and $A^c[r_1, w] = 2$. Therefore, either $P_{A^c}[y, w]$ or $P_{A^c}[x, y]$ must be directly forced. In the first situation, $P_{A^c}[x, y]$ can be obtained by applying Theorem 3.1 on columns x, y and w. In the second situation, $P_{A^c}[x, y]$ is directly available.$\diamondsuit$

3.2.4 Realizability of the matrix A^c

The direct and implied phase relationships described above enable us to extend the 2-gamete test and state the necessary and sufficient conditions for the realizability of a genotype matrix A^c.

Theorem 3.3 (The Realizability Theorem) *A column-sorted genotype matrix A^c is realizable by a perfect phylogeny* iff *$P_{A^c}[x, y] \neq \psi$ for every pair of columns x and y, $x < y$, in A^c.*

Proof The *only if* part of the theorem is obvious. If $P_{A^c}[x, y] = \psi$, then the rows 01 and 11 will be forced in the matrix $B^c[*, xy]$. Hence the matrix B^c will fail the 2-gamete test, and is therefore not realizable by a perfect phylogeny. Now let us look at the *if* part of the Theorem.

In any matrix B^c that has a perfect phylogeny, any pair of columns are either in-phase or out-of-phase. Therefore, to prove that the matrix A^c has a perfect phylogeny, we need to prove that there will be an explanation B^c for the matrix A^c in which $P_{B^c}[x, y] \varepsilon \{0, 1\}$ for every pair of columns x and y, $x < y$.

Let us assume that we know all the pairwise relationships (direct and implied) between columns in the matrix A^c, and that no entry in P_{A^c} is ψ. Let x be the column with the lowest index in A^c such that all pairs of columns up to x are either forced in-phase or out-of-phase with each other. i.e, for every pair i and j, $i < j < x$, $P_{A^c}[i, j] \in \{0, 1\}$. Therefore, $P_{A^c}[i, j] = P_{B^c}[i, j]$ for all these columns. From the definition, column x is neither forced in-phase nor forced out of phase with at least one column before x. i.e., there is at least one column i, $i < x$, such that $P_{A^c}[i, x] = \phi$. Let S be the set of all such columns. i.e., for every column $i \in S$, $i < x$ and $P_{A^c}[i, x] = \phi$. From Property 3.4, S can be non-empty only if the column x does not have any '1's.

First, we show that every column c_1 such that $c_1 < x$ and there is a row r in which $A^c[r, c_1] = A^c[r, x] = 2$, is in the set S. Since every column $i \in S$ is '2' in every row in which x is '2', all the columns in S are '2' in row r. Let us consider any column $c_2 \in S$. Since all the three column c_1, c_2 and x are '2' in row r, Theorem 3.1 can be applied on the columns c_1, c_2 and x if any two pairwise phase relationships are in {0,1}. Since both c_1 and c_2 are to the left of x, we know that $P_{A^c}[c_1, c_2]$ is in {0,1}. Therefore, if $P_{A^c}[c_1, x]$ is in {0,1}, we can apply Theorem 3.1, and $P_{A^c}[c_2, x]$ will be in {0,1}. However, we know that $P_{A^c}[c_2, x] = \phi$ since c_2 is in S. Hence, c_1 must also be in S. Hence, every column $j < x$ that is '2' in a row in which x is 2 is in S. Expanding the column x in-phase or out-of-phase with any one column $c_1 \in S$ will force the column x in-phase or out-of-phase with every other column $c_2 \in S$, due to Theorem 3.1. Therefore, column x can be expanded so as not to violate the 2-gamete test with any column $j < x$.

Now, let us consider the columns with higher index than x. Let $\acute{S}$ be the set of columns with higher index than x that are '2' in some row in which x is '2'. For any column $y \in \acute{S}$, $P_{A^c}[j, y]$ must be ϕ for every column $j \in S$, since $P_{A^c}[x, y]$ will be in {0,1} otherwise (due to Theorem 3.2). Hence, none of the newly implied phase relationships that can be inferred because of setting $P_{A^c}[x, y]$ to 0 or 1 can make $P_{A^c}[x, y]$ to be ψ. Hence, it is possible to expand column x in such way as not to violate the 2-gamete test with any column $y > x$. Once we account for all the newly implied/introduced phase relationships, we can proceed to the next column z which is neither forced in-phase nor forced out-of-phase with at least one column before it. The same conditions apply at z, and there will be at least one explanation B^c of A^c such that B^c has a perfect phylogeny. ♢

3.3 The FlexTree Data Structure

3.3.1 Motivation for the FlexTree data structure

The in-phase and out-of-phase relationships described in the previous section directly translate to relative positions in the PPH tree. If two columns y and z, $y < z$, are forced in-phase, then the edge labeled with column y must be in the path from the root to the edge labeled with column z in any rooted perfect phylogeny for the matrix A^c. Similarly, if two columns x and y, $x < y$, are forced out-of-phase, then the edge labeled with column x can not be in the path to the edge labeled with column y in any perfect phylogeny for the matrix A^c. Assume we have three columns $x < y < z$, so that y is forced in-phase with z. If x is forced in-phase with y, x must always be in the path to y, and since y must always be in the path to z, x will always be in the path to z. If x is forced out of phase with y, x must never be in the path to y, and hence x can never be in the path to z. In either case, we need not explicitly know the relationship between the columns x and z, as this can always be inferred through the column y. Therefore, at any column z, if we know the column y with the highest index such that z is forced in-phase with y, we can infer the relation ship of z with any column with lower index than y.

In any given perfect phylogeny, a site z is said to *follow* a site y if the site y is the first site in the path from site z to the root. For any column z, let the column y be the column with the highest index such that $y < z$ and y is forced in-phase with z. If all the columns between y and z are forced out-of-phase with z, then z must follow the column y in every perfect phylogeny for the matrix A^c. Under these circumstances, the column z can be considered *fixed* to column y, and we call the column y as the *parent* of column z. The situation is depicted in Figure 3.3a. On the other hand, if there are columns between y and z that are not forced out-of-phase with z, z might follow different columns in different phylogenies for A^c. In this case, as there is some *flexibility* in the columns that z can follow, we call the

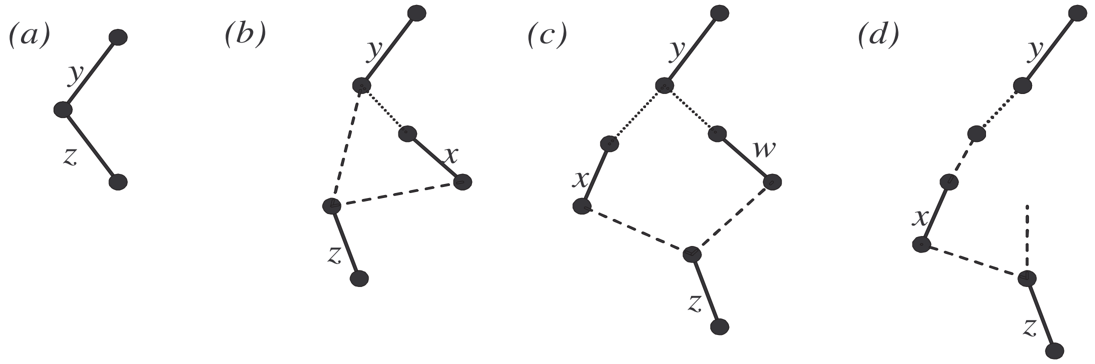


Figure 3.3: (a) A fixed column z with parent y; Possible scenarios when z is flexible = (b) Case 1; (c)Case 2; (d) Case 3

column z *flexible*. Let S be the set of columns between y and z that are not forced out of phase with z. When z is a flexible column, let the column x be the column with the highest index such that $x \in S$. i.e., x is the column with the highest index that z can follow in any perfect phylogeny. We call the column x as the *f-parent0* of column z. The relative positions of z and x in different situations are shown in Figures 3.3b, 3.3c and 3.3d. In each one of these situations, the other columns that z can follow can either be defined or deduced with respect to the column x. Let S_x be the set of columns in S that are forced out of phase with x. We introduce a new term called *f-parent1*.

Case 1: $P_{A^c}[y, x] = 0$ and $P_{A^c}[w, x] = 0$ for every column $w \in S$. The column z can follow either column x or column y. Column y is called the *f-parent1* of column z. The situation is depicted in Figure 3.3b.

Case 2: S_x is not empty, and w is the column with the highest index in S_x. i.e, $P_{A^c}[w, x] = 1$. Since both w and x are not forced in-phase or out-of-phase with z, there must at least one row in which all three columns x, w and z are '2'. Hence, Theorem 3.1 applies, and $P_{A^c}[w, z] = 1$ if $P_{A^c}[x, z] = 0$ or $P_{A^c}[x, z] = 1$ if $P_{A^c}[w, z] = 0$. In the first case, z must follow x and in the second case, z must follow w. Again, there are only two columns that z can follow, and we call w as the *f-parent1* of column z. The situation is depicted in 3.3c.

Case 3: $P_{A^c}[y, x] = \phi$ and S_x is empty. There are more than two columns that z can follow. In fact, z can follow any column that x can follow. Hence, all we need to know about column z is that z can follow x. In this situation, *f-parent1* of column z is not defined(*null*). The situation is shown in Figure 3.3d.

For any flexible column f-parent1 and f-parent0 are collectively referred to as the *flexible parents* of the column. If we introduce a dummy all-1 column with index 0 to the matrix A^c, every column will be forced in-phase with column 0. This will ensure that either the parent or f-parent0 are defined for every column except column 0. The added dummy column will not violate the column ordering since it has the highest possible column sum.

Theorem 3.4 *In any realizable matrix A^c, if two columns y and z are such that $y < z$ and $P_{A^c}[y, z] = 0$, then $P_{A^c}[x, z] = P_{A^c}[x, y]$ for any site $x < y$.*

Proof The proof is divided into three cases:

- **Case 1:** $P_{A^c}[x, y] = 0$. i.e., in any PPH tree, the edge labeled with site x must be in the path from the root to the edge labeled with site y. But, since $P_{A^c}[y, z] = 0$, the edge labeled with site y must be in the path from the root to the edge labeled with the site z. Hence, the site x will be in the path from the root to the edge labeled with site z in any PPH tree for the matrix A^c. Hence, $P_{A^c}[x, z] = 0 = P_{A^c}[x, y]$.

- **Case 2:** $P_{A^c}[x, y] = 1$. Similar to Case 1. x cannot be in the path to y in any PPH tree. Since y must be in the path to z in every PPH tree for A^c, x can not be in the path to z. Hence, $P_{A^c}[x, z] = 1 = P_{A^c}[x, y]$.

- **Case 3:** $P_{A^c}[x, y] = \phi$. This means that are no '1's in column y, due to Property 3.4. Hence there must be at least one row r in A^c such that $A^c[r, yz] = 22$. But, since $P_{A^c}[x, y] = \phi$, the columns x must be '2' in every row in which column y is '2'. Hence, $A^c[r, x] = 2$. As all the three columns x, y and z are '2' in row r, Theorem 3.1 applies, and $P_{A^c}[y, z] = P_{A^c}[x, y] \oplus P_{A^c}[x, z]$. Since we know that $P_{A^c}[y, z] = 0$, $P_{A^c}[x, y]$ must be equal to $P_{A^c}[x, z]$ in order to satisfy Theorem 3.1.$\diamondsuit$

Theorem 3.4 allows us to build the phase matrix by explicitly storing only parts of the phase matrix. The FlexTree data structure utilizes this property, and stores only the absolute minimum phase relationships necessary to reconstruct the phase matrix. It will be clear from the following discussion that we need to explicitly store at most two entries in any column of the phase matrix. The rest of P_{A^c} can be inferred by just knowing a small portion of P_{A^c}. Theorem 3.5 tells us exactly what information in P_{A^c} is necessary in order to deduce the rest of P_{A^c}.

Theorem 3.5 *In any realizable matrix A^c, the phase matrix P_{A^c} can be constructed if we know the parent, f-parent0 and f-parent1 of each column.*

Proof The proof is by induction. Let us assume that we are at a column z, and that we could construct the matrix P_{A^c} completely up to the column $z-1$ by just knowing the parent f-parent0 and f-parent1 of every column up to $z-1$. We will show that we can obtain all the pairwise relationships of the column z with any column $x < z$ by just knowing the parent (Case 1) or f-parent0 and f-parent1 (case 2).

Case 1: The column z has a parent y. By definition, $P_{A^c}[x,z] = 1$ for every column x such that $y < x < z$. Also, $P_{A^c}[y,z] = 0$ by definition. It is clear from Theorem 3.4 that $P_{A^c}[x,z] = P_{A^c}[x,y]$ for every column x such that $x < y$. Therefore, for every column $x < y$, since $y \leq z-1$ and since we have the phase matrix P_{A^c} built up until column $z-1$, we know $P_{A^c}[x,y]$, from which we can obtain $P_{A^c}[x,z]$.

Case 2: Column z does not have a parent. Column y_0 is f-parent0 of column z and column y_1 is an f-parent1 of column z. There are three possibilities:

Case 2-(a): $y_1 \neq null$ and $P_{A^c}[y_1, y_0] = 1$. We divide the columns into three ranges:

1. $y_0 < x < z$: By definition, $P_{A^c}[x,z] = 1$

2. $y_1 < x < y_0$: There are three possibilities:

   (a) $P_{A^c}[x, y_0] = 1$. $P_{A^c}[x,z]$ cannot be ϕ or 0, as y_1 will be equal to x if $P_{A^c}[x,z] = \phi$ or 0. As $y_1 < x$ by definition, $P_{A^c}[x,z]$ must be 1.

(b) $P_{A^c}[x, y_0] = 0$. $P_{A^c}[x, z]$ cannot be 0, as y_1 will be equal to x if $P_{A^c}[x, z] = 0$. By applying Theorem 3.4 on the three columns y_1, x and y_0, $P_{A^c}[y_1, x] = P_{A^c}[y_1, y_0]$. Let r be any row such that $A^c[r, z] = 2$. Both the columns y_1 and y_0 must be '2' in row r, as $P_{A^c}[y_0, z] = P_{A^c}[y_0, z] = \phi$. Therefore, $A^c[r, x]$ must be '2', since $A^c[r, x]$ being 0 or 1 will contradict with what we already know about the columns y_1, x and y_0. Hence all the three columns x, y_0 and z are '2' in row r. Theorem 3.1 will apply, and $P_{A^c}[x, z]$ will be in {0,1} if $P_{A^c}[y_0, z]$ is in {0,1}. But, since we know that $P_{A^c}[y_0, z] = \phi$, $P_{A^c}[x, z]$ must be ϕ.

(c) $P_{A^c}[x, y_0] = \phi$. Not possible. As in (b) above, there must be at least one row in which all four columns y_1, x, y_0 and z are '2'. Hence, Theorem 3.2 applies on the columns y_1, x and y_0, and $P_{A^c}[y_1, y_0] \in \{0, 1\}$, as $P_{A^c}[y_1, y_0] = 1$. Applying Theorem 3.1 on y_1, x and y_0, we see that $P_{A^c}[x, y_0]$ has to be in {0,1}.

3. $x < y_1$: There are 5 valid pairwise relations between the columns x, y_1 and y_0. We can infer $P_{A^c}[x, z]$ in all five cases, as shown in Table 3.2

Case 2-(b): $y_1 \neq$ null and $P_{A^c}[y_1, z] = 0$. Proof similar to case 2-(a).

Case 2-(c): $y_1 =$ null. Proof similar to case 2-(a). $\diamondsuit$

$P_{A^c}[x, y_1]$	$P_{A^c}[x, y_0]$	$P_{A^c}[y_1, y_0]$	$P_{A^c}[x, z]$
0	0	1	0
0	1	1	ϕ
0	ϕ	1	not possible
1	0	1	ϕ
1	1	1	1
1	ϕ	1	not possible
ϕ	0	1	not possible
ϕ	1	1	not possible
ϕ	ϕ	1	ϕ

Table 3.2: $P_{A^c}[x, z]$ can be obtained from $P_{A^c}[x, y_1]$ and $P_{A^c}[x, y_0]$

.

3.3.2 The FlexTree

The FlexTree data structure is a special kind of weakly connected directed acyclic graph (DAG). The FlexTree provides an intuitive and simple representation of all the pairwise relationships between pairs of columns. The FlexTree has a tree-like structure. In fact, if the matrix A^c has a unique perfect phylogeny, the underlying undirected graph of the FlexTree for A^c will be a rooted tree.

In the FlexTree, each site is represented by a directed edge labeled with the site. Every edge in the FlexTree is directed toward the root. If column i is the parent of column j, the relationship is represented by the edge labeled with column i being adjacent to the edge labeled with column j. The flexible parent relationships are represented by directed unlabeled *glue* edges. If column i is the f-parent1 or f-parent0 of column j, the relationship is represented by an unlabeled directed edge from the edge labeled with column j to the edge

(a)

$$A^c = \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 1 & 0 & 2 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 2 & 2 & 2 & 0 & 0 \\ 1 & 2 & 0 & 0 & 2 & 2 \end{bmatrix} \end{array}$$

(b)

P_{A^c}	1	2	3	4	5	6
1	–	0	0	0	0	0
2	–	–	1	ϕ	ϕ	ϕ
3	–	–	–	ϕ	1	1
4	–	–	–	–	1	1
5	–	–	–	–	–	ϕ
6	–	–	–	–	–	–

(c)

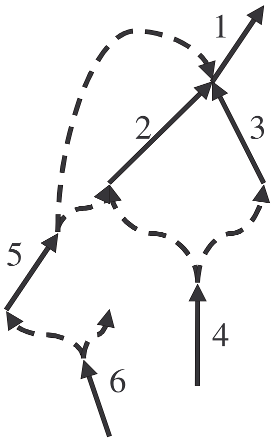


Figure 3.4: (a) A genotype matrix A^c; (b) The Phase matrix for A^c; (c) The FlexTree T for A^c - the broken edges represent the glue edges;

labeled with the column i. Figure 3.4 shows a matrix A^c, the phase matrix P_{A^c} of A^c, and the flex tree T for the matrix A^c.

The phase relationships reduce to reachability in the FlexTree. For two sites $i < j$, if $P_{A^c}[i, j] = 1$, the edge labeled with site i is not reachable from the edge labeled with site j. If $P_{A^c}[i, j] = 0$, every path from edge labeled with site j to the root will include the edge labeled with i. If $P_{A^c}[i, j] = \phi$, then there will at least one glue edge in the path from the edge labeled with site j to the edge labeled with site i. As the FlexTree represents all the phase relationships given by the phase matrix P_{A^c}, any PPH tree for A^c can be built from the FlexTree by removing some glue edges and contracting the others. (We will show how to do this in Section 3.4.7).

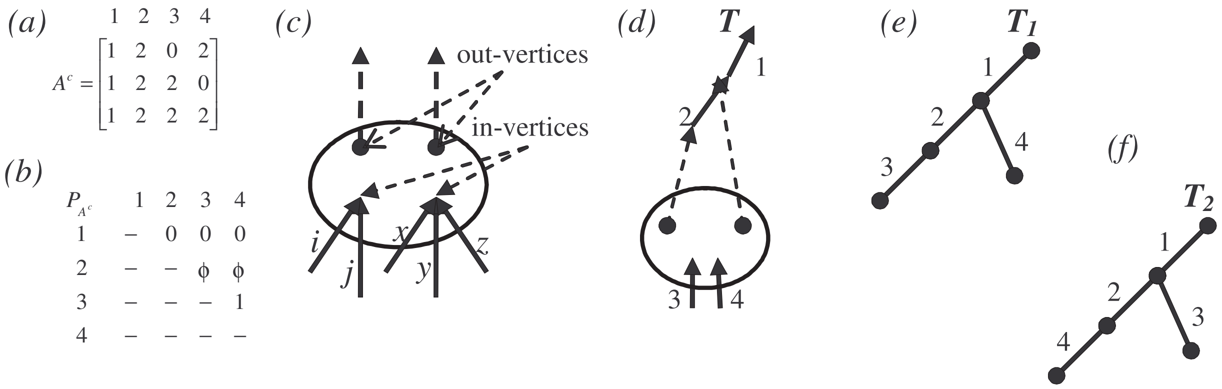


Figure 3.5: (a) A genotype matrix A^c; (b) The Phase matrix for A^c; (c) General structure of a partition; (d) The complete FlexTree T for the matrix in (a); (e) and (f) - The two PPH trees T_1 and T_2 represented by the FlexTree in (d)

3.3.3 Representing the interdependence between phase relationships

The FlexTree, as described above, correctly represents all the phase relationships between pairs of columns. However, some of the phase relationships are dependent on each other as per Theorem 3.1. These dependencies need to be represented in the FlexTree. For example, consider the matrix A^c and the phase matrix P_{A^c} shown in Figures 3.5a and 3.5b. Columns 2, 3 and 4 are all '2' in row 4, and hence the pairwise phase relationships are linked - $P_{A^c}[2,3] = P_{A^c}[2,4] \oplus P_{A^c}[3,4]$. Therefore, setting any one of the phase relationships $P_{A^c}[2,3]$ or $P_{A^c}[2,4]$ to '0' will result in the other being '1'.

In order to represent the interdependence between phase relationships, we introduce a special system of vertices that we call a *partition.* A partition consists of four vertices, as shown in Figure 3.5c. Two of these vertices are the in-vertices of the partition - the in-degree is at least 1 and the out-degree is 0 for each of them. The other two are out-vertices - the in-degree is 0 and the out-degree is 1 for each of them. Each of the two out-vertices is incident on an un-labeled glue edge. The four vertices in the partition represent two vertices in any PPH tree. In any PPH tree, one of the in-vertices merges with one of the out-vertices, and other in-vertex merges with the remaining out-vertex. The condition is that the two in-vertices have to be distinct vertices in any PPH tree. Hence, both in-vertices are not allowed to merge with the same out-vertex. The complete FlexTree for the matrix in Figure 3.5a is shown in Figure 3.5d. The two PPH trees described by the FlexTree in Figure 3.5d are shown in Figure 3.5e and Figure 3.5f.

In the FlexTree, all the edges that are incident on any of the in-vertices are interpreted as being connected to both the glue edges coming out of the partition. This is because of the fact that any edge i incident on one of the in-vertices has two possibilities as given by the two glue edges. Each choice leads to one PPH tree for A^c.

Any given column can be involved in at most one partition. For example, refer to figure 3.6. The column-pairs (4,6), (3,5) and (3,4) are all out-of-phase. But, since there is a row in which the columns 2, 4 and 6 are '2', columns 4 and 6 must be in a partition. The same situation applies for columns (2,3,5) and (2,3,4). All these relationships can be expressed using a single partition as shown in Figure 3.6b.

(a)

$$A^c = \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 2 \\ 1 & 2 & 0 & 0 & 2 & 0 \\ 1 & 2 & 0 & 2 & 0 & 2 \\ 1 & 2 & 2 & 0 & 0 & 0 \\ 1 & 2 & 2 & 0 & 2 & 0 \\ 1 & 2 & 2 & 2 & 0 & 0 \end{bmatrix} \end{array}$$

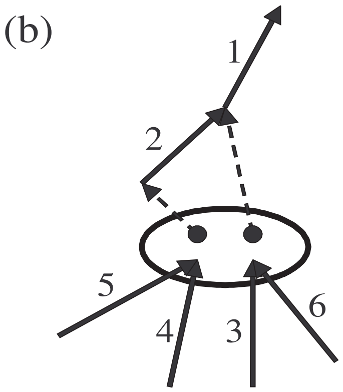


Figure 3.6: (a) A matrix A^c; (b) The FlexTree T for the matrix A^c

3.3.4 Haplotypes represented by the FlexTree

It is essential that the reader understands how the FlexTree represents the possibilities for each column. If a column i is reachable from a column j, it does not necessarily imply that the column j can follow column i, even if the column j is flexible. The column j can follow column i only if the following conditions are satisfied:

1. The column i is reachable from column j.

2. Either: (a) Column i is the parent of column j. For example, columns i and j in Figure 3.7a. or:

   (b) The first and last edges in at least one of the paths between the edges i and j (not including the edges labeled with sites i and j) are both unlabeled glue edges. For example, columns i and j in Figure 3.7b.

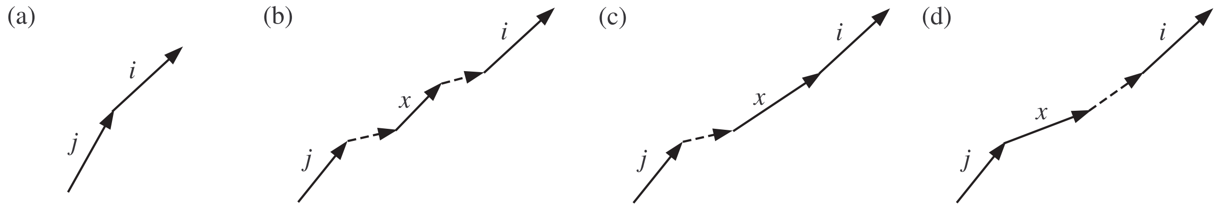


Figure 3.7: (a) and (b) - Situations in which the j can follow i;(c) and (d) - Situations in which j can not follow i even though i is reachable from j.

Two situations in which column i is reachable from column j, but not in one of the columns that j can follow are shown in Figures 3.7c and 3.7d.

If we fix each flexible column in the FlexTree to one of the columns that it can follow, the resulting graph will be a DAG that is free of partitions and glue edges, and will contain only directed labeled edges. The underlying undirected graph of this DAG will be a perfect phylogeny. Each node in the perfect phylogeny describes a haplotype. Therefore, the FlexTree represents every haplotype that labels a node in some perfect phylogeny described by the FlexTree. Given a haplotype H, we can easily check if H is among the haplotypes represented by the FlexTree. Let i and j be two columns such that $H[j] = 1$, and i is the column with the highest index such that $i < j$ and $H[i] = 1$. i.e., all the columns (if any) between i and j are '0' in H. The haplotype H will be in the haplotypes represented by the FlexTree if and only if i is one of the columns j can follow, for every such pair of columns i and j in H.

3.3.5 Representation of the FlexTree

Because of the partitions, the FlexTree is not exactly a DAG. As is evident from the description of a partition, all the columns involved in a partition have the same set of flexible parents. Each partition involves two groups of sites, each group representing the sites that are incident on one of the two in-vertices of the partition. The two groups are arbitrarily numbered as group-0 and group-1. Therefore, for each partition, we need to store the information about the f-parents and the two groups of sites involved in the partition. For each site that is not in a partition, we need to know the *parent*, *f-parent1*, *f-parent0* of the site. If the site is involved in a partition, we need to store a pointer to the partition. In order to optimize the performance of the algorithm, each site involved in a partition also needs to store which group of the partition it is in. The FlexTree is stored as two tables, the column-table and the partition-table, which give information about the sites and partitions, respectively. The representation of the FlexTree in Figure 3.6 is given in Table 3.3. The *partition* field in the column-table stores a pointer to the partition that the column is involved in. The *group* field gives the group number of the column within the partition.

For each column, we need a constant amount of space in the column-table. Hence the total space required by the column-table is $O(m)$. The partition table stores the index of each partition, the two f-parents, and the list of sites in each group of the partition. The *size* of a partition is defined as the total number of columns involved in the partition. The size of a partition is equal to the sum of the in-degrees of the two in-vertices. As each column can

Column	parent	f-parent0	f-parent1	partition	group	FlexEnd
1	root	-	-	-	-	1
2	1	-	-	-	-	1
3	-	-	-	1	0	3
4	-	-	-	1	1	4
5	-	-	-	1	1	5
6	-	-	-	1	0	6

Partition Number	f-parent0	f-parent1	group[0]	group[1]
1	2	1	3,6	4,5

Table 3.3: The column-table (above) and the partition-table (below) for the FlexTree in Figure 3.6.

be involved in only one partition at any given time, the combined size of all the partitions in the FlexTree is $O(m)$. The total number of partitions in the partition table cannot exceed $m/2$.

3.4 The opph Algorithm

The fundamental idea behind the opph algorithm is to start with an empty FlexTree and process the rows of the matrix one after the other. When a row is processed, the FlexTree should be updated to represent the pairwise relationships and dependencies imposed by the row. An edge labeled with column i must be added to the FlexTree when the first row in which column i takes a non-zero value is processed. At any point in the algorithm, the FlexTree must correctly represent all the pairwise relationships and dependencies induced by the rows that have already been analyzed.

From Property 3.4, we know that a '1' in a column i will either force the column i in-phase or out-of-phase with every column before it, or render the matrix unrealizable. In the FlexTree, a '1' in the column i ensures that column i is fixed. Therefore, a row with a '1' in column i gives us the maximum information about column i. Hence, we would like to process the rows with a '1' in column i before we process the rows in which the column i is '0' or '2'. This observation suggests that the rows in the matrix should be ordered using the lexicographic order $1 < 0 < 2$. We denote this row-sorted matrix using M. As the phase relationships and the column ordering in M are no different from those is A^c, the phase matrix for M is the same as that for A^c. In the rest of the chapter, we refer to the phase matrix as P_M. An extra, all-1 column with index 0 is added to M, as explained in the previous section.

The row ordering is not just a matter of convenience - it provides a 'context' for adding new rows to the FlexTree. This context is essential in limiting the complexity of the opph algorithm to $O(nm)$. Because of the row ordering, each row shares a prefix with the row before it. The length of this shared prefix must be at least 1, since the column with index 0 is '1' in every row. The maximum length of the shared prefix can be $m + 1$, in which case the row is a copy of the row before it. Assume that the first $r - 1$ rows in M have been processed, and the FlexTree has been constructed for the first $r - 1$ rows. Let the length of the shared prefix for the rows r and $r - 1$ be e_r, where $e_r \leq m$. Since the FlexTree represents the $(r - 1)$th row, we know that the FlexTree also represents the two haplotypes for the genotype represented by the length-e_r prefix of row r. The *end*s of these two haplotypes correspond to either one or two vertices in the FlexTree. The haplotypes for the complete row r are extensions of these two haplotypes, and hence the ends of these two haplotypes provide a context for the complete haplotypes. Due to the row ordering, we can be sure that it is the first time that we are encountering the $(e_r + 1)$-length prefix of row r, and this helps in deciding how each column with index e_r or greater is effected by adding row r to the FlexTree. **Every column with higher index than e_r that takes a non-zero value in row r and is already in the FlexTree must be either forced in-phase or out-of-phase with column e_r.** This property, proved by the lemmas and theorems in the rest of this section, forms the basis for the opph algorithm, shown in Figure 3.9.

In the following, we introduce some terms that will be used in describing the opph algorithm. A column is said to be *in* the FlexTree if an edge labeled with the column is in the FlexTree. The *FlexEnd* of a fixed column i is the first flexible column in the path from the edge labeled with i to the root in the FlexTree. By convention, a flexible column is the *FlexEnd* for itself. A *partial genotype vector* is a prefix of a row in M, to which a string of 0's have been appended so that the length of resulting vector is exactly $m + 1$ (The vector needs to be of length $m + 1$ so that it remains to be a valid genotype vector even when the columns are re-arranged to represent the original order of the sites in matrix A.). The ith partial genotype vector of a row r, denoted by $M[r, 0...i]$, is the prefix of row r of length $i+1$, to which $m-i$ zeros have been appended at the end. The mth partial genotype vector of row is the row itself. A haplotype vector (or a genotype vector) h is said to *end* in a site j if j is the non-zero column with the highest index in h. We denote the two haplotypes of a partial genotype vector $M[r, 0 ... i]$ using h_r^i and k_r^i. By convention, h_r^i is the haplotype vector that ends in the column with the higher index among the two haplotypes h_r^i and k_r^i. For simplicity of notation we denote the site in which h_r^i ends by h_r^i itself, and the site in which k_r^i ends by k_r^i itself. From the context, it will be clear whether it is the haplotype h_r^i or the site h_r^i that is being referred to.

A partial genotype vector is said to be *split* if both the sites h_r^i and k_r^i are defined (not null). Because of the convention, the site h_r^i is always defined. The site k_r^i will be defined if there is only one possible column in which the haplotype k_r^i can end. If there are multiple sites in which the haplotype k_r^i can end, then site k_r^i is not defined. During the construction of the FlexTree, the algorithm maintains two additional arrays $h[]$ and $k[]$, each of size $m+1$, in addition to the fields shown in Table 3.3. When the algorithm is processing row r, the fields $h[i]$ and $k[i]$ represent the sites h_{r-1}^i and k_{r-1}^i.

Constructing the FlexTree for the first row is trivial as shown in the ProcessNewRow procedure in Figure 3.8. When the algorithm reaches a row r, the FlexTree correctly represents the solutions for the first $r-1$ rows. The algorithm is based on the fact that the row r is a result of combining at most two distinct haplotypes. Prefixes of the two haplotypes will correspond to at most two distinct paths in the FlexTree. Let P_0 and P_1 be the two paths. After processing the row r, any non-zero column (by non-zero column, we mean a column that takes a value other than zero, i.e., 1 or 2) in the row r must be in P_0 or P_1. Based on this principal, the opph algorithm identifies the paths P_0 and P_1, adds new columns to the

FlexTree and makes changes to the columns already in the FlexTree so that combining the two haplotypes described by the paths P_0 and P_1 results in the genotype given by row r.

Figure 3.8: ProcessNewRow procedure

```
inputs: $T$ (the column table and partition table) $r$, $e_r$
Result: updates $T$ to accommodate row $r$
for $c_i \leftarrow e_r$ to $m$ do
    if $M[r, c_i] \neq 0$ and $c_i$ is in $T$ then
        $M$ is not realizable by a perfect phylogeny, stop
    if $M[r, c_i] = 0$ then $h[c_i] \leftarrow h[c_i - 1]$, $k[c_i] \leftarrow k[c_i - 1]$
    else if $M[r, c_i] = 1$ then
        if $h[c_i - 1] \neq k[c_i - 1]$ then $M$ is not realizable by a perfect phylogeny, stop
        parent$[c_i] \leftarrow h[c_i - 1]$
        FlexEnd$[c_i] \leftarrow$ FlexEnd$[parent[c_i]]$
        $h[c_i] \leftarrow c_i$, $k[c_i] \leftarrow c_i$
    else
        if $h[c_i - 1] = k[c_i - 1]$ then
            parent$[c_i] \leftarrow h[c_i - 1]$
            $h[c_i] \leftarrow c_i$, $k[c_i] \leftarrow k[c_i - 1]$
            FlexEnd$[c_i] \leftarrow$ FlexEnd$[parent[c_i]]$
        else
            set_fp0($c_i$, $h[c_i - 1]$)
            set_fp1($c_i$, $k[c_i - 1]$)
            FlexEnd$[c_i] \leftarrow c_i$
            $h[c_i] \leftarrow c_i$, $k[c_i] \leftarrow null$
```

The paths P_0 and P_1 are essential for the algorithm because of the following properties:

1. All the '1's in the row r must be in the shared path between the paths P_0 and P_1.

2. Any non-zero column not reachable from any column in P_0 must be in P_1.

3. Any non-zero column but reachable from any column in P_1 must be in P_0.

For any row r in the matrix M such that $r \geq 1$, the *EntryPoint*(denoted by e_r) is the column i with the lowest index such that $M[r-1, i] \neq M[r, i]$. i.e, the entry point is the first column from the left in which the rows $r-1$ and r differ. The *SplitPoint* (denoted by s_r) of a row r is the column with the highest index before e_r at which the row $r-1$ is split. i.e., s_r is the highest column i such that $i < e_r$ and the site $k[i]$ (i.e, the site k^i_{r-1}) is defined.

Figure 3.9: The opph algorithm

```
inputs: $A^c$, $n$, $m$
Result: The FlexTree $T$ for $A^c$
Sort the rows in $A^c$ and add an all-1 column with index 0 to produce the matrix $M$
Initialize every entry in the column table and partition table to null
$h[0] \leftarrow 0$, $k[0] \leftarrow 0$, FlexEnd[0] $\leftarrow 0$
ProcessNewRow(1, 1)
for $r = 2$ to $n$ do
    ($e_r$, $s_r$) $\leftarrow$ ScanForward( $M$, $T$, $r$)
    if $e_r \leq m$ then
        if $M[r, e_r] = 0$ then
            ProcessNewRow($r$,$e_r$)
        else
            TraceUpRow($r$, $e_r$, $s_r$)
            TraceDown($r$)
```

The algorithm consists of three steps - ScanForward, TraceUp and TraceDown. We describe each one of the steps in detail in the following sections.

3.4.1 Building the FlexTree for the first row

As none of the pairwise relationships are known before we start with the first row, the row will have a FlexTree as long as it does not violate Property 3.3. i.e., if there are no '2's to the left of a '1'. The column with index 0 is the dummy all-1 column, hence Parent[0] is initialized to 0, by convention. All other values in the column table are set to *null*, except for $h[0]$ and $k[0]$, which are set to 0.

The procedure for building the FlexTree T for the first row directly follows from the observations 3.1, 3.2 and 3.3 in section 3.1. The procedure ProcessNewRow, shown in Figure 3.8, is called with the parameters $r = 1$, $e_r = 1$. The ProcessNewRow function takes the suffix of the row r starting at e_r and adds all non-zero elements in this suffix to the FlexTree. The ProcessNewRow procedure requires that none of the sites already in the FlexTree be non-zero in the suffix of the row starting with e_r. Since none of the sites are in the FlexTree before processing the first row, this condition is always satisfied for the first row of the matrix M.

Lemma 3.2 *After procedure ProcessNewRow(1,1) the FlexTree accurately represents the phase relationships imposed by the first row.*

Proof This is trivially true for any pair of columns i and j such that $i < j$, $M[1, i] = 0$, and $M[1, j] \neq 0$. $P_M[i, j] = 1$ for any such pair since the column i is not in the FlexTree and hence not reachable from the column j.

Without loss of generality, assume that there are at least two columns that are '2' in the first row. Let c_1 be the column with the lowest index such that $M[1, c_1] = 2$, and let c_2 be first column to the right of c_1 such that $M[1, c_2] = 2$. Let l_1 be the column with the highest index such that $l_1 < c_1$ and $M[1, l_1] = 1$. Each non-zero column with index less than or equal to c_1 is fixed to the non-zero column immediately to the left, and hence $P_M[i,j] = 0$ for any two such columns i and j, $i < j$. For the column c_2, f-parent0 is c_1 and f-parent1 is l_1. Hence, $P_M[l_1, c_2] = 0$, since l_1 will always be reachable from c_2. Since c_2 is not fixed to c_1, $P_M[c_1, c_2] = \phi$. For any column $i > c_2$ such that $M[1,i] = 2$, f-parent0 is the immediate non-zero column to the left, and f-parent1 is *null*. Hence, for any two such columns i and j such that $i < j$, $P_M[i,j] = P_M[c_1,i] = P_M[c_1,j] = P_M[c_2,i] = P_M[c_2,j] = \phi$. l_1 is always in the path to any such column i, and hence $P_M[l_1,i] = 0$. All the remaining relationships are correctly represented due to Theorem 3.4. $\diamondsuit$

3.4.2 The Scan Forward procedure

The algorithm processes the rows in M in lexicographic order and makes modifications to the FlexTree to accommodate the pairwise relationships induced by the rows. Hence, when the algorithm is at a row r, all the pairwise relationships induced by the first $r-1$ rows are correctly represented in T. In the scan forward step, the algorithm mainly finds e_r and s_r, the EntryPoint and SplitPoint for the row r. The partial genotype vector $M[r, 0...e_r - 1]$ is exactly identical to the partial genotype vector $M[r-1, 0...e_r - 1]$, from the definition of e_r. Hence, there can be no new pairwise relation ships induced by the partial genotype vector $M[r, 0...e_r]$, as all the pairwise relationships in $M[r-1, 0...e_r]$ are already represented in T. The scan forward procedure also finds s_r. As both $h[s_r]$ and $k[s_r]$ are defined, one of them must be in the path to h_r^m and the other must be in the path to k_r^m. A high-level description for the scan forward step is shown in Figure 3.17.

3.4.3 Trace Up

As can be seen from Figure 3.9, the TraceUp procedure is called only when $M[r, e_r] = 2$ (Since $M[r, 1...e_r]$ follows $M[r-1, 1...e_r]$ lexicographically, $M[r, e_r]$ can not be '1'). In this step, the algorithm first tries to find the site p_0 in T with the highest index such that $M[r, p_0] = 2$ and $p_0 \geq e_r$.

Lemma 3.3 *Given that matrix M is realizable by a perfect phylogeny and that the TraceUp step is invoked for row r, if there is a column that satisfies the conditions for p_0 in row r, then there will be a column $j \leq e_r$ such that $M[r, j] = 2$ and $P_M[j, p_0] \in \{0, 1\}$.*

Proof By the time the algorithm reaches the rth row, all the non-zero columns within the first $r-1$ rows will be in the FlexTree. Since p_0 is already in the tree by definition, there must be a row $r_0 < r$ such that $M[r_0, p_0] \neq 0$. Now, since both rows r_0 and $r-1$ precede the row r lexicographically, there must be at least one column $j \leq e_r$ such that $(M[r_0, j], M[r, j])$ is (1,0), (0,2) or (1,2). If $M[r_0, j] = 1$ and $M[r, j] = 0$, the matrix M will not be realizable by a perfect phylogeny, which contradicts our assumption that the matrix is realizable. Hence there are only two possibilities in a realizable matrix:

Case 1: $M[r_0, j] = 0$ and $M[r, j] = 2$. Since $M[r_0, p_0] \neq 0$, $P_M[j, p_0] = 1$.

Case 2: $M[r_0, j] = 1$ and $M[r, j] = 2$. Since $M[r_0, p_0] \neq 0$, $P_M[j, p_0] = 0$.

Hence, there must be a column $j \leq e_r$ such that $M[r, j] = 2$ and $P_M[j, p_0] \in \{0, 1\}$. $\diamondsuit$

Theorem 3.6 *In TraceUp step for row r, if p_0 is defined, then every site i such that $e_r \leq i < p_0$, $M[r, i] = 2$ and i is reachable from p_0 must be forced in-phase with p_0.*

Proof From Lemma 3.3, we know that there must be a column $j \leq e_r$ such that $M[r, j] = 2$ and $P_M[j, p_0] \in \{0, 1\}$. Since the column i is reachable from p_0, we know that $P_M[i, p_0] \in \{0, \phi\}$. If $P_M[i, p_0]$ is already 0, there is nothing to prove. Let us consider the case when $P_M[i, p_0] = \phi$. This implies that the column i is '2' in every row less than r in which the column j is '2', including the row r_0 in which $M[r_0, j] \neq 2$. Hence, from the same discussion as in Lemma 3.3, we know that $P_M[j, i]$ must be equal to $P_M[j, p_0]$. All the three columns j, i and p_0 are '2' in row r, and hence the Theorem 3.1 applies, and $P_M[i, p_0] = P_M[j, i] \oplus P_M[j, p_0]$. Since $P_M[j, i] = P_M[j, p_0]$, $P_M[i, p_0] = 0$ irrespective of whether $P_M[j, p_0]$ is 0 or 1. $\diamondsuit$

The TraceUp procedure finds the column p_0, and uses Theorem 3.6 to force all the non-zero columns between e_r and p_0 that are reachable from p_0 in-phase with p_0. Simultaneously, it tries to find the column p_1 with the highest index such that $M[r, p_1] = 2$ and $P_M[p_1, p_0] = 1$.

Lemma 3.4 *In the row r, if p_0 and p_1 are defined, every non-zero column $i \geq k[s_r]$ such that $M[r, i] = 2$ that is not reachable from p_0 must be forced in-phase with p_1.*

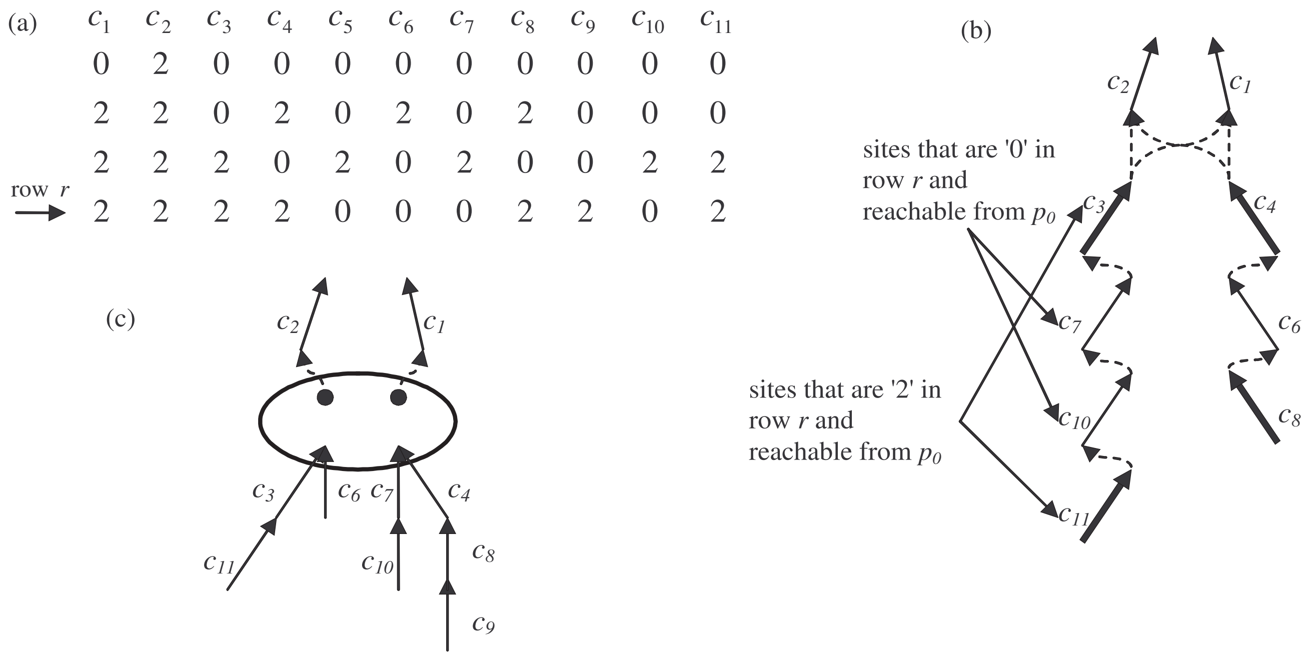
Proof Since i is not reachable from p_0, $P_M[i, p_0] = 1$. By definition, $P_M[p_1, p_0] = 1$. Applying Theorem 3.1 on i, p_1 and p_0, we have $P_M[i, p_1] = P_M[i, p_0] \oplus P_M[p_1, p_0] = 0$. $\diamondsuit$

Lemma 3.5 *In the row r, if both p_0 and p_1 are defined, any column $i \geq k[s_r]$ such that $M[r, i] = 2$ that is forced in-phase with any one column out of p_0 and p_1 must be forced out-of-phase with the other.*

Proof Direct application of Theorem 3.1 on i, p_0 and p_1. $\diamondsuit$

Hence, once the site p_0 is found, all the new pairwise relationships induced by the row r on the non-zero columns with index less than r can be deduced using Theorem 3.6 and lemmas 3.4 and 3.5. In addition, any column that is zero in row r and reachable from $p_0(p_1)$ is obviously forced out of phase with $p_0(p_1)$, and hence must be rendered unreachable from $p_0(p_1)$. Figure 3.10 illustrates the effect of the above lemmas and theorems. A part of the matrix is shown in Figure 3.10a and the FlexTree just before processing row r is shown in Figure 3.10b. From the definition of p_0 and p_1, $p_0 = c_{11}$ and $p_1 = c_9$ for the row r. Columns c_7 and c_{10} are reachable from p_0 but '0' in row r, and hence must not be reachable from p_0 after processing row r. Column c_8 is not reachable from p_0, and hence must be forced in-phase with p_1. Column c_6 is reachable from c_8, but '0' in the row r, and hence must not be reachable from c_8 (and therefore from p_1) after processing row r. The FlexTree after processing row r is shown in Figure 3.10c.

The trace up procedure starts by scanning the row from right to left, and tries to find p_0. If the procedure reaches e_r without finding p_0, then the row r does not involve any non-zero columns after e_r that are already in the tree, and the algorithm moves to the TraceDown procedure directly. If $M[r-1, e_r] = 1$ and $M[r, e_r] = 0$, then there should be no non-zero



	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
	0	2	0	0	0	0	0	0	0	0	0
	2	2	0	2	0	2	0	2	0	0	0
	2	2	2	0	2	0	2	0	0	2	2
row r →	2	2	2	2	0	0	0	2	2	0	2

c_2
c_1
sites that are '0' in row r and reachable from p_0
c_3
c_4
c_7
c_6
sites that are '2' in row r and reachable from p_0
c_{10}
c_8
c_{11}
c_2
c_1
c_3
c_6
c_7
c_4
c_{11}
c_{10}
c_8
c_9

Figure 3.10: (a) part of the matrix M; (b) part of the FlexTree before processing row r in matrix M; (c)The FlexTree after processing row r

column with higher index that e_r that is already in T if matrix M is to be realizable, and the algorithm directly invokes the ProcessNewRow procedure instead of the TraceUp procedure.

Once the column p_0 is found, the TraceUp procedure effectively *traces up the tree* starting at the edge labeled with p_0. It uses four pointers n_p_0, $n\_p'_0$, p_p_0 and $p\_p'_0$ to keep track of where it is in the tree:

p_p_0: The latest site (the site with the lowest index, since the scanning is from right to left in M) that is reachable from p_0 and is '2' in row r. From Theorem 3.6 and Lemma 3.5, it is clear that either $P_M[p\_p_0, p_0]$ is already known to be '0', or must be set to '0' because of row r. Initially, p_p_0 is set to p_0 immediately after finding p_0.

n_p_0: The next site that is reachable from p_0 and non-zero in the row r.

$p\_p'_0$: The latest site that is reachable from p_0 and is '0' in row r. Clearly, $M[r, (p\_p'_0)p_0]$ is 02, and hence $P_M[p\_p'_0, p_0]$ must be 1. Hence, $p\_p'_0$ must be rendered unreachable from p_0 during the processing of row r.

$n\_p'_0$: The next site that is reachable from p_0 and is '0' in row 'r'. Like $p\_p'_0$, $n\_p'_0$ must be forced out of phase with p_0, and hence must be rendered unreachable from p_0. Also, by applying Theorem 3.1 on the columns $n\_p'_0$, $p\_p'_0$, and p_0, we can infer that $P_M[n\_p'_0, p\_p'_0]$ must be 0.

Similarly, the algorithm maintains four variables n_p_1, $n\_p'_1$, p_p_1 and $p\_p'_1$ in order to keep track of the columns that are reachable from p_1. All these variables are initially set to *null.* Once p_0 is defined, p_p_1 is used to keep track of the columns that are '2' in row r but not reachable from p_0.

Beyond e_r, the TraceUp procedure must continue until one of the following conditions are satisfied:

- Case (a): Until the TraceUp procedure establishes fixed paths from p_0 to $h[s_r]$ and from p_1 to $k[s_r]$.
- Case (b): Until the TraceUp procedure establishes fixed paths from p_1 to $h[s_r]$ and p_0 to $k[s_r]$.
- Case (c): A site $y < e_r$ such that $M[r, y] = 2$ and the site y is neither forced in-phase with p_0 nor forced in-phase with p_1 is reached.
- Case (d): A site with index greater than or equal to $k[s_r]$ is reached, at which it can be determined that the matrix M is not realizable by a perfect phylogeny.

The site $k[s_r]$ is the site with the lowest index that the TraceUp procedure can reach. Since both $k[s_r]$ and $h[s_r]$ are defined, one of the two haplotypes for the partial genotype vector $M[r, 0...s_r]$ must end in $h[s_r]$ and the other must end $k[s_r]$, in any PPH tree for the matrix M. Therefore, either one of the two sites ($k[s_r]$, $h[s_r]$) must be reachable from p_0, and the other must be reachable from p_1, or both must be reachable from both p_0 and p_1. The TraceUp procedure can terminate as soon as it can ensure this reachability criteria. Figure 3.11 shows the three possible scenarios in which the TraceUp procedure can stop.

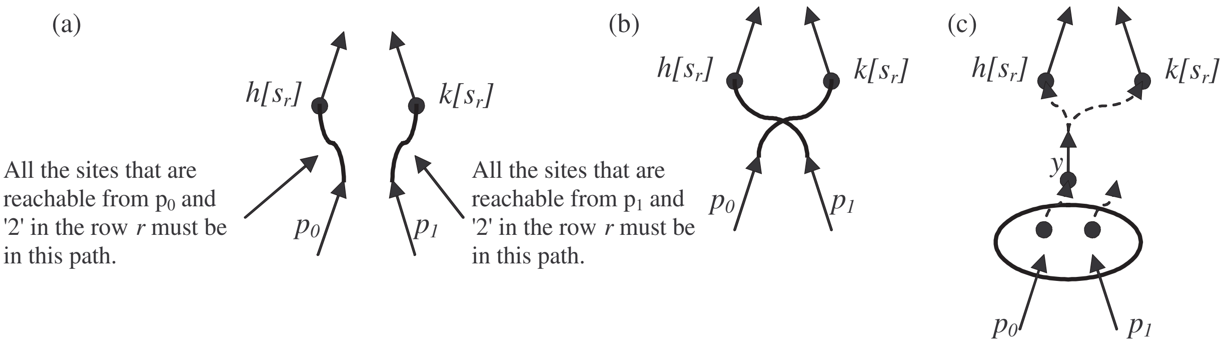

(a)
h[s_r]
k[s_r]
All the sites that are reachable from p0 and '2' in the row r must be in this path.
All the sites that are reachable from p1 and '2' in the row r must be in this path.
p0
p1
(b)
h[s_r]
k[s_r]
p0
p1
(c)
h[s_r]
k[s_r]
y
p0
p1

Figure 3.11: (a) $h[s_r]$ is reachable only from p_0 and $k[s_r]$ only from p_1; (b) $h[s_r]$ is reachable from p_1 and $k[s_r]$ from p_0; (c) both $h[s_r]$ and $k[s_r]$ are reachable from both p_0 and p_1

A high level description of the TraceUp procedure is given in Figure 3.12. The advanceNonZeroPath procedure (called from the TraceUp procedure) is shown in Figure 3.20. The advanceZeroPath procedure is similar to the advanceNonZeroPath procedure. Whenever a flexible site i is about to be fixed, the variable $L[i]$ is used to store the f-parent of i that will not be the parent of i. The L[i] values are used later in the TraceDown step for correctly maintaining f-parents for sites that are connected to i through a flexible edge.

3.4.4 Fixing a flexible site

Assigning a parent to a flexible site may effect other sites in the FlexTree. The following things have to be taken care of when assigning a parent to a site:

Figure 3.12: TraceUp procedure

```
inputs: $T$, $e_r$, $s_r$
Result: Modifications to $T$ to accommodate row $r$
$p\_p_0 \leftarrow null$, $p\_p'_0 \leftarrow null$, $n\_p_0 \leftarrow null$, $n\_p_0 \leftarrow null$
$p\_p_1 \leftarrow null$, $p\_p'_1 \leftarrow null$, $n\_p_1 \leftarrow null$, $n\_p_1 \leftarrow null$
$p_0 = null$, $p_1 = null$
P0Flag ← false, P1Flag ← false
$L[i] = null \; \forall \; i$, $0 \leq i \leq m$
$done \leftarrow false$, $c_i \leftarrow m$
while $done = false$ and $c_i > k[s_r]$ do
    if $c_i = n\_p_0$ then fixNonZeroPath($r$, $c_i$, $n\_p_0$,$n\_p'_0$,$p\_p_0$,$p\_p'_0$, P0Flag)
    if $c_i = n\_p_1$ then fixNonZeroPath($r$, $c_i$, $n\_p_1$,$n\_p'_1$,$p\_p_1$,$p\_p'_1$, P1Flag)
    if $c_i = n\_p'_0$ then fixZeroPath($r$, $c_i$, $n\_p_0$,$n\_p'_0$,$p\_p_0$,$p\_p'_0$, P0Flag)
    if $c_i = n\_p'_1$ then fixZeroPath($r$, $c_i$, $n\_p_1$,$n\_p'_1$,$p\_p_1$,$p\_p'_1$, P1Flag)
    if $M[r, c_i] = 1$ then $M$ not realizable by a perfect phylogeny. Stop
    else if $M[r, c_i] = 2$ then
        if $if\ c_i \geq e_r$ AND ($c_i = n\_p'_0$ OR $c_i = n\_p'_1$ OR ($n\_p_0 \neq null$ AND $n\_p_0 = n\_p_1$)) then
            $M$ not realizable by a perfect phylogeny. Stop
        if $p_0 = null$ then
            if $c_i$ in $T$ then { $p_0 \leftarrow c_i$, advanceNonZeroPath(r, ci, L, $n\_p_0$, $n\_p'_0$, $p\_p_0$,$p\_p'_0$)}
        else
            if $c_i = n\_p_0$ then
                if $c_i = n\_p'_1$ then done ← true. stop TraceUp.
                advanceNonZeroPath($r$,$c_i$, $e_r$, $n\_p_0$, $n\_p'_0$, $p\_p_0$, $p\_p'_0$, P0Flag)
            else if $c_i = n\_p_1$ then
                if $c_i = n\_p'_0$ then done ← true. stop TraceUp.
                advanceNonZeroPath($r$,$c_i$, $e_r$, $n\_p_1$, $n\_p'_1$, $p\_p_1$, $p\_p'_1$, P1Flag)
            else if $p_1 = null$ then
                if $c_i = n\_p'_0$ then done ← true. stop TraceUp.
                if $c_i$ not in $T$ then
                    if $p\_p_1 \neq null$ then fix $p\_p_1$ to $c_i$
                    $p\_p_1 \leftarrow c_i$
                else
                    $p_1 \leftarrow c_i$
                    if $p\_p_1 \neq null$ then fix $p\_p_1$ to $c_i$
                    advanceNonZeroPath($r$,$c_i$, $e_r$, $n\_p_1$, $n\_p'_1$, $p\_p_1$, $p\_p'_1$, P1Flag)
            else $M$ is not realizable. Stop
    else if $M[r, c_i] = 0$ then
        if $n\_p'_0 = n\_p'_1$ AND $FlexEnd[n\_p'_0] \neq 0$ then $M$ is not realizable. Stop
        if $c_i = n\_p'_0$ then advanceZeroPath(r, $c_i$, L, $n\_p_0$, $n\_p'_0$, $p\_p_0$, $p\_p'_0$)
        if $c_i = n\_p'_1$ then advanceZeroPath(r, $c_i$, L, $n\_p_1$, $n\_p'_1$, $p\_p_1$, $p\_p'_1$)
    if $n\_p_0 \neq null$ AND $M[r, n\_p_0] = 0$ then $M$ is not realizable. Stop
    if $n\_p_1 \neq null$ AND $M[r, n\_p_1] = 0$ then $M$ is not realizable. Stop
    if $c_i \leq e_r$ then
        if $p_0 = null$ then done ← true. Stop trace Up
        done ← checkIfTraceUpDone($r$,$c_i$,$n\_p_0$, $n\_p_1$, $p\_p_0$, $p\_p_1$)
    $c_i \leftarrow c_i - 1$
```

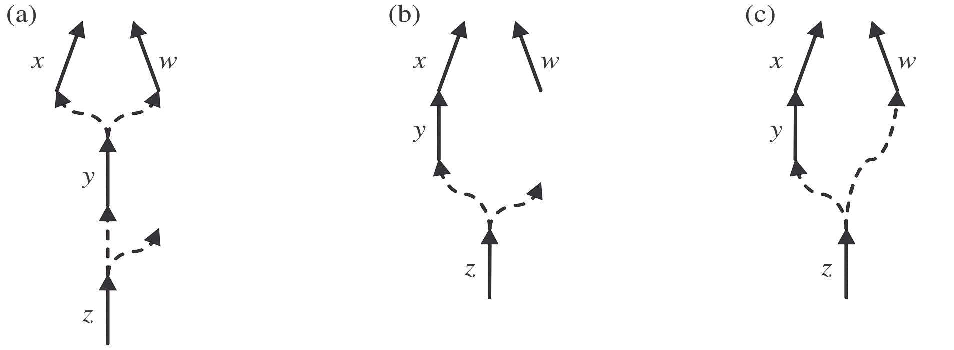


Figure 3.13: (a) Part of a FlexTree; (b) The FlexTree after fixing the site y to the site x in the TraceUp procedure; (c) The FlexTree after the TraceDown procedure

1. *orphan* sites: Consider the sites w, x, y and z as shown in Figure 3.13a. For the site y, f-parent0 is x and f-parent1 is w. For site z, f-parent0 is site x, and f-parent1 is *null*. Now, while processing some row r, if $M[r,x] = M[r,y] = 2$, $M[r,w] = M[r,z] = 0$, and if $p_0 < z$, the trace up procedure will fix the site y to the site x. However, simply doing so will make the site w not reachable from site z, as shown in Figure 3.13b. Clearly, this is not correct, as $P_M[w,z]$ is still ϕ and hence w must be reachable from z, as shown in Figure 3.13c. In this situation, the site z is an *orphan* site. The algorithm uses the array $L[]$ to handle these situations. During the TraceUp procedure, $L[y]$ is set to w. During the TraceDown procedure, if the f-parent1 of a flexible site z is *null*, then f-parent1 is set to $L[\text{fp0}(z)]$, where fp0(z) is f-parent0 of column z. Hence the TraceDown step makes sure that the site w is reachable from the site z.

2. Dealing with partitions - Fixing a partition: When a flexible site that is involved in a partition needs to be fixed, all the other sites involved in the partition also get effected.

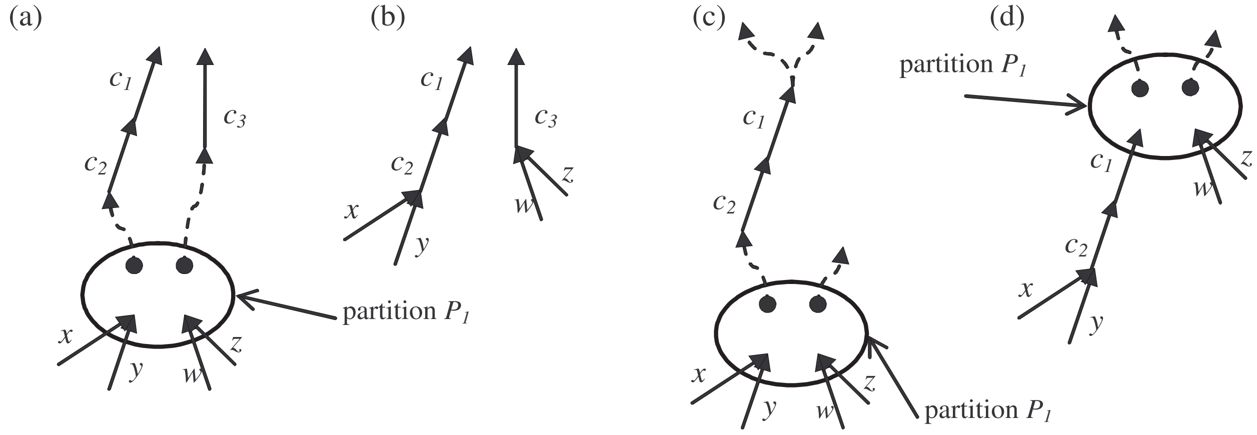


Figure 3.14: (a) A part of the FlexTree with a partition; (b) The FlexTree in *(a)* after fixing the site y; (c)Another FlexTree; (d) The FlexTree in *(c)* after fixing the site y to c_2

For example, consider the situation in Figure 3.14a. In some row r, if it is discovered that $P_M[c_2, y] = 0$, then the site c_2 must become the parent of site y. However, $P_M[y, i] = 1$ for every site i on the opposite side of the partition. Also because the sites y and i are in a partition, we know that all three sites c_2, y and i were '2' in some row before r. Hence Theorem 3.1 can be applied on the columns c_2, y, and i, and we can infer that $P_M[c_2, i] = 1$. The same logic applies to columns c_3, c_2 and i, and we can infer that $P_M[c_3, i] = 0$. Similarly, for every site i on the same side of the partition as y, there will be at least one site j on the other side of the partition so that the Theorem 3.1 can be applied on the columns c_2, i and j to infer that $P_M[c_2, j] = 0$. Hence, when we fix y to c_2, all the sites on the same side of the partition as y also get fixed to c_2, and all the sites on the other side of the partition get fixed to c_3. The impact of fixing y to c_2 is shown in figure 3.14b.

3. Dealing with partitions - Fixing one side of a partition: Consider the scenario shown in Figure 3.14c. In some row r, if it is discovered that $P_M[c_2, y] = 0$, the column y needs to get fixed to column c_2. As explained before, this also means that all the sites on the same side of the partition must get fixed to c_2. However, as f-parent(1) of partition $P1$ is *null*, the sites on the other side of the partition do not get fixed to any site. However, if c_1 is the FlexEnd of c_2, $P_M[c_1, i] = 1$ for every site i on the other side of the partition P_1. Hence the partition P_1 must now involve c_1. The overall effect is shown in Figure 3.14d. The f-parents of site c_1 now become the f-parents of partition P_1.

Lemma 3.6 *After the TraceUp step, the phase relationships between every pair of columns i and j such that $(i, j) \leq p_0$, $i \neq j$, $M[r, i] \neq 0$, $M[r, j] \neq 0$ are correctly represented in the FlexTree.*

Proof The Trace Up procedure forces every non-zero column between e_r and p_0 either in-phase with p_0 or in-phase with p_1 due to Theorem 3.6 and Lemma 3.4. Hence all the columns between e_r and p_0 that are forced in-phase with p_0 are forced in-phase with each other. The same is true for the columns between e_r and p_0 that are forced in-phase with p_1. Hence the pair-wise relationships between any pair of non-zero columns between e_r and p_0 are correctly represented in the FlexTree.

Since the partial genotype vectors $M[r-1, 1...e_r-1]$ and $M[r, 1...e_r-1]$ are both identical, no new pair-wise relationships are directly forced between any pair of such non-zero columns i and j. Therefore, any new pair-wise relationships between i and j must be indirectly inferred through a third column $x \geq e_r$. It is clear from Lemmas 3.4 and 3.5 that these relationships are correctly interpreted in the TraceUp procedure. Hence, all the pairwise relationships between any pair of non-zero columns with index less than or equal to p_0 are correctly represented in the FlexTree by the end of the TraceUp step. $\diamondsuit$

3.4.5 Trace Down

Trace down procedure mainly does four things: (1) Update the FlexEnd of every site (2) Correct *orphan* sites - the flexible edges that have only one path to the root, with the alternate path not defined (3) Update $h[]$ and $k[]$ arrays (4) Add the non-zero columns with index greater than p_0 to the FlexTree. The trace down procedure is simple and very straight forward. At each flexible site i at which f-parent1[i] is not defined, f-parent1[i] is set to L[f-parent0[i]]. At each fixed site, the FlexEnd of the parent is copied onto itself. Also, at any fixed site i, if $L[i]$ is not defined, $L[i]$ is set to L[parent[i]]. A high level description of the TraceDown procedure is shown in figure 3.15.

Lemma 3.7 *For any flexible site i with f-parent1[i] not defined, L[f-parent0[i]] must be the f-parent1 of i.*

Proof In the TraceUp procedure, when any flexible column j is about to be fixed to a column p, $L[j]$ is set to the flexible parent of j that is not equal to p. If both the flexible parents of j were defined before j gets fixed, then j must get fixed to one of them. Therefore,

Figure 3.15: The TraceDown() procedure

```
inputs: The column table, the partition table of $T$, $r$, $L$,$h$, $k$
Result: updates $h$ and $k$, adds new non-zero sites beyond $p_0$ to $T$, adds f-parent1 to some sites if
        necessary
for $c_i \leftarrow 1$ to $m$ do
    if $c_i$ is a fixed site then
        if $L[i] = null$ then $L[i] \leftarrow L[\text{parent}[i]]$
        FlexEnd[$i$] $\leftarrow$ FlexEnd[parent[$i$]]
        if $M[r, c_i] = 1$ then $h[c_i] \leftarrow c_i$, $k[c_i] \leftarrow c_i$
        else if $M[r, c_i] = 2$ then
            $h[c_i] \leftarrow c_i$
            if $h[c_i - 1] = parent[c_i]$ then $k[c_i] \leftarrow k[c_i - 1]$else $k[c_i] \leftarrow h[c_i - 1]$
        else $h[c_i] \leftarrow h[c_i - 1]$,$k[c_i] \leftarrow k[c_i - 1]$
    else if $c_i$ is not in $T$ then
        if $M[r, c_i] = 1$ then declare $M$ not realizable
        else if $M[r, c_i] = 2$ then
            if $h[c_i - 1] = k[c_i - 1]$ then
                fix $c_i$ to $h[c_i - 1]$
                $h[c_i] \leftarrow c_i$, $k[c_i] \leftarrow k[c_i - 1]$
            else
                set_fp0($c_i$, $h[c_i - 1]$),set_fp1($c_i$, $h[c_i - 1]$)
                $h[c_i] \leftarrow c_i$, $k[c_i] \leftarrow c_i$
        else $h[c_i] \leftarrow h[c_i - 1]$,$k[c_i] \leftarrow k[c_i - 1]$
    else if $fp1(c_i) = null$ then
        if $L[fp0(c_i)] \neq null$ then set_fp1($c_i$,$L$[fp0($c_i$)])
        if $M[r, c_i] = 2$ then
            $h[c_i] \leftarrow c_i$; if $h[c_i - 1] \neq fp0[c_i]$ then $k[c_i] = h[c_i - 1]$else $k[c_i] \leftarrow$ null
        else $h[c_i] \leftarrow c_i$, $k[c_i] \leftarrow k[c_i - 1]$
    else
        if $M[r, c_i] = 2$ then $h[c_i] \leftarrow c_i$ else $h[c_i] \leftarrow h[c_i - 1]$
        $k[c_i]$ = null
```

the other flexible parent will no longer be connected to j through a glue edge, and hence $L[j]$ is set to be equal to this flexible parent. When only f-parent0 of j is defined, j may or may not get fixed to f-parent0[j]. When j gets fixed to f-parent0[j], there is no column that is rendered unreachable from j. Hence $L[j]$ need not be set. However, if j is getting fixed to some column other than f-parent0[j], f-parent0[j] will be rendered unreachable from j, and hence $L[j]$ is set to f-parent0[j].

The TraceDown step processes the row from left to right. Hence, the trace down step always visits the parents of fixed sites and flexible parents of flexible sites before it visits the sites themselves. At any fixed site j, if $L[j]$ is not defined, $L[j]$ is set to L[parent[j]]. Now, let us consider the flexible column i with only f-parent0[i] defined. If L[f-parent0[i]] is defined, it means that L[f-parent0[i]] was reachable from i before processing row r. Therefore, L[f-parent0[i]] must be the f-parent1 of i after processing row r, in order to leave the phase relation ships between i and L[f-parent0[i]] unaltered. ♢

3.4.6 Correctness

Theorem 3.7 *Assuming the FlexTree correctly represents the pairwise relationships induced by the first $r-1$ rows before processing the row r, the FlexTree correctly represents the pairwise relationships induced by the first r rows after processing the row r.*

Proof There are many possible scenarios. we will consider each one of them.

Case 1: $M[r-1,e_r] = 1$ and $M[r,e_r] = 0$. Let $j < e_r$ be the column with the highest index such that $M[r,j] \neq 0$. Since $M[r-1,e_r] = 1$, there are no '2's in $M[r-1, 1...e_r - 1]$. Since $M[r-1, 1...e_r-1]$ and $M[r, 1...e_r-1]$ are identical, there are no '2's in $M[r, 1...e_r-1]$. Therefore, $s_r = e_r - 1$, and $h[s_r] = k[s_r] = j$. Now, the matrix M is not realizable by a perfect phylogeny if there is any column $i \geq e_r$ that is already in the FlexTree. Therefore, all the non-zero elements in $M[r, e_r...m]$ must be '0' in every row before r. The non-zero columns in $M[r, e_r...m]$ describe a new sub tree rooted in column j. The situation is identical to that of processing the first row. The ProcessNewRow procedure is called with the parameters (r, e_r), and represents all the new pairwise relationships introduced by row r correctly, based on the same reasoning as in Lemma 3.2.

Case 2: $M[r,e_r] = 2$ and p_0 is not defined. Since p_0 is not defined, none of the non-zero columns in $M[r, e_r...m]$ are already in the FlexTree. The TraceDown procedure behaves exactly like the ProcessNewRow Procedure when $M[r, 1...e_r - 1]$ is not split. When $M[r, 1...e_r - 1]$ is split, i.e. when $e_r - 1 = s_r$, there will be two possibilities - (a) $h[s_r] = k[s_r]$, which implies that $M[r, h[s_r]] = 1$. The situation is the same as in Case 1 above, and the TraceDown procedure behaves exactly like the ProcessNewRow Procedure. (b) $h[s_r] \neq k[s_r]$, which implies that $M[r, h[s_r]] = 2$. $h[s_r]$ will be the f-parent0 of e_r and $k[s_r]$ will be f-parent1 of e_r. For every non-zero column after e_r, the TraceDown procedure behaves exactly like the ProcessNewRow procedure, and hence represents the phase relationships correctly.

Case 3: p_0 is defined, and every non-zero column in row r between $h[e_r - 1]$ and p_0 is reachable from p_0. This implies that p_1 is not defined until $h[e_r - 1]$ is reached. Every nonzero column between e_r and p_0 must now be forced in-phase with p_0, according to Theorem 3.6. Let r_0 be the row with the highest index such that $r_0 < r$ and $M[r_0, e_r] \neq 0$. Now, since $M[r_0, 1...e_r]$ is lexicographically smaller than $M[r, 1...e_r]$, there must be at least one column $j \leq e_r$ such that either $M[r_0, j] = 1$ and $M[r, j] = 2$ or $M[r_0, j] = 0$ and $M[r, j] = 2$. Using the same argument as in Theorem 3.6, we can show thate_r must now be forced in-phase with $h[e_r - 1]$. Since $P_M[h[e_r - 1], e_r] = 0$, the phase relationship of e_r with any column with index less than $h[e_r - 1]$ can be deduced from Theorem 3.4. Hence, all the phase relationships are correctly represented, and the TraceUp procedure does not have to reach beyond $h[e_r - 1]$. In the TraceDown step after p_0, the every new non-zero column is dealt with as in the ProcessNewRow procedure.

Case 4: Both p_0 and p_1 are defined. The TraceUp procedure stops when one of the following conditions are satisfied:

- *A non-zero column j between $h[s_r] + 1$ and $h[e_r - 1]$ that is forced out-of-phase with both p_0 and p_1 is reached.* The matrix is unrealizable, since Theorem 3.1 is violated on the columns j, p_0 and p_1.

- *A non-zero column j between $h[s_r] + 1$ and $h[e_r - 1]$ that is neither forced in-phase nor forced out-of-phase with both p_0 and p_1 is reached.* The latest column in the path to p_0 (p_p_0) and latest column in the path to p_1 (p_p_1) must be on the opposite sides of a partition. Hence, a new partition is introduced, for which f-parent0 is j and f-parent1 is not defined.

- *Every non-zero column until* $h[s_r] + 1$ *is forced in-phase with* p_0 *or* p_1 *and both* $h[s_r]$ *and* $k[s_r]$ *are neither forced in-phase nor forced out-of-phase with* p_0 *and* p_1. Similar to the situation above. A partition with flexible parents $h[s_r]$ and $k[s_r]$ is introduced between p_p_0 and p_p_1.

- *Every non-zero column until* $h[s_r] + 1$ *is forced in-phase with* p_0 *or* p_1*, and at least one of* $h[s_r]$ *and* $k[s_r]$ *is forced in-phase or out-of-phase with* p_0 *or* p_1. If $h[s_r]$ is forced in-phase with p_p_0 or $k[s_r]$ is forced out-of-phase with p_p_0, p_p_1 must get fixed to $k[s_r]$. The matrix is not realizable if p_0 and p_1 are both forced in-phase with $h[s_r]$ and $h[s_r] \neq k[s_r]$. The matrix is also not realizable when both p_0 and p_1 are either forced out-of-phase with either $h[s_r]$ or $k[s_r]$.

In the TraceDown step after p_0, every new non-zero column is dealt with as in the ProcessNewRow procedure. ♢

If the matrix is not realizable by a perfect phylogeny, there can be no FlexTree that describes all the phase relationships imposed by all the rows in the matrix, and hence the algorithm fails to build a FlexTree and reports the same.

3.4.7 Obtaining a PPH Tree from the FlexTree

The total number of PPH trees represented by the FlexTree is given by the following expression:

$$\gamma = 2^{([\text{no. of partitions}] + [\text{no. of flexible sites not in a partition}])} \tag{3.2}$$

Any of these γ solutions can be computed in $O(m)$ time from the FlexTree. The high level description of the BuildPPHTree procedure is shown in Figure 3.16. Please refer to the Appendix for the explanation of the functions *fp0()* and *fp1()*. The procedure fixes each flexible site, starting from the site with the lowest index and processing the sites in M from left to right. There will be only two possibilities at any flexible site, as all the sites with higher indices are already fixed. Different criteria can be applied to choose between the two choices, in order to obtain the *deepest* or the *broadest* tree.

Figure 3.16: The BuildPPHTree() procedure

```
inputs: The column table and the partition table of $T$
Result: A PPH Tree described by $T$
$L[i] \leftarrow$ null $\forall i$, $1 \leq i \leq m$
for $i \leftarrow 1$ to $m$ do
    if $i$ both f-parents are defined then
        arbitrarily set L[i] to one of the f-parents
        fix $i$ to the other f-parent
    else if $fp0(i) \neq$ null but $fp1(i) = (null)$ then
        arbitrarily set L[i] to either $fp0(i)$ or $L[fp0(i)]$
        fix $i$ to the column out of ($fp0(i)$,$L[fp0(i)]$) that is not equal to L[i]
    else $L[i] \leftarrow L[\text{parent}[i]]$
```

3.5 Complexity

3.5.1 Pre-processing

It takes $O(nm)$ time to compute the column sums. Once the column sums are computed, it takes $m\log(m)$ time to sort the columns (using quick sort) according to the column sums. The lexicographic ordering of the rows takes $O(nm)$ time and space, using radix sort. The total time required for the preprocessing step is $O(nm)$.

3.5.2 Scan Forward

The ScanForward step is straight forward, as shown in Figure 3.17. Takes $O(m)$ time.

Figure 3.17: ScanForward procedure - finds e_r and s_r

```
**inputs** : $M$, $k[]$, $r$
**outputs**: $e_r$, $s_r$
$i \longleftarrow 1$,$e_r \longleftarrow 1$, $s_r = 0$
**while** $M[r, i] = M[r-1, i]$ **do**
    $e_r \leftarrow i + 1$
    **if** $k[i] \neq null$ **then**
        $s_r \leftarrow i$
    $i \leftarrow i + 1$
```

3.5.3 Trace Up

As long as partitions are not involved, the Trace Up procedure takes constant time at each site. However, the Trace Up procedure might spend up to $O(m)$ time at sites that are involved in partitions. Introducing a new partition is always a constant-time operation, as a new partition always involves just two sites. Adding a single partition to an existing partition is also a constant time operation. Merging two partitions into one, or fixing one side or both sides of the partition, takes time in the order of the size of the partition(s) involved. However, the total amortized cost for all the mergers and fixings while processing any single row is $O(m)$. This is because of the fact that the algorithm has to deal with at most two 'independent' partitions at any time, one involving p_p_0, and the other involving p_p_1. The first time the site p_p_0' is encountered, the algorithm introduces a partition $P0$ between the FlexEnd of p_p_0' and the p_p_0. Another partition reachable from p_0 will not be encountered until the TraceUp procedure reaches beyond the current FlexEnds of both p_p_0' and p_p_0. After this point, whenever the TraceUp reaches the next site y that is reachable from p_0, the algorithm does the following:

- Depending on whether y has to be forced in-phase with p_p_0 or p_p_0', removes all the sites from the corresponding side of the partition and fixes them to site y.
- Adds the FlexEnd of site y to the appropriate (the empty) side of partition $P0$. If the FlexEnd of y is already in a partition, removes all the sites from that partition and adds them to the appropriate side of $P0$.

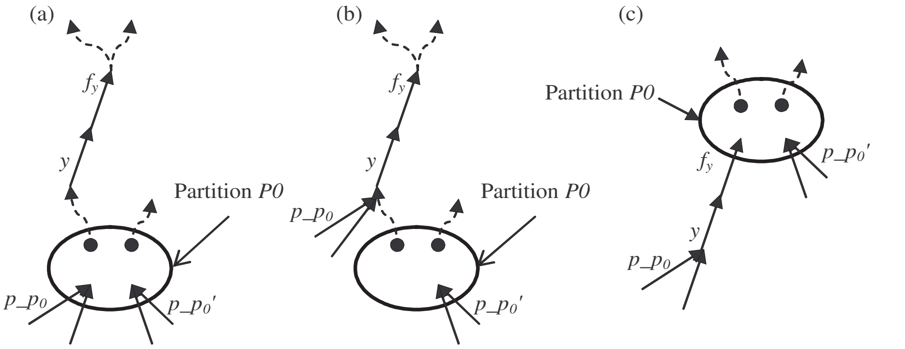


Figure 3.18: Illustration of how the TraceUp procedure deals with partitions (a) The partition $P0$ just before TraceUp reaches the site y; (b) The sites on the side of the partition $P0$ that should be fixed to y are removed from $P0$ and fixed to y; (c) f_y, the FlexEnd of y, is added to $P0$ and the f-parents of $P0$ are updated to those of f_y

- Updates the f-parents of the partition $P0$ to those of FlexEnd[y] just before FlexEnd[y] was inserted into $P0$.

Clearly, nothing needs to be done for the sites that are on the opposite side of the partition that was fixed to y. The above steps are shown in Figure 3.18. In some cases, both sides of the $P0$ get fixed, and $P0$ will be completely empty. Therefore, as the TraceUp procedure proceeds, sites enter (become part of $P0$) and exit $P0$ (get fixed). A constant amount of time needs to be spent on every site that enters or exits $P0$. Once a site gets fixed, it exits $P0$, it has no way of re-entering $P0$. As at most $O(m)$ sites can enter or exit $P0$, the total amortized cost is $O(m)$.

Similar will be the case with the partition $P1$ that involves the FlexEnd of p_p_1. When the TraceUp procedure terminates, either one or both sides of $P0$ and $P1$ get fixed, or $P0$ and $P1$ merge into a single partition. In any case, the time required will be $O(m)$, as the combined size of $P0$ and $P1$ is at most m.

3.5.4 Trace Down

The trace down is also straight forward. It involves a constant number of operations at each site. Therefore, takes $O(m)$ time for each row.

3.6 Results

A opph algorithm has been implemented in C++. The results indicate that the performance is as expected, indicating that there are no hidden constraints. Table 3.4 shows how the opph algorithm performs in comparison to algorithms gpph[Gus02] and dpph[BGL02]. The times for opph are averages over 1000 test cases. The times for gpph and dpph are averages over five cases. It is clear that the opph algorithm outperforms both gpph and dpph algorithms. The tests were carried on simulated data. A random PPH tree was generated, and the genotypes were obtained by selecting two random haplotypes from the tree and combining them together. The binaries for the implementation are available for download from http://www.cs.ucf.edu/∼rvijaya/opph/.

Test case ($n \times m$)	gpph	dpph	opph
50×50	0.11	0.01	0.007
100×100	0.71	0.07	0.017
200×200	4.49	0.53	0.06
500×500	83.2	7.99	0.28
1000×1000	662	66.5	0.43
1000×2000	did not complete	302.78	0.97

Table 3.4: Performance results - all times are in seconds on a P4 3GHz machine

.

3.7 Discussion

The FlexTree data structure presented in this chapter is a simple, intuitive data structure for representing all the PPH solutions for a given genotype matrix. The applications of this data structure extend beyond opph algorithm and the PPH problem.

3.7.1 MPPH problem

The Minimum Perfect Phylogeny Problem (MPPH) problem is to find the PPH solution that uses the minimum number of distinct haplotypes. The problem was proven to be NP-hard in a recent paper [BGH04]. The FlexTree data structure helps in defining a non-trivial lower bound on the number of distinct haplotypes in the MPPH solution. If row r in the matrix is split, the two *ending* sites are defined for the row. In any PPH tree, the two haplotypes must end in the sites given by h_r^m and k_r^m. i.e., in any PPH tree the nodes represented by the two haplotypes for the row r are well-defined. Even in case of a row that is not split, h_r^m is defined. Therefore, the cardinality of the set of distinct h_r^m and k_r^m values for all the rows in the matrix A^c gives a non-trivial lower bound for the problem. If every row in the matrix is split, then this quantity will be the exact number of haplotypes in the PPH problem. In general, there will be very few PPH solutions for any given genotype matrix, and the FlexTree data structure might be used to develop an efficient, practical solution for the MPPH problem.

3.7.2 Selecting a PPH tree

If the input matrix has multiple PPH solutions, the FlexTree helps in finding the most *desirable* solution under certain criteria. Intuitively, the *deepest* and *broadest* possible PPH trees can be built by making minor modifications to the BuildPPHTree procedure in section 3.4.7. In addition, the PPH solution that includes or excludes a given haplotype can be easily obtained by first making simple modifications to the FlexTree in order to force the inclusion or exclusion of a given haplotype.

3.8 Pseudocode for Some Procedures

The following figures provide a high-level description of some of the fundamental procedures used by the opph algorithm.

fp0(i): Returns the f-parent0 of column i. If the site i is not involved in a partition, returns f-parent0 from the column table. If the site i is in a partition P, returns f-parent0 of the partition in the partition table.

fp1(i): Similar to *fp0()*. Returns f-parent1 of the site i.

set_fp0(i, c): Sets the f-parent0 of site i to c. If the column i is in a partition P, sets f-parent0 of the partition P in the partition table.

set_fp1(i, c): Similar to *set_fp0()*. Sets the f-parent1 of site i to c.

Fix site i to c: Assigns site c to be the parent of site i, while modifying the other columns if necessary, as described in section 3.4.4.

checkIfTRaceUpDone(): The routine checks if TraceUp procedure can stop. Performs the necessary operations if the TraceUp procedure can stop (like introducing a partition between the FlexEnd of p_0 and the FlexEnd of p_1, if necessary).

Figure 3.19: The fixNonZeroPath() procedure

```
inputs: $r$, $c_i$, $L[]$, $n\_p$, $n\_p'$, $p\_p$, $p\_p'$, PFlag
Result: fixes $p\_p$ to $n\_p$ or to $c_i$
if $p\_p \neq$ null then
    if $n\_p \neq$ null then fix $p\_p$ to $c_i$
    else if ($p\_p$) is not fixed then
        if PFlag is true then fix $p\_p$ to $n\_p$ and set $L[p\_p]$ to $n\_p'$
        else fix $p\_p$ to $c_i$
```

Figure 3.20: The advanceNonZeroPath() procedure

```
inputs: The column table, the partition table of $T$, $r$, $c_i$, $n\_p$, $n\_p'$, $p\_p$, $p\_p'$, $L$
Result: fixes $p\_p$, updates $n\_p$,$p\_p$ and/or $n\_p'$ if necessary
if parent[$c_i$] $\neq$ null then
    $n\_p \leftarrow$ parent[$c_i$]
    if $M[r, n\_p] = 0$ then  declare that $M$ is not realizable, Stop
else
    if $c_i < n\_p'$ AND P0Flag is false then
        if fp1($c_i$) = null then
            if $M[r, fp0(c_i)] \neq 0$ then $n\_p \leftarrow$ fp0($c_i$)
            else
                $n\_p' =$ fp0($c_i$)
                $L[c_i] \leftarrow n\_p'$
                if partition[$c_i$] =null then create a partition between $c_i$ and FlexEnd[$n\_p'$]
                else fix the group in partition[$c_i$] that does not include $c_i$ to $n\_p'$
        else
            if $M[r, fp0(c_i)] \neq 0$ then
                $n\_p \leftarrow$ fp0($c_i$),$n\_p' \leftarrow$ fp1($c_i$)
                if $M[r, n\_p'] \neq 0$ then P0Flag $\leftarrow$ true
                else  $L[c_i] \leftarrow n\_p'$
            else $n\_p' \leftarrow$ fp0($c_i$),$n\_p \leftarrow$ fp1($c_i$), $L[c_i] \leftarrow n\_p'$
            if $M[r, n\_p'] = 0$ then fix $c_i$ to $n\_p$
$p\_p \leftarrow c_i$
```

CHAPTER 4

CONSTRUCTING NEAR-PERFECT PHYLOGENIES

4.1 Imperfect Phylogenies

Biological data rarely, if ever, conforms to perfect phylogeny. Deviations from perfect phylogeny are common due to repeated mutations and recombinations. With repeated mutations, the phylogeny is a tree with multiple edges labeled with the same character. With recombinations, the phylogeny will no longer be a tree, but a network with recombination cycles. Ability to construct imperfect phylogenies is critical for applying perfect-phylogeny-based haplotype inference methods on real-life genotype data.

In the human genome, the deviations from perfect phylogeny are expected to be small within a 'block' of the genome. When the deviations from perfect phylogeny are small, and are due to repeated/back mutations, the phylogenies are referred to as *near-perfect* phylogenies. This chapter deals with algorithms for constructing near-perfect phylogenies on both haplotype and genotype data.

The previous chapter dealt with constructing *rooted* phylogenies. In case of a perfect phylogeny, it is always possible to transform the input data so that the root must be an all-zero vector. There is no such known transformation in case of an imperfect phylogeny. Hence, we deal with *unrooted* phylogenies in this chapter. In case of an unrooted phylogeny, there is no distinction between a repeated mutation and a back mutation. The term *homoplasy event* is used to refer to a repeated/back mutation.

4.1.1 Previous work on constructing near-perfect phylogenies

The problem of constructing near-perfect phylogenies with multiple homoplasy events has been tackled before [FL03]. The complexity of their algorithm for constructing near perfect phylogenies on a set of n haploid taxa is given by $O(nm^q 2^{q^2 r^2})$, where r is maximum number of alleles in any site, and q is the number of repeated/back mutations. Here, we are only concerned with bi-allelic SNP data, and hence $r = 2$. Even in case of bi-allelic data, the above algorithm is clearly impractical for values of q as small as four. Recently, Sridhar et al. [SDB05] proposed a more practical algorithm for binary data with complexity $(q + p)^{O(q)} nm + O(nm^2)$ where p is the number of characters that share four gametes with some other character.

This chapter deals with fixed-parameter versions of the near-perfect phylogeny problem on both haplotype and genotype data and presents polynomial time algorithms for these problems. Song et al. [SWG05] have introduced a restricted version of the near-perfect phylogeny haplotyping problem that allows a single homoplasy event. This version of the problem is called the H1 Near-Perfect Phylogeny Haplotyping (H1-NPPH) problem. The notation 'H1' indicates that there is a single homoplasy event in the phylogeny. Song et al. [SWG05] first identify the column with the homoplasy event, construct a perfect phylogeny T' for the remaining columns, and then convert T' into an H1-NPP T that includes the column with the homoplasy event. In converting T' into T, the procedure followed in [SWG05] is to remove pairs of edges from T' and carry out certain tests on the disconnected subtrees produced as a result of removing the pair of edges from T'. The overall complexity of the algorithm is $O(n^4)$.

The fundamental approach in this chapter is similar to that presented in [SWG05]. However, removing pairs of vertices from T' leads to a faster algorithm than removing pairs of edges from T'. This observation results in a faster $O(m^2(n+m))$ algorithm that can be easily extended to handle multiple homoplasy events. The framework for constructing near-perfect phylogenies presented in the rest of the chapter is based on this observation. This frame work can be generalized to extend to constructing near-perfect phylogenies(NPPs) that involve multiple homoplasy events, both for haplotype and genotype data.

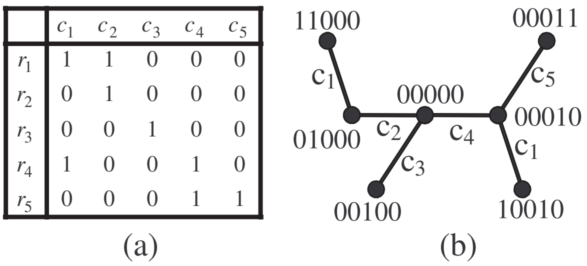
An $\text{H}(1, q)$ NPP is a near-perfect phylogeny involving q homoplasy events in a single site. Similarly, a $\text{H}(p, q)$-NPP is a near perfect phylogeny in which at most p sites have homoplasy events, with at most q homoplasy events in each site. Under this notation, a near-perfect phylogeny with a single homoplasy event is denoted as the $\text{H}(1, 1)$-NPP.

Section 4.2, presents a polynomial-time algorithms for constructing near-perfect phylogenies for haplotype data. In Section 4.3, these algorithms are extended to genotype data.

4.2 Constructing Near-Perfect Phylogenies from Haplotype data

In the following, we present polynomial-time algorithms for restricted versions of Near-Perfect Phylogeny (NPP) problem. In all the problems that we describe in this section, the input is an $n \times m$ matrix M over the alphabet $\{0, 1\}$, where the columns $c_1, c_2, ..., c_m$ indicate sites and the rows $r_1, r_2, ..., r_n$ indicate samples. Given that the matrix M does not admit a perfect phylogeny, we want to construct a near-perfect phylogeny for M that is the closest to a perfect phylogeny.

We define the following terms. An ordered pair of values (a, b), $a \in \{0, 1\}$, $b \in \{0, 1\}$, is said to be *induced* by a pair of ordered columns (i, j) if there is a row r in M such that $M[r, i] = a$ and $M[r, j] = b$. The set of ordered pairs induced by a pair of columns (i, j) is denoted by $I(i, j)$. According to the well-established four-gamete test [HK85], the matrix M does not admit a perfect phylogeny if $|I(i, j)| = 4$ for any pair of columns (i, j). We say that two columns i and j *conflict* with each other if $|I(i, j)| = 4$. A *conflict graph* $G_c = (V, E)$ is a graph in which each vertex $v_i \in V$ corresponds to a column c_i in M. An edge (v_i, v_j) is in E if the sites c_i and c_j conflict with each other.



	c_1	c_2	c_3	c_4	c_5
r_1	1	1	0	0	0
r_2	0	1	0	0	0
r_3	0	0	1	0	0
r_4	1	0	0	1	0
r_5	0	0	0	1	1

Figure 4.1: (a) A haplotype matrix M; (b) A phylogeny T for M

The general definition of a phylogeny is that the phylogeny is a tree in which the leaves represent the input taxa. As we are constructing character-based phylogenies, we are only interested in the topology of the phylogeny. Therefore we use the term phylogeny to refer to an edge and vertex labeled tree T. Each edge in T is labeled by a site in M, and indicates a mutation in that site. An example of a phylogeny is shown in Figure 4.1. Each vertex in the phylogeny is labeled by a 0-1 vector of length m, and indicates the state of each site at the vertex. For any vertex v, we denote the vertex label of v as $\mathcal{L}(v)$. Since T is a phylogeny for M, for each row r in M, there must be a vertex v such that $\mathcal{L}(v) = M[r]$. This mapping of a row r to a vertex v is represented using the notation $\nu(r) = v$. Multiple rows in M might map to the same vertex in T, and some vertices in T might not represent any row in M. Notice that the phylogeny in Figure 4.1 is not a perfect phylogeny. There are two edges in T labeled with column c_1.

Removing a set of vertices S_c from any tree T divides T into a set of connected (trivial or non-trivial) components denoted by $T_{/S_c}$. Note that, since T is a tree, each connected component $T_i \in T_{/S_c}$ will also be a tree. For any connected component T_i of T, we define

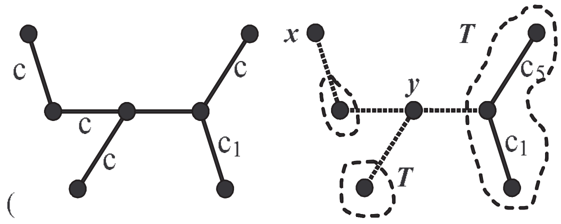


Figure 4.2: (a) The tree T before removing the vertices x and y; (b) The three connected components T_1, T_2 and T_3 after removing the vertices x and y

$R(T_i)$ as the set of rows of M that map to any vertex in T_i. A column c is said to be *non-polymorphic* in T_i if the column c has the same state in each row $r \in R(T_i)$. For example, refer to Figure 4.2a, which is the same phylogeny as in Figure 4.1. The three connected components produced by removing the vertices x and y in Figure 4.2a are shown in Figure 4.2b (in dotted regions). In the matrix M, the row r_2 maps to T_1, r_3 maps to T_2, and the set of rows $\{r_4, r_5\}$ map to T_3. All the columns are non-polymorphic in T_1 and T_2. However, columns c_5 and c_1 are *polymorphic* in T_3. Columns c_2, c_3 and c_4 are non-polymorphic in T_3.

4.2.1 The H1-NPP construction problem

In the following, we describe the conditions under which a given set of haplotypes admit an H1-NPP. There are efficient algorithms to determine if the matrix M admits a perfect phylogeny. When M does not admit a perfect phylogeny, the problem is to construct an

H1-NPP for the matrix M, or determine that M does not admit an H1-NPP. For simplicity, we call the H1-NPP construction problem as the H1-NPP problem in the rest of the chapter.

Let M be a matrix that does not admit a perfect phylogeny, but admits an H1-NPP. Let c_b be the column with the recurrent mutation. Let T be the H1-NPP for M. By definition, if an edge (u, v) is labeled by a site i, it implies that $\mathcal{L}(u)[i] = \overline{\mathcal{L}(v)[i]}$. Clearly, there will be two edges in T that are labeled with c_b. Let the two edges be (u, v) and (w, x), as shown in Figure 4.3. We call the path between the two vertices v and w as the *recurrent mutation path*, or *RMP*. Let S be the set of all sites, i.e., $S = \{c_1, c_2, ..., c_m\}$. Let S_{RMP} be the set of sites that label an edge in *RMP*. Let S_e be the set of sites other than c_b that are not in RMP. i.e., $S_e = S - \{S_{RMP} \bigcup \{c_b\}\}$.

Theorem 4.1 *Every site $c \in S_{RMP}$ conflicts with c_b, and every site $c \in S_e$ does not conflict with c_b.*

Proof Let $\mathcal{L}(u)[c_b] = a$. Clearly, $\mathcal{L}(v)[c_b] = \overline{a} = \mathcal{L}(w)[c_b]$ and $\mathcal{L}(x)[c_b] = a$. For any site $c \in S_{RMP}$, $\mathcal{L}(v)[c] = \overline{\mathcal{L}(w)[c]}$. The site c_1 connecting the vertices y and z in Figure 4.3 is such a site. Let $\mathcal{L}(y)[c_1] = b$, which implies that $\mathcal{L}(y)[c_1] = \overline{b}$. The phylogeny T can be divided into four subtrees T_1, T_2, T_3 and T_4 with respect to the sites c_b and c_1, as shown in Figure 4.3. The pair of sites (c_b, c_1) take the states (a, b), $(\overline{a}, b)$, $(\overline{a}, \overline{b})$ and $(a, \overline{b})$, in subtrees T_1, T_2, T_3 and T_4, respectively. Now, $R(T_1)$, $R(T_2)$, $R(T_3)$ and $R(T_4)$ are all non-empty. This is because the matrix M will admit a perfect phylogeny if $R(T_1)$ or $R(T_4)$ are empty, and

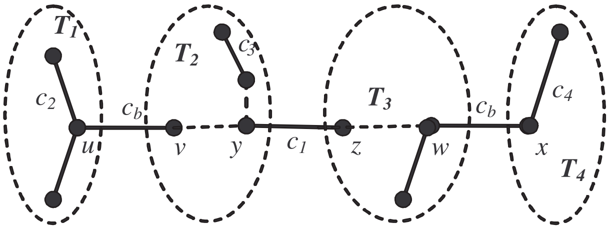


Figure 4.3: Illustration of Theorem 4.1

c_1 need not be in RMP if $R(T_2)$ or $R(T_3)$ are empty. Therefore, $|I(c_b, c_1)| = 4$, and hence c_b conflicts with c_1.

It can similarly be shown that every site $c \in S_e$ will not conflict with c_b. Sites c_2, c_3 and c_4 in Figure 4.3 are examples of such sites. $\diamondsuit$ As explained before, $T_{/\{u,v,w,x\}}$ is the set of connected components generated by removing vertices u, v, w and x from T. Removing the vertices u, v, w and x removes both the edges labeled with c_b from T. Therefore, no connected component in $T_{/\{u,v,w,x\}}$ will have an edge labeled with c_b. Therefore, the column c_b will be non-polymorphic within any connected component $T_i \in T_{/\{u,v,w,x\}}$.

We will now state and prove a theorem that gives the necessary and sufficient conditions for a haplotype matrix to admit a H1-NPP. Let M be a matrix such that M does not admit a perfect phylogeny, but the matrix M' produced by removing a column c_b from M admits a perfect phylogeny T'. Since the rows in M correspond one-to-one with rows in M', the rows in M can be mapped to vertices in T'. It will be helpful to visualize the matrix M as the matrix M' with a single column c_b appended as the rightmost column of M. We state the following theorem:

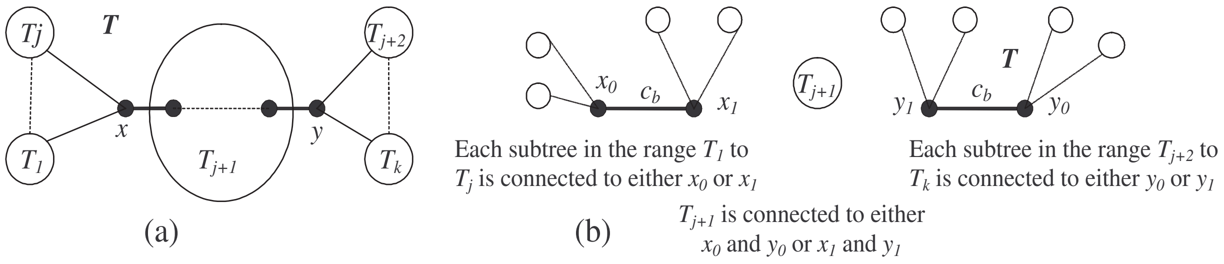

Tj
T
T1
x
Tj+1
y
Tj+2
Tk
(a)
x0
cb
x1
Tj+1
T
y1
cb
y0
Each subtree in the range T1 to Tj is connected to either x0 or x1
Each subtree in the range Tj+2 to Tk is connected to either y0 or y1
Tj+1 is connected to either x0 and y0 or x1 and y1
(b)

Figure 4.4: (a) The perfect phylogeny T', showing $\{T_1, ...T_k\}$, the connected components in $T'_{/\{x,y\}}$; (b) Constructing T from $T'_{/\{x,y\}}$

Theorem 4.2 *The matrix M admits an H1-NPP iff there are two vertices x and y in T' such that the site c_b is non-polymorphic in every connected component in $T'_{/\{x,y\}}$.*

Proof Let $T'_{/\{x,y\}} = \{T_1, T_2, ...T_k\}$, as shown in Figure 4.4a, where $k = d(x)+d(y)-1$, $d(x)$ is the degree of x and $d(y)$ is the degree of y in T'. We show that we can construct an H1-NPP T for M by expanding the vertices x and y into edges labeled with c_b. We start with an empty tree T. We replace x with two new vertices x_0, x_1, and y with two new vertices y_0 and y_1, and add two edges (x_0, x_1) and (y_0, y_1), both labeled with c_b. The two vertices x_0 and x_1 are labeled based on the label of the vertex x in T' as - $\mathcal{L}(x_0)[i] = \mathcal{L}(x_1)[i] = \mathcal{L}(x)[i]$ for every column $i \neq c_b$. This is equivalent to taking the matrix M' and associating the vertex label of x in T' to both the vertices x_0 and x_1. The site c_b is now associated with the edge (x_0, x_1) as follows: $\mathcal{L}(x_0)[c_b] = 0$, and $\mathcal{L}(x_1)[c_b] = 1$. The vertices y_0 and y_1 are similarly labeled based the label of the vertex y in T' in every site other than c_b. In site c_b, $\mathcal{L}(y_0)[c_b] = 0$ and $\mathcal{L}(y_1)[c_b] = 1$. With reference to Figure 4.4b, in each component T_i, $1 \leq i \leq j$, there will be a vertex v_i so that (x, v_i) is an edge in T'. Since T_i is non-polymorphic in c_b, we introduce an edge (x_0, v_i) or (x_1, v_i) in T, depending on whether $\mathcal{L}(v)[c_b] = 0$, or $\mathcal{L}(v)[c_b] = 1$, respectively. Similarly, each component from T_{j+2} to T_k are connected to either y_0 or y_1 by an edge, as shown in Figure 4.4b. If T_{j+1} is non-empty, there will be vertices v_1 and v_2 in T_{j+1} so that (x, v_1) and (y, v_2) are edges in T'. If $\mathcal{L}(v_1)[c_b] = 0$, we can introduce the edges (x_0, v_1) and (y_0, v_2) in T. If $\mathcal{L}(v_1)[c_b] = 1$, we can introduce the edges (x_1, v_1) and (y_1, v_2) in T. If T_{j+1} is empty (i.e., if x and y are adjacent in T'), we can arbitrarily introduce either the edge (x_0, y_0) or (x_1, y_1) in T. Therefore, all the edges in T' can be inserted back into T in addition to the two edges labeled with c_b. Every row in M can be mapped to vertex in T, and hence T is an H1-NPP for M. This proves that the existence of the two vertices x and y is a sufficient condition for the matrix M to admit an H1-NPP.

To prove that the existence of the two vertices x and y is a necessary condition, assume that a given matrix M admits an H1-NPP T. We prove that there must be two vertices x and y in T so that $T'_{/\{x,y\}}$ is non-polymorphic in c_b. Since T is an H1-NPP, there must be exactly two edges labeled with c_b in T. Remove the two edges, by collapsing the edges into vertices. Call these vertices x and y. Now obtain the set of trees $T'_{/\{x,y\}}$. Since c_b does not appear as an edge in any of the trees in $T'_{/\{x,y\}}$, c_b is non polymorphic in each component tree. Hence, the existence of the vertices x and y is a necessary condition. $\diamondsuit$

4.2.2 The H1-NPP Construction Algorithm

Theorem 4.1 and Theorem 4.2 allow us to determine if a given matrix M admits an H1-NPP and lead to an efficient algorithm to determine a H1-NPP solution for the given matrix M. The heart of the algorithm consists of determining the vertices x and y satisfying Theorem 4.2 and expanding the nodes into edges labeled with c_b. We have already observed the following properties of the conflict graph G_c:

- The conflict graph G_c for M must have a single non-trivial connected component and there must be at most one vertex with degree greater than one in the conflict graph. If there is any vertex with degree greater than one in G_c, c_b must be that column. If the conflict graph is a single edge connected by two sites, c_b must be one of the two sites.

- Let M' be the matrix produced by removing the column c_b from M. All the sites connected to c_b in the conflict graph must form a path P in the perfect phylogeny T' for the matrix M'.

- Let e_1 and e_2 be the two terminal vertices of the path P in T'. The site c_b should be non-polymorphic in each connected component $T_i \in T'_{/\{e_1,e_2\}}$.

These properties lead to an algorithm for the construction of an H1-IPP for M.

Algorithm Steps

1. Build the conflict graph G_c for M. If G_c has more than one non-trivial connected components, or if there is more than one vertex in G_c with degree greater than 1, M does not admit an H1-NPP. Otherwise proceed to Step 2.

2. Select the column c_b. c_b will be the column with degree greater than 1 in G_c. If the connected component in G_c is a single edge, arbitrarily pick any of the two vertices that form the edge.

3. Remove the column c_b from M, and construct a perfect phylogeny T' for the resulting matrix.

4. Construct the set of columns S_c that are adjacent to c_b in G_c. If M admits an H1-NPP, the columns in S_c must define a path P in T'. Obtain the two terminal ends x and y of this path. If S_c does not define a path in T', M does not admit an H1-NPP.

5. Check if every connected component in $T'_{/\{x,y\}}$ is non-polymorphic in c_b. If any connected component in $T'_{/\{x,y\}}$ is polymorphic in c_b, M does not admit a perfect phylogeny.

6. Expand the vertices x and y into the edges (x_0, x_1) and (y_0, y_1), both labeled with the column c_b. Build the phylogeny T as described in the proof of Theorem 4.2.

Figure 4.5 illustrates the algorithm. Figure 4.5a shows a matrix M with nine sites and ten rows. The conflict graph G_c for M is shown in Figure 4.5b. From the conflict graph, it is clear that removing column c_3 will result in a perfect phylogeny. The perfect phylogeny T' after removing c_3 is shown in Figure 4.5c. The site c_3 conflicts with sites c_5 and c_7, Hence

(a) M

$$\begin{array}{c|ccccccccc}
 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 \\
r_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
r_2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
r_3 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
r_4 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
r_5 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
r_6 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
r_7 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
r_8 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
r_9 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
r_{10} & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1
\end{array}$$

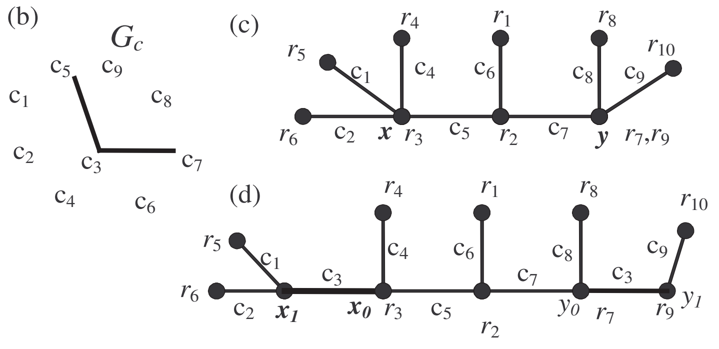


Figure 4.5: (a) A matrix M (b) Conflict graph for M (c) Perfect phylogeny T' after removing c_3. (d) The H1-NPP T for M.

the path defined by the edges c_5 and c_7 should be the path between the two mutations in site c_3. Hence the vertices x and y in Figure 4.5c must be replaced by the edges (x_0, x_1) and (y_0, y_1) in Figure 4.5d. In Figure 4.5c, the edges labeled with c_1, c_2, c_4 and c_5 are incident in x. In Figure 4.5d, the edges c_1 and c_2 are incident on x_1 and c_4 and c_5 are incident on x_0, because of the state of c_3 in r_5, r_6, r_4 and r_2, respectively. The row r_3 now maps to x_0, since $M[r_3, c_3] = 0$. Similarly the edges out of y in T' are distributed between the vertices y_0 and y_1 in T.

Complexity Analysis

Building the conflict graph G_c takes $O(nm^2)$ time. Finding the connected components in G takes $O(m)$ time using depth-first search. Constructing the perfect phylogeny T' takes $O(nm)$ time, using the opph [VM05, VM06] algorithm. The mapping $\nu(r)$ of each row in M to a vertex in T' can be done in using $O(n)$ space and $O(nm)$ time. Finding the two vertices x and y takes $O(nm)$ time. Building and checking each component in $T'_{/\{x,y\}}$ for being non-polymorphic in c_b takes $O(m)$ time. The overall complexity of the algorithm is thus entirely dominated by the construction of the conflict graph G_c and hence is $O(nm^2)$.

4.2.3 Multiple Homoplasy Events in a Single Site

An extension of the H1-NPP problem is the case when multiple homoplasy events within the same site are allowed. This situation occurs quite frequently with true haplotype data. For example, the site 16519 in human mtDNA is expected to have mutated multiple times. We call this problem the H$(1,q)$-NPP problem. Formally, the H$(1,q)$-NPP problem is to construct a phylogeny for the input taxa in which a single site has mutated at most $q+1$ times, where q is an integer greater than 0.

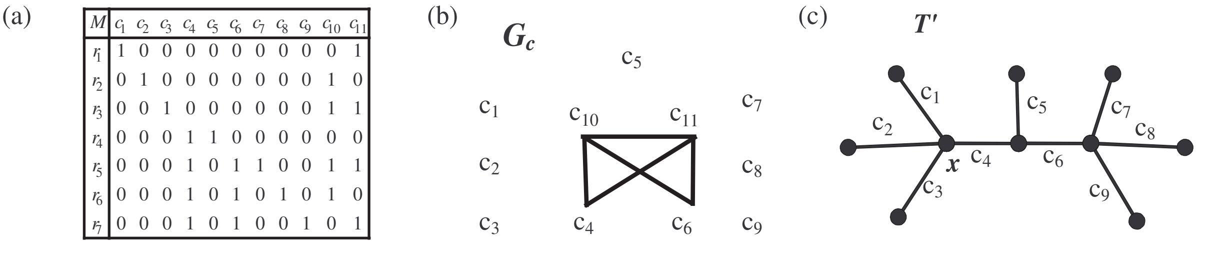
The solution to the H$(1,q)$-NPP problem is an obvious extension of the solution to the H1-NPP problem. As before, the conflict graph G_c for M must have a single connected component, and there should be a single site c_b with degree greater than 1 within this connected component. We can build a perfect phylogeny T' for the matrix M' obtained by removing the column c_b from M. Now, we need to find if there are $q+1$(or fewer) vertices in T' so that expanding each one of these $q+1$ vertices into an edge labeled with c_b will result in a phylogeny T for M. This can be done by testing all possible combinations of $q+1$ vertices in T' to check if they can lead to an H$(1,q)$-NPP solution. A set $\mathcal{Q}$ of $q+1$ vertices admits an H$(1,q)$-NPP solution if each component in $T'_{/\mathcal{Q}}$ is non-polymorphic in c_b. For any set of vertices $\mathcal{Q}$, this can be tested in $O(m)$ time. We repeat this procedure for values of q starting from 1 to a given maximum value k for q. There are exactly m vertices in T', and there are $\binom{m}{q+1} \cong m^{q+1}$ ways in which $q+1$ vertices can be selected from the m vertices. Therefore, in theory, the complexity of the algorithm is $O(nm^2 + m^{q+2})$ for a given q.

In practice, however, the algorithm can be implemented to run much faster. The following observations reduce the search space significantly:

- If two rows r_1 and r_2 in M with $M[r_1, c_b] = 0$ and $M[r_2, c_b] = 1$ both map to the same vertex z in T', then we call the vertex z as a *polymorphic vertex* with respect to c_b. For obvious reasons, all polymorphic vertices in T' must be expanded into edges labeled with c_b in any H$(1, q)$-NPP for M. Let V_p be the set of polymorphic vertices in T' with respect to c_b.

- Let S_c be the set of sites in G_c that are adjacent to c_b. Each one of the $q+1$ vertices selected for expansion must be incident on an edge labeled with a site in S_c. Therefore, the $q+1$ vertices have to be selected out of l vertices, where $l \leq m$ is the number of distinct vertices in T' that are incident on a edge labeled with a site in S_c. In general, if the degree of c_b in G_c is d, l will be less than or equal to $2d$. Let V_a be the set of vertices in T' that are incident on an edge in S_c.

- Let T_c be the subtree(or forrest) in T' formed exclusively by the sites in S_c. All the leaves of T_c must always be selected for expansion into edges labeled with c_b. Let V_l be the leaves of T_c in T'.



M	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
r_1	1	0	0	0	0	0	0	0	0	0	1
r_2	0	1	0	0	0	0	0	0	0	1	0
r_3	0	0	1	0	0	0	0	0	0	1	1
r_4	0	0	0	1	1	0	0	0	0	0	0
r_5	0	0	0	1	0	1	1	0	0	1	1
r_6	0	0	0	1	0	1	0	1	0	1	0
r_7	0	0	0	1	0	1	0	0	1	0	1

Figure 4.6: (a) Matrix M; (b) The conflict graph for the matrix M;(c) The tree T' after removing c_{10} and c_{11}

Let $m_c = |V_a|$, and let $m_g = |V_p \bigcup V_l|$. The actual number of sets $\mathcal{Q}$ that need to be searched is given by $\binom{m_c - m_g}{q+1-m_g}$. Hence, for any matrix M, q will be greater than or equal to $m_g - 1$.

4.2.4 Allowing Homoplasy Events in Multiple Sites

Extending the problem even further, we define the H(p, q)-NPP problem. An H(p, q)-NPP is a phylogeny in which at most p sites have homoplasy events, with at most q homoplasy events in each site. The conflict graph in this case will have multiple connected components and/or multiple vertices with degree greater than 1.

Let G'_c be the graph obtained by removing all degree-0 vertices from G_c. If the matrix M is to admit an H(p, q)-NPP, G'_c must have a vertex cover with size less than or equal to p. If such a vertex cover $\mathcal{C}$ is found, removing the vertices in $\mathcal{C}$ from G_c will result in a graph with no non-trivial connected components. We will be able to construct a perfect phylogeny T'

for the vertices in $S-\mathcal{C}$. Once T' is constructed, adding any site in $\mathcal{C}$ to T' is an H$(1,q)$-NPP problem.

A necessary (but not sufficient) condition for the existence of an H(p,q) solution is that for each site $i \in \mathcal{C}$, the set of sites $\{S-\mathcal{C}\}\bigcup\{i\}$ must have a H$(1,q)$ solution. However, adding multiple sites in $\mathcal{C}$ to T' is a more difficult problem. Even if each of the p sites in $\mathcal{C}$ can be added to T' to form H$(1,q)$-NPPs, it does not necessarily imply that the matrix M has an H(p,q)-NPP solution. For example, refer to Figure 4.6. The conflict graph for matrix M in Figure 4.6a is shown in Figure 4.6b. The tree T' after removing c_{10} and c_{11} is shown in Figure 4.6c. A H$(1,2)$-NPP can be constructed by adding either c_{10} or c_{11} T', but there is no H$(2,2)$-NPP that includes both c_{10} and c_{11}.

Therefore, to solve the H(p,q)-NPP problem, we need to determine if there is a way to combine the p individual H$(1,q)$-NPP solutions into a H(p,q)-NPP solution. For each site i in $\mathcal{C}$, let Q_i be the set of vertices in T' which have to be expanded into edges labeled with site i in order to add the site i to T' to form an H$(1,q)$-NPP. For each vertex x in T', let $P_x = \{i | x \in Q_i\}$.

Definition. A site $i \in \mathcal{C}$ is *fully specified* at a vertex $x \in T'$ with respect to an H$(1,q)$ solution consisting of the vertices Q_i if *any one* of the following conditions are satisfied:

1. At least one row in M maps to the vertex x.

2. The vertex x is in a connected component $T_x \in T'_{/Q_i}$, and at least one row in M maps to a vertex in T_x.

Let x and y be two vertices that are adjacent to each other in T'. We define that the two vertices x and y are *pair-wise independent* with respect to a set of $\mathrm{H}(1,q)$ solutions for the sites in $\mathcal{C}$ if all of the following conditions are satisfied:

1. Every site $i \in P_x$ is fully specified with respect to Q_i at the vertex y.

2. Every site $j \in P_y$ is fully specified with respect to Q_j at the vertex x.

3. $|P_x \bigcap P_y| = 0$.

A vertex x in T' is defined to be *isolated* (w.r.to the given set of $\mathrm{H}(1,q)$ solutions) if x is pair-wise independent with all the vertices adjacent to it.

Each vertex x in T' must be replaced by a phylogeny $\mathcal{T}_x$ over the sites in P_x. The phylogeny $\mathcal{T}_x$ should be a phylogeny where the taxa include the following:

- The states of the sites in P_x in each row (if any) of M that map to the vertex x.

- For each site y adjacent to x, the state of the sites in P_x at the vertex y.

For example, the vertex x in Figure 4.6 should be replaced by a phylogeny $\mathcal{T}_x$ over the sites $\{c_{10}, c_{11}\}$, where the taxa are $\{00, 01, 10, 11\}$.

When the vertex x is isolated, it can be trivially shown that the following conditions hold true:

1. All the node labels that must label some node in the phylogeny $\mathcal{T}_x$ are known.

2. For any vertex y adjacent to x, there will be a vertex u in $\mathcal{T}_x$ and a vertex v in $\mathcal{T}_y$ such that $\mathcal{L}(u) = \mathcal{L}(v)$. Therefore, the edge (x, y) in T' can be replaced by the edge (u, v) in a phylogeny that includes all the vertices in $\mathcal{C}$, and edge (u, v) will not require any more mutations than the edge (x, y).

When any vertex x in T' is not isolated, and/or if $\mathcal{T}_x$ is not a perfect phylogeny, the H(p, q)-NPP problem is quite complicated. The phylogenies $\mathcal{T}_x$ and $\mathcal{T}_y$ that replace adjacent vertices will be interdependent, and replacing the edge (x, y) with an edge between some node in $\mathcal{T}_x$ and some node in $\mathcal{T}_y$ might incur additional cost. For example, refer to Figure 4.7. Let x and y two vertices adjacent to each other with $|P_x \bigcap P_y| = 3$. Let i, j and k be the sites that are common in P_x and P_y, and let $\mathcal{T}_x$ be the phylogeny shown in Figure 4.7a and $\mathcal{T}_y$ be the phylogeny shown in Figure 4.7b. As there are no common vertices in $\mathcal{T}_x$ and $\mathcal{T}_y$, connecting a vertex in $\mathcal{T}_x$ to a vertex in $\mathcal{T}_y$ requires at least one additional mutation in the sites i, j or k.

We leave the unrestricted H(p, q)-NPP problem as an open problem. However, when the following conditions are satisfied, there is a simple solution to the H(p, q)-NPP problem:

1. Each vertex in T' is isolated with respect to the given set of H$(1, q)$ solutions.

2. For each vertex x in T', the phylogeny $\mathcal{T}_x$ that must replace the vertex x is a perfect phylogeny.

When the above two conditions are satisfied, each vertex x can be simply replaced by the perfect phylogeny $\mathcal{T}_x$. As x is isolated, each edge incident on the vertex x in T' can be replaced by an edge incident on some vertex in $\mathcal{T}_x$, without incurring any additional cost.

4.2.4.1 Complexity

Finding all vertex covers in G'_c with size at most p takes exponential time with respect to p. Assuming the size of G_c is $O(m)$, finding all such vertex covers takes $O(m^{p+1})$ time. For each vertex cover, we need to construct the initial perfect phylogeny T', and find a H$(1, q)$-NPP solution for each site in $\mathcal{C}$. If the set of H$(1, q)$-NPP solutions satisfy the conditions described above, replacing each vertex in T' by a perfect phylogeny takes $O(np)$ time. Hence the over all complexity of the restricted version of the problem is $O(nm^2 + m^{p+1} + \eta p m^{q+2})$ time, where η is the number of distinct vertex covers of G'_c with size less than or equal to p.

4.2.4.2 Special Scenarios

A special situation arises when each non-trivial connected component in G_c has at most one site with degree greater than 1. In that case, p will be equal to the number of non-trivial connected components in G_c. The set $\mathcal{C}$ is fixed. This reduces the problem to p completely independent H$(1, q)$-NPP problems that can be solved in $O(nm^2 + pm^{q+2})$ time. In general,

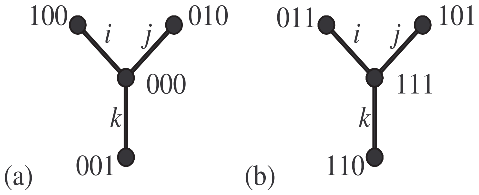


Figure 4.7: An example of phylogenies (a) $\mathcal{T}_x$ and (b) $\mathcal{T}_y$ that must replace two adjacent vertices x and y when x and y are not independent. The node labels of each node over three sites i, j and k are shown.

each connected component in G_c that is either a single edge or involves a single vertex with degree greater than 1 will reduce the effective value of p by 1.

4.3 Near-Perfect Phylogeny Haplotyping

In case of the NPPH problem, the input is a set of genotypes. The aim in general is to construct a set of haplotypes that are the most likely explanation for the given set of genotypes. Parsimony is widely accepted as the most accurate criterion to reconstruct the phylogeny. Therefore, the aim is to obtain, out of all possible explanations for the given genotypes, the set of haplotypes that admit a phylogeny with the least number of recurrent mutations.

4.3.1 The H1-NPPH Problem

We formally state the H1-NPPH problem as follows. We are given an $n \times m$ genotype matrix A over the alphabet $\{0, 1, 2\}$. Each row in A represents a genotype. As before, the columns represent SNP sites. The aim is to construct a $2n \times m$ haplotype matrix M such that:

1. Each row r in A is a result of combining the rows r and r' in M
2. The matrix M admits an H1-NPP.

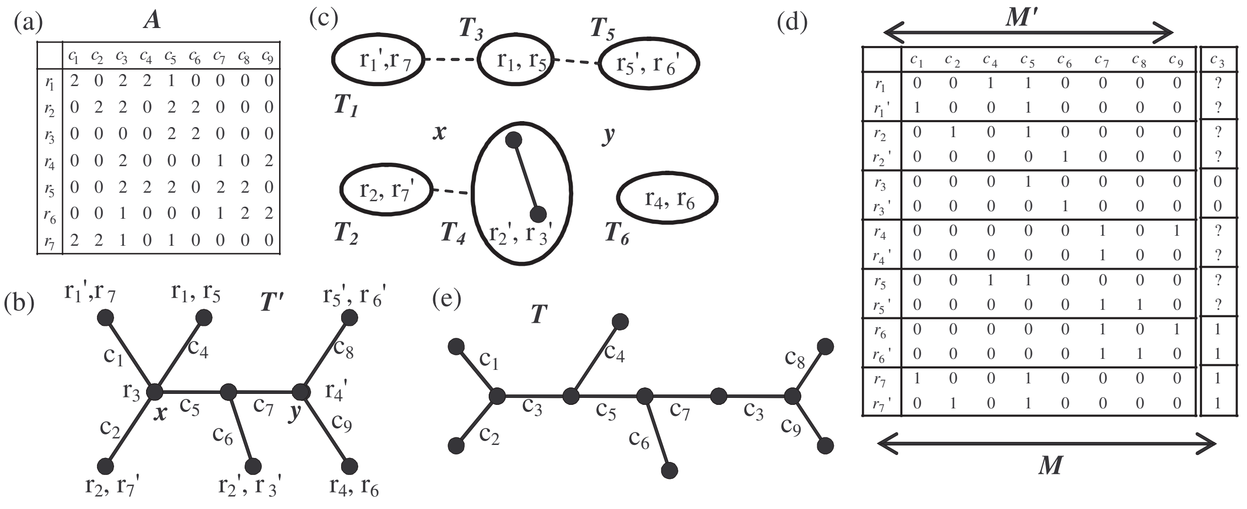
The solution to the H1-NPPH problem is very similar to that for the H1-NPP problem, except that it might not be possible to fully construct the conflict graph for a genotype matrix. In a genotype matrix A, an ordered pair of values (a, b), $a \in \{0, 1\}$, $b \in \{0, 1\}$ is in $I(i, j)$ for a pair of columns (i, j) if

1. There is a row r in A such that $A[r, i] = a$ and $A[r, j] = b$,or
2. $A[r, i] = a$ and $A[r, j] = 2$, or
3. $A[r, i] = 2$ and $A[r, j] = b$.

If two columns i and j are '2' in some genotype, the states of i and j in the two haplotypes for the genotype could be either $\{(0, 0), (1, 1)\}$ or $\{(0, 1), (1, 0)\}$. Therefore, we might not be able to completely specify $I(i, j)$. $I(i, j)$ can be completely specified only in two situations: when $|I(i, j)| = 4$ because of rows in A in which either the column i or the column j is not '2', or when there are no rows in A in which both i and j are '2'. Hence, though we might be able to construct some edges in the conflict graph in G_c, we might not be able to construct all the edges in G_c. Therefore, we need other ways to find the column c_b that has a recurrent mutation. One obvious procedure for finding c_b is to remove each column from A, and check if the rest of the matrix admits a perfect phylogeny. If we can find such a column c_b, then there might be a H1-NPPH solution for A. This is the procedure used in [SWG05] to find the column c_b. We adopt the same procedure to find c_b. Then, we propose our new algorithm to construct H1-NPPH solution.

Once the column c_b is found, we can build the perfect phylogeny T' for the matrix A' obtained by removing c_b from A. In general, the matrix A' might have multiple perfect phylogenies. Chung and Gusfield [CG02] have empirically shown that the likelihood for the phylogeny being unique increases quickly with the number of genotypes. In the following, we assume that A' has a unique perfect phylogeny T'. If A' admits multiple perfect phylogenies, the following procedure has to be repeated for each such perfect phylogeny.

Using the phylogeny T', we construct the haplotype matrix M' for A'. We denote the rows of A' by $r_1, r_2, ..., r_n$ and the corresponding pairs of rows in M' as $r_1, r'_1, r_2, r'_2, .., r_n, r'_n$. The matrix M should now be built by adding the column c_b to M'. We can also assign values to some rows in column c_b of the matrix M. In a row r_i of A, if $A[r_i, c_b]$ is either 0 or 1, then both the haplotypes for this row will also be either 0 or 1, respectively, in column c_b. We can then set $M[r_i, c_b] = M[r'_i, c_b] = A[r_i, c_b]$. When $A[r, c_b] = 2$, we know that $M[r_i, c_b] = \overline{M[r'_i, c_b]}$, but we can not determine which one of them must be 0 for M to admit an H1-NPP. We call such a pair of rows (r_i, r'_i) in M as an *ambiguous pair.* Thus



	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
r_1	2	0	2	2	1	0	0	0	0
r_2	0	2	2	0	2	2	0	0	0
r_3	0	0	0	0	2	2	0	0	0
r_4	0	0	2	0	0	0	1	0	2
r_5	0	0	2	2	2	0	2	2	0
r_6	0	0	1	0	0	0	1	2	2
r_7	2	2	1	0	1	0	0	0	0

	c_1	c_2	c_4	c_5	c_6	c_7	c_8	c_9	c_3
r_1	0	0	1	1	0	0	0	0	?
r_1'	1	0	0	1	0	0	0	0	?
r_2	0	1	0	1	0	0	0	0	?
r_2'	0	0	0	0	1	0	0	0	?
r_3	0	0	0	1	0	0	0	0	0
r_3'	0	0	0	0	1	0	0	0	0
r_4	0	0	0	0	0	1	0	1	?
r_4'	0	0	0	0	0	1	0	0	?
r_5	0	0	1	1	0	0	0	0	?
r_5'	0	0	0	0	0	1	1	0	?
r_6	0	0	0	0	0	1	0	1	1
r_6'	0	0	0	0	0	1	1	0	1
r_7	1	0	0	1	0	0	0	0	1
r_7'	0	1	0	1	0	0	0	0	1

Figure 4.8: (a) Matrix A; (b) The tree T'; (d) Components in $T'_{/\{x,y\}}$ overlaid with the edges in G_a; (d) Matrices M' and M; (e) The H1-NPP T for the matrix M

the problem of determining whether A admits an H1-NPP solution reduces to determining whether there is an assignment of values to each such ambiguous pair so that matrix M admits an H1-NPP.

Each row in M' (and hence in M) can be mapped to a vertex in T'. As in the H1-NPP case, we represent this mapping using the notation $\nu(r_i) = v$, where r_i is a row in M, and v is a vertex in T'. For any vertex v in T', zero or more rows in M can map to vertex v.

The underlying idea of our algorithm is based on Theorem 1. We need to identify two vertices x and y, if they exist, such that each connected component in $T'_{/\{x,y\}}$ is non-polymorphic with respect to c_b. We will show how to use this property to actually obtain an assignment of values to each ambiguous pair of rows in M. We arbitrarily choose two vertices x and y in T' and construct a graph $G_a = (V, E)$, where the vertices in V correspond one-to-one to connected components in $T'_{/\{x,y\}}$. For each ambiguous pair (r_i, r'_i) in M, we know that $M[r_i, c_b] = \overline{M[r'_i, c_b]}$. Therefore, if $\nu(r_i)$ is in a component T_i, and $\nu(r'_i)$ is in T_j, we add the edge (v_i, v_j) to E. As each connected component T_i has to be non-polymorphic in c_b, if any un-ambiguous row r_j maps to a vertex in T_i, we assign the value $M[r_j, c_b]$ to the vertex v_i. Since the value of $M[r_j, c_b]$ is either 0 or 1, we can imagine these two values to represent two 'colors'. Thus, if the chosen pair of vertices $\{x, y\}$ leads to a valid assignment of values to the ambiguous pairs of rows, each connected component in G_a should be two-colorable with the coloring scheme of vertices in G_a as described. Intuitively, a valid two coloring is possible only if the following is true: Let R_0 be the set of rows in M such that $M[r, c_b] = 0$ and similarly let R_1 be the set of rows in M such that $M[r, c_b] = 1$. Then each component G_a has a valid two coloring if and only if for each $T_i \in T'_{/\{x,y\}}$, $R(T_i)$ is a subset of either R_0 or R_1.

If G_a is two colorable given the current coloring of the vertices, each un-colored vertex in G_a can be assigned a color (value) of 0 or 1. When a vertex v_i is assigned a value $a \in \{0, 1\}$, we can assign $M[r, c_b] = a$ for every row r such that $\nu(r)$ is in T_i and $M[r, c_b]$ is un-assigned. After every unknown entry in column c_b of M is filled like this, each connected component $T_i \in T'_{/\{x,y\}}$ will be non-polymorphic in c_b, and hence T' can be converted into an H1-NPP T for M.

Figure 4.8 shows each step of the procedure. A matrix A is shown in 4.8a. The perfect phylogeny after removing column c_3 from A is shown in Figure 4.8b. The matrices M' and M, constructed through T' are shown in Figure 4.8d. The components in $T'_{/\{x,y\}}$ are shown in Figure 4.8c. Since the rows r_1 and r'_1 in M form an ambiguous pair, components T_1 and T_3 in G_a are connected. Similarly, components T_2 and T_4 will be connected due to the ambiguous pair (r_2, r'_2), and components T_3 and T_5 are connected due to ambiguous pair (r_5, r'_5). These edges are shown using dashed lines in Figure 4.8c. Though the rows r_4 and r'_4 also form an ambiguous pair, no edge is added to G_a since one of them (r'_4) maps to the vertex y. Since y will be expanded into two vertices y_0 and y_1, r'_4 can map to any of the two vertices y_0 and y_1, and hence the pair of rows (r_4, r'_4) does not impose any restriction on the coloring of the vertices in G_a. Components T_1, T_2, T_4, T_5 and T_6 can similarly be assigned a color of 1 because of the rows r_7, r'_7, r'_3, r'_6 and r_6, respectively. The connected component T_3 can not directly be assigned any color, since no unambiguous row maps to it. It can be seen that G_a is two-colorable, and the only possible coloring is to assign color 0 T_3. The final H1-NPP T is shown in Figure 4.8e.

The fundamental problem now is how to find the two sites x and y in T'. In case of the H1-NPP problem in Section 4.2, the conflict graph G_c could be constructed, RMP could be deduced from G_c, and the two vertices x and y could be directly selected as the terminal ends of RMP. In case of the H1-NPPH problem, since we can not construct the conflict graph completely (unless in very obvious special scenarios), we must exhaustively search for the vertices by checking each pair of vertices in T'. Since there are exactly m vertices in T', there will $O(m^2)$ pairs of vertices that we need to check.

For each pair of vertices, the graph G_a can be constructed in $O(n + m)$ time, allowing parallel edges. Since there are at most $O(n)$ edges in G_a (at most one for each row in A), the connected components in G_a can be identified in $O(n+m)$ time using depth-first search. Two-coloring of G_a can be obtained in $O(n+m)$ time using breadth-first search. Hence, the overall complexity of the algorithm is $O(m^2(n+m))$.

It might seem that the $O(n^4)$ algorithm of Song et al. [SWG05] might perform better if $m > n$. However, m can never be greater than $O(n)$ without having duplicate rows in M. This is because even if each of the $2n$ haplotypes are distinct, there can be no more than $4n-4$ edges in the tree. With only one homoplasy event, each column except c_b has to label a distinct edge, and hence there can be at most $4n-3$ distinct columns in the matrix M. If the matrix M has more than $4n-3$ distinct columns, it will not admit an H1-NPP.

On the other hand, n can be as high as $O(m^2)$. Hence, our algorithm has better time-complexity than the previous $O(n^4)$ algorithm for any value of n and m.

4.3.2 Making use of the conflict graph

The conflict graph provides useful information that can be utilized to speed up the above algorithm. Even though it might not be possible to build the conflict graph completely, we can make use of what is available of the conflict graph in order to reduce the $O(m^2)$ search space of the pairs of vertices.

Inferring the recurrent mutation path

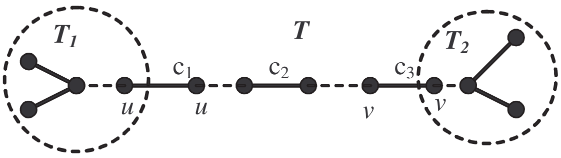


Figure 4.9: Any solution must involve a vertex from T_1 and a vertex from T_2

From the discussion in Section 4.2, it is clear that the set S_c of sites adjacent to c_b in the conflict graph must all lie in a path in T'. Let $S_c = \{c_1, c_2, c_3\}$, and let all three of them lie in a path in T', as shown in Figure 4.9. If the matrix A admits an H1-NPP, the path between the two vertices x and y that are selected to be expanded must clearly include all the sites in S_c. Therefore, one of them (say, x) has to be in T_1 and the other (say, y) has to be in T_2, as shown in Figure 4.9. Therefore, the conflict graph can be effectively used to reduce the pairs (x, y) that need to be checked. The following is another interesting result:

Lemma 4.1 *The sites in S_c must form a contiguous path in T' if the matrix A admits an H1-NPP.*

i.e, the sites c_1, c_2 and c_3 must form a contiguous path, instead of a broken path as depicted in Figure 4.9.

4.3.2.1 Using the ambiguous pairs more effectively

For any ambiguous pair of rows (r, r') in M, the path between the vertices $\nu(r)$ and $\nu(r')$ must include an edge (in general, an odd number of edges) labeled with c_b. This means that any pair of vertices x and y in T' that are a possible solution must be such that $\nu(r)$ and $\nu(r')$ are not in the same connected component $T_i \in T'_{/\{x,y\}}$. The following lemma states this property formally:

Lemma 4.2 *For any two vertices x and y in T' that can be expanded to form a H1-NPPH solution for matrix A, the path between the vertices $\nu(r)$ and $\nu(r')$ for every ambiguous pair (r, r') must include the vertex x or y or both.*

Proof Let there be an ambiguous pair (r, r') in M so that the path in T' between the two vertices $\nu(r)$ and $\nu(r')$ does not include both x and y. This means that the vertices $\nu(r)$ and $\nu(r')$ are in the same connected component $T_i \in T'_{/\{x,y\}}$. Since $M[r, c_b] = \overline{M[r', c_b]}$, this implies that T_i is polymorphic with respect to c_b. Hence, there must be an edge within T_i labeled with c_b in addition to the two edges labeled with c_b inserted at the vertices x and y. Hence the two vertices x and y can not lead to an H1-NPPH solution for the matrix A. Therefore, for any pair of vertices x and y in T' that can be expanded into an H1-NPPH solution for matrix A, the path between the vertices $\nu(r)$ and $\nu(r')$ for every ambiguous pair (r, r') must include the vertex x or y or both. $\diamondsuit$

Lemma 4.2 can be used to avoid checking some vertex pairs. Let R be the set of rows in A such that $A[r, c_b] = 2$ for every $r \in R$. Let $R_x \subseteq R$ be the set of rows in A such that, for every $r \in R_x$, the path between the vertices $\nu(r)$ and $\nu(r')$ in T' includes the vertex x in T'. Similarly, let R_y be the corresponding set of rows for the vertex y in T'. The pair of vertices x and y can not be a solution unless $R = R_x \bigcup R_y$.

4.3.3 The H$(1, q)$-NPPH problem

The solution for the H$(1, q)$-NPPH problem is a simple extension to the solution for the H1-NPPH problem. All the discussion above applies to H$(1, q)$-NPPH problem, with the only difference being that instead of finding a pair of vertices x and y, we need to find a set of $q + 1$ vertices $\mathcal{Q}$ so that T' can be converted into an H$(1, q)$-NPP T by expanding each one of $q + 1$ vertices in $\mathcal{Q}$ into an edge labeled with c_b.

In case of the H$(1, q)$-NPP problem, we could use G_c to narrow down the possible sets of vertices for $\mathcal{Q}$. We can not do the same thing here, since G_c is not complete. Therefore, we need to try all-possible sets of vertices of size $q + 1$. There are $\binom{m}{q+1}$ such possible sets of vertices. For each set, testing if the set of vertices form a solution is identical to the procedure for the H1-NPPH problem - we build the graph G_a in which each vertex represents a connected component in $T'_{/\mathcal{Q}}$. As before, two vertices v_i and v_j have an edge between them if there is an ambiguous pair (r, r') in M so that the vertex $\nu(r)$ is in v_i and the vertex $\nu(r')$ is in v_j. We need to test if the graph G_a is two-colorable. As in the case of the H$(1, q)$-NPP problem, This algorithm can be implemented to run in $O(nm^2 + m^{q+1}(n + m)$ time.

4.3.4 The H(p, q)-NPPH problem

Like the H(p, q)-NPP problem, the H(p, q)-NPPH problem can be viewed as a set of H$(1, q)$-NPPH problems. We first need to find a set of p columns $\mathcal{C}$ so that the matrix A' obtained by removing the columns in $\mathcal{C}$ from A has a perfect phylogeny T'. Once T' is constructed, we can solve for each of the sites in $\mathcal{C}$ as an H$(1, q)$-NPPH problem. The haplotype matrix M can be constructed for a given set of H$(1, q)$-NPP solutions, and the H(p, q)-NPPH problem on the matrix A will be equivalent to the the H(p, q)-NPP problem on the matrix M. However, if any site $i \in \mathcal{C}$ has multiple H$(1, q)$-NPP solutions, there will be multiple such matrices M, and the matrix A will admit an H(p, q)-NPP if any one of those matrices admit a H$(1, q)$ NPP. The time complexity of the algorithm will be similar to that of the H(p, q)-NPP algorithm.

4.4 Results

We have implemented our algorithm for the H1-NPPH problem in C++. In this section, we compare the performance of our algorithm to that of PHASE [SSD01] using simulated data. To generate the simulated data, we follow the same procedure as in [SWG05]. We first generate homoplasy-free haplotype matrices with minimum allele frequency (MAF) $\geq 2\%$ using the program MS [Hud02]. In each matrix, we introduce a homoplasy column by randomly selecting two vertices in the perfect phylogeny for the dataset and expanding the two vertices into edges labeled with the newly introduced column. We ensure that the newly introduced column has a MAF $\geq 2\%$ by selecting two non-adjacent vertices for expansion. Finally, we construct the genotype matrix by pairing consecutive rows in the haplotype matrix.

Table 4.1: Comparison of our H1-NPPH method with PHASE for different datasets. The running times are on Pentium 3.2 GHz PC

test case	Our H-1 NPPH algorithm			PHASE		
$n \times m$	std. error	%of mis-phased 2's	run time	std. error	%of mis-phased 2's	run time
50 × 50	0.0116	0.157%	0.01s	0.0138	0.269%	109s
100 × 50	0.0054	0.064%	0.016s	0.0046	0.065%	268s
50 × 100	0.011	0.105%	0.031s	0.0156	0.214%	497s
100 × 100	0.0048	0.046%	0.047s	0.011	0.136%	874s

Table 4.2: Properties of the data sets generated

test case	#of datasets (out of 100) that admit a perfect phylogeny	#of datasets admitting H-1 NPPH solutions (with a unique PPH solution for A')
50 × 50	16	84 (49)
100 × 50	10	90 (54)
50 × 100	3	97 (55)
100 × 100	8	92 (42)

The results are summarized in Tables 4.1 and 4.2. We provide two measures of accuracy. The first measure, the standard error, is the ratio of the genotypes that are incorrectly inferred to the total number of genotypes in the data set. The second measure is simply the percentage of mis-phased 2s. We used 100 datasets for each problem size. The run-times and error-rates shown are averages for the hundred datasets.

4.5 Discussion

The algorithms and problem formulations we introduced here are applicable in a wide a variety of problems encountered in genome variation studies and population genetics. With the help of simulated data, we demonstrated that the algorithms are applicable and practical in case of the haplotype inference problem. We believe that these algorithms will also be practical for phylogenetic reconstruction problems in general. Specifically, the algorithms will be extremely useful for inferring phylogenies for haploid genomes, like mtDNA and the human Y-chromosome.

CHAPTER 5

THE INCOMPLETE PERFECT PHYLOGENY PROBLEM

5.1 Missing Data

Real biological data is generally *incomplete*. i.e., the state of some loci might not be known in each taxon. Under these circumstances, the problem of determining if there is a perfect phylogeny for the given taxa is called as the *incomplete perfect phylogeny* (IPP) problem. The IPP problem was proven to be NP-complete [Ste92], even when each character is bi-allelic. However, if at least one taxon in the input set is complete that taxon can be considered as the root for the phylogeny, and the problem is called as the incomplete directed phylogeny (IDP) problem. Peer et al [PPS04] have shown that the IDP problem is solvable in polynomial time, and presented an algorithm that takes an expected time of $\tilde{O}(nm)$, where n is the number of taxa and m is the number of characters. Halperin et al [HK04] took a different approach, and made certain assumptions about the input data, and presented a $\tilde{O}(nm)$ algorithm for the IPP problem that can be used when the input data satisfies those assumptions.

When the input consists of incomplete genotypes, the problem is called as the *incomplete perfect phylogeny haplotyping* (IPPH) problem. The IPPH problem is clearly NP-complete since the IPP problem can be viewed as a special case of the IPPH problem in which there are no heterozygous loci. Interestingly, even the rooted version of the IPPH problem was shown to be NP-complete [KS05].

In this chapter, we handle the IPP and IPPH problems in their original form, without making any assumptions about the input data. Using empirical analysis, we demonstrate that the IPP problem can almost always be solved in polynomial time, even when as much as 50% of the input data is missing. We extend this approach to the IPPH problem, and present an efficient algorithm for the IPPH problem. As stated in [HK04], the necessary and sufficient conditions under which an incomplete matrix admits a unique perfect phylogeny are unknown. We solve this open problem, and formulate a set of necessary and sufficient conditions under which any given IPP or IPPH instance has a unique solution.

5.2 Problem statement and Previous Work

As in the previous chapters, a complete haplotype is represented by a length-m vector over the alphabet $\{0, 1\}$, where 0 and 1 are representative of the two alleles in each position. An incomplete haplotype is a length-m vector over the alphabet $\{0, 1, ?\}$, where '?' represents missing data. A complete genotype is represented by a length-m vector over the alphabet $\{0, 1, 2\}$, where '0' or '1' indicate that the corresponding SNP is homozygous in the genotype with the '0' or '1' allele respectively, and '2' indicates that the corresponding SNP is heterozygous. An incomplete genotype is a length-m vector over the alphabet $\{0, 1, 2, ?\}$.

The input to the IPP problem is an $n \times m$ matrix M over the alphabet $\{0, 1, ?\}$. Each of the n rows in the matrix represent a haplotype. The incomplete perfect phylogeny (IPP) problem is to determine if there is an assignment of '0' or '1' to each '?' in M so that the resulting matrix admits a perfect phylogeny.

We define the following terms. An ordered pair (a, b), $a \in \{0, 1\}$, $b \in \{0, 1\}$, is said to be *induced* by a pair of ordered columns (i, j) if there is a row r in M such that $M[r, i] = a$ and $M[r, j] = b$. The set of ordered pairs induced by a pair of columns (i, j) is denoted by $I(i, j)$. According to the well-established four-gamete test, the matrix M will admit a perfect phylogeny only if $|I(i, j)| \leq 3$ for every pair of columns (i, j).

Halperin et al. [HK04] made the assumption that $|I(i, j)| = 3$ for every pair of columns (i, j) in M. They call this assumption *rich data hypothesis*. When an incomplete matrix M satisfies the rich data hypothesis, if there is a perfect phylogeny for M, the perfect phylogeny will be the unique perfect phylogeny for M. Under these conditions, they presented a rather involved procedure to recover a complete haplotype and construct the perfect phylogeny for M. In Section 5.5, we present a simple procedure that recovers a complete haplotype when the rich data hypothesis is satisfied. The procedure is applicable in many situations, even when the rich data hypothesis is not satisfied.

When the root, i.e., any complete haplotype that must be in the tree is available, the IPP problem can be solved as the IDP problem. Peer et al [PPS04] present an efficient solution for the IDP problem when the root is an all-0 vector. If the root is not an all-0 vector, it can be converted into an all-zero vector by flipping (replacing each '0' by '1' and each '1' by a '0') every column that is not ‘0’ in the root. Kimmel and Shamir [KS05] presented a worst-case exponential-time algorithm with expected time of $\tilde{O}(nm^2)$ when certain assumptions about the input data are satisfied. One of their assumptions is that $m = O(n^{0.5})$. Their algorithm, in fact, involves an exhaustive search through all-possible haplotype vectors that could be the root of the perfect phylogeny. For each root, they try all possible ‘phase’ relationships between pairs of columns in order to search for the solutions.

They construct a bipartite graph $G = (R, C, E)$, where C is the set of characters (columns) and R is the set of species (rows) of the matrix. An edge (c, r) is in E if the column c is ‘1’ in the row r. They make a very interesting observation - if any sub-graph of G induced by two vertices from C and three vertices from R is connected, then the matrix M will not admit a perfect phylogeny. This observation is equivalent to the 4-gamete rule stated before, but helps in obtaining an efficient solution for IDP. In case of the IPPH problem, Kimmel and Shamir [KS05] present an algorithm with expected time of $\tilde{O}(nm^2)$, when certain assumptions about the input data are satisfied. The most significant of these assumptions is that the number of columns in the matrix is much fewer than the number of rows. Specifically, they assume that $m = O(n^{0.5})$. Their algorithm, in fact, involves an exhaustive search through all-possible haplotype vectors that could be the root of the perfect phylogeny. For each root, they try all possible ‘phase’ relationships between pairs of columns in order to search for the solutions. Though their algorithm has exponential time worst-case complexity, they show that the algorithm takes $\tilde{O}(nm^2)$ time when the assumptions they make are satisfied.

Gramm et al. [GNS04] introduced a special case of IPPH problem, where the perfect phylogeny is known to be a path. They show even this problem, known as *Perfect Path Phylogeny Haplotyping*, is NP-hard.

5.3 Realizability conditions for the IPP problem

In this Section, we present the conditions under which a given undirected IPP instance admits a perfect phylogeny. Our algorithm for the IPP problem is based on these conditions. In the following, we introduce some definitions.

For any pair of columns (i, j), the set of *non-induced* pairs, denoted by $U(i, j)$, is given by $U(i, j) = \{(0, 0), (0, 1), (1, 0), (1, 1)\} - I(i, j)$. The four-gamete test can be re-stated in terms of the non-induced pairs as in the following sentence - any matrix M (complete or incomplete) does not have a perfect phylogeny if $|U(i, j)| = 0$ for any pair of columns (i, j). When $|U(i, j)| = 1$, the ordered pair $(f_{ij}, f_{ji}) \in U(i, j)$ is defined as the *forbidden pair* for the pair of columns (i, j), denoted by $\mathcal{F}(i, j)$. Throughout this chapter, we follow the notation that $\mathcal{F}(i, j) = (f_{ij}, f_{ji})$.

A column i is said to be *non-polymorphic* if there are no '1's or '0's in the sub-matrix $M[*, i]$. It can be trivially shown that non-polymorphic columns are un-informative, and hence can be removed from the matrix without effecting the matrix in any way.

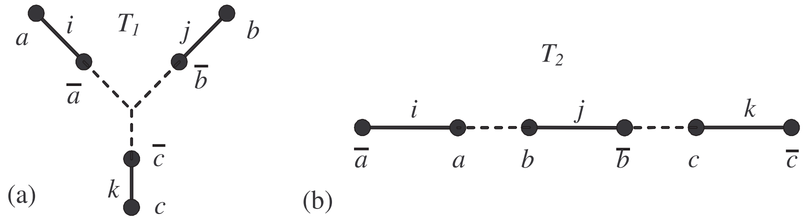


Figure 5.1: The two possible topologies for any three sites i, j and k in a perfect phylogeny

5.3.1 Significance of the forbidden pairs

In a perfect phylogeny, there are certain relationships between the forbidden pairs of any three columns. In any perfect phylogeny, the topology of the tree formed by a triplet of columns (i, j, k) must be one of the two topologies shown in Figure 5.1. i.e, the three of them must form a Y-shaped tree as shown in Figure 5.1a, or a path, as in Figure 5.1b. The edges can be labeled differently, but the overall topology must be either that in Figure 5.1a or that in 5.1b. Let $(a, \overline{a})$, $(b, \overline{b})$, $(c, \overline{c})$, be the pairs of alleles for the sites i, j and k respectively. Consider the labeling in Figure 5.1a. There can be no vertex in the perfect phylogeny T_1 with the allele a in site i and b in site j. Hence, $\mathcal{F}(i, j) = (a, b)$, where $f_{ij} = a$ and $f_{ji} = b$. Similarly, $f_{ik} = a$, $f_{ki} = c$, and $f_{jk} = b$, $f_{kj} = c$. Therefore, for the topology in T_1, $f_{ij} = f_{ik}$, $f_{ji} = f_{jk}$, and $f_{ki} = f_{kj}$, irrespective of how the edges are actually labeled. Similarly, for the topology in T_2, $f_{ij} = f_{ji}$, $f_{ki} = f_{kj}$, and $f_{ji} = \overline{f_{jk}}$, where j is the column in the middle. Therefore, in any perfect phylogeny, there are some restrictions and associations between the forbidden pairs of triplets of columns. In the following Sections, we present a formalization for these associations.

5.3.2 The 3-way compatibility expression

For any three distinct columns i,j and k with $\mathcal{F}(i,j) = (f_{ij}, f_{ji})$, $\mathcal{F}(j,k) = (f_{jk}, f_{kj})$, and $\mathcal{F}(i,k) = (f_{ik}, f_{ki})$, we define the *3-way compatibility expression*, denoted by $R(i,j,k)$:

$$R(i,j,k) = (E_j + E_j)(E_j + E_k)(E_k + E_i) = E_iE_j + E_jE_k + E_kE_i \tag{5.1}$$

where $E_i = 1 \oplus f_{ij} \oplus f_{ik}$, $E_j = 1 \oplus f_{ji} \oplus f_{jk}$ and $E_k = 1 \oplus f_{ki} \oplus f_{kj}$. Here, '+' is the logical OR operator, and '$\oplus$' is the logical XOR operator. We define the three columns i, j and k to be *3-way compatible* if $R(i,j,k) = 1$, and *3-way incompatible* if $R(i,j,k) = 0$.

Theorem 5.1 *A complete matrix M with $|I(i,j)| = 3$ for every pair of columns i and j admits a perfect phylogeny* iff *$R(i,j,k) = 1$ for every triplet of columns (i,j,k).*

Proof We first prove that M does not admit a perfect phylogeny if $R(i,j,k) = 0$ for any triplet of columns (i,j,k). Since the expression R is symmetric with respect to E_i, E_j and E_k, we only prove the case when $E_i = E_j = 0$. From the definition of E_i and E_j, we get:

$$f_{ij} = \overline{f_{ik}} \text{ and } f_{ji} = \overline{f_{jk}} \tag{5.2}$$

Since $|I(i,j)| = 3$ and $\mathcal{F}(i,j) = (f_{ij}, f_{ji})$, the pair $(\overline{f_{ij}}, f_{ji})$ is in $I(i,j)$, and there will be a row r_1 in M with $M[r_1,i] = \overline{f_{ij}} = f_{ik}$ and $M[r_1,j] = \overline{f_{ji}} = f_{jk}$. Similarly, there will be a row r_2 in M with $M[r_2,i] = \overline{f_{ik}} = f_{ij}$ and $M[r_2,k] = f_{ki}$. Without loss of generality, assume that $M[r_1,k] =?$ and $M[r_2,j] =?$. We will show that any assignment of values to $M[r_1,k]$ and $M[r_2,j]$ leads to a forbidden pair for some pair of columns.

Since $M[r_1, i] = f_{ik}$ and $M[r_1, j] = f_{jk}$, in order to avoid a forbidden pair in row in r_1, we must assign $M[r_1, k] = \overline{f_{ki}} = \overline{f_{kj}}$. This implies that $f_{ki} = f_{kj}$. In row r_2, since $M[r_2, i] = f_{ij}$, $M[r_2, j]$ must be equal to $\overline{f_{ji}}$ in order to avoid $\mathcal{F}(i, j)$. However, since $\overline{f_{ji}} = f_{jk}$, this means that f_{ki} cannot be equal to f_{kj} in order to avoid having $\mathcal{F}(j, k)$ in r_2. Therefore, it is not possible to complete both the rows r_1 and r_2 without introducing the forbidden pair in some pair of columns. Hence, the matrix M does not admit a perfect a phylogeny when $R(i, j, k) = 0$ for any triplet of columns.

Now, we prove that M admits a perfect phylogeny when $R(i, j, k) = 1$ for every triplet of columns (i, j, k). This means that we should be able to assign values to all missing entries in M without inducing the forbidden pair for any pair of columns. The proof is by contradiction. Assume that there is some entry $M[r, k] =?$ which cannot be assigned a value without forcing a forbidden pair for some pair of columns. This can only happen if there are at least two columns i and j such that $M[r, i] = f_{ik}$, $M[r, j] = f_{jk}$ and $f_{ki} = \overline{f_{kj}}$. If $M[r, k]$ is set to f_{ki}, $\mathcal{F}(i, k)$ will be induced into row r. Since $R(i, j, k) = 1$, $f_{ki} = \overline{f_{kj}}$ (i.e., $E_k = 0$) implies that $f_{ij} = f_{ik}$ ($E_i = 1$) and $f_{jk} = f_{ji}$ ($E_j = 1$). This implies that $\mathcal{F}(i, j)$ is already induced by the row r. However, this is not possible, since we know that $|I(i, j)| = 3$. Therefore, there can be no such entry $M[r, k]$ in M, and every '?' in M can be assigned a 0 or 1 so that there is a perfect phylogeny for the resulting matrix.$\diamondsuit$

Theorem 5.1 can be better understood from the matrix representation of the forbidden pairs shown in Figure 5.2-(b). The variables along the diagonal are not defined. The terms A, B and C in $R(i, j, k)$ are relationships between the two variables in the rows i, j and k,

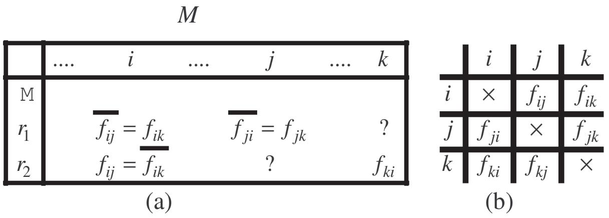

M
.... i .... j .... k
M
r_1 $\overline{f_{ij}} = f_{ik}$ $\overline{f_{ji}} = f_{jk}$?
r_2 $f_{ij} = \overline{f_{ik}}$? f_{ki}
(a)
i j k
i × f_{ij} f_{ik}
j f_{ji} × f_{jk}
k f_{ki} f_{kj} ×
(b)

Figure 5.2: (a) The rows r_1 and r_2 in M; (b) A matrix representation of the forbidden pairs

respectively. A, B or C will be zero if the two variables in the corresponding row are not equal to each other. The columns (i, j, k) will be 3-way compatible if variables in at least two rows are equal to each other.

In some situations, Theorem 5.1 allows us to define $\mathcal{F}(i, j)$ even if it is not directly induced by the matrix M. For example, consider the case when $\mathcal{F}(i, k)$ and $\mathcal{F}(j, k)$ are known, and $f_{ki} = \overline{f_{kj}}$. Applying Theorem 5.1 will tell us that f_{ij} must be equal to f_{ik} and f_{ji} must be equal to f_{jk} if M is to allow a perfect phylogeny. Hence we can indirectly define $\mathcal{F}(i, j)$ in this case, using Theorem 5.1.

5.3.3 Conditions for any matrix M

In the following, we answer this question - given the matrix M in which $|I(i, j)| < 3$ for some pairs of columns (i, j), is there an assignment $\mathcal{F}(i, j) = (f_{ij}, f_{ji})$ for every pair of such columns (i, j) that leads to a perfect phylogeny? In other words, is it possible to have a matrix in which the forbidden row cannot be defined for some pairs of columns and every possible assignment of forbidden pairs results in a matrix that does not admit a perfect phylogeny, but the original matrix allows a perfect phylogeny? To answer this question, we need to examine under which circumstances $|I(i,j)|$ can be less than 3 for two columns i and j in a perfect phylogeny.

Property 5.1 directly follows from the fact that each column in the resulting matrix M is polymorphic:

Property 5.1 *For any pair of columns i and j in the perfect phylogeny, $2 \leq |I(i,j)| \leq 3$.*

Also, it can be trivially shown that the following property holds true:

Property 5.2 *For any pair of columns i and j in the perfect phylogeny $|I(i,j)| = 2$ only if i and j label the same edge in T.*

Property 5.1 directly follows from the fact that every column in M in must be polymorphic. Property 5.2 is evident from Figure 5.3. Let i and j label the two edges as shown, with the two alleles of the site i being a and $\overline{a}$ and the two alleles of the site j being b and $\overline{b}$. As shown, the state of the columns (i,j) is (a,b) at vertex A and $(\overline{a},\overline{b})$ at vertex B. If there is any internal node C in the path from A to B (other than A and B), the state of C will be $(\overline{a},b)$. When $|I(i,j)| = 2$, there can be no such third node C. Therefore, i and j must label a single edge connecting the nodes A and B.

Property 5.2 leads to an additional result - Since $I(i,j) = \{(a,b),(\overline{a},\overline{b})\}$ when $|I(i,j)| = 2$, $I(i,j)$ must be either $\{(0,0),(1,1)\}$ or $\{(0,1),(1,0)\}$.

Theorem 5.2 *An incomplete matrix M with $|I(i,j)| < 3$ for some pairs of columns (i,j) will admit a perfect phylogeny* iff *there is a matrix M' obtained by adding additional rows to M so that (a) $|I(i,j)| = 3$ for every pair of columns (i,j) in M'; and (b) M' admits a perfect phylogeny.*

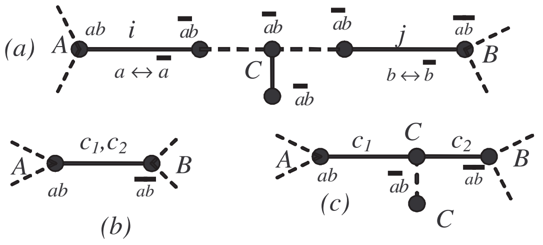


Figure 5.3: (a) Illustration of Property 5.1; (b) Columns c_1 and c_2 in the original tree;(c) Columns c_1 and c_2 after splitting the edge (A,B)

Proof Let M be a matrix that admits a perfect phylogeny T. From Property 5.1, we know that $I(i,j) \geq 2$ for every pair of columns (i,j) in a complete matrix M. Let c_1 and c_2 be two columns that label the same edge in T, as depicted by the edge (A,B) in Figure 5.3-(b). From Property 5.2, we know that $I|(c_1,c_2)| = 2$ in M. Clearly, the node labels of A and B are identical except in the sites c_1 and c_2. If $c_1 = a$ and $c_2 = b$ at A, c_1 and c_2 will be $\overline{a}$ and $\overline{b}$ at B. We can always introduce a new node C so that c_1 labels the edge (A,C) and c_2 labels the edge (B,C) by introducing an extant leaf C' as shown in Figure 5.3-(c). At vertices C and C', $c_1 = \overline{a}$ and $c_2 = b$, and every other column takes same value as at nodes A and B. If we add the label of the leaf C' to M', $|I(c_1,c_2)|$ will be equal to 3 in M'. The same can be done for every pair of columns (i,j) for which $|I(i,j)| = 2$ in M. Hence, for every incomplete matrix M that admits a perfect phylogeny, there will be a matrix M' in which $|I(i,j)| = 3$. $\diamondsuit$

Because of Theorem 5.2, we can use the 3-way compatibility expression to determine if a given matrix M allows a perfect phylogeny, even if $|I(i,j)| < 3$ for some pairs of columns in M. If M allows a perfect a phylogeny, $\mathcal{F}(i,j)$ can be defined for every pair of columns

(i, j). Applying the 3-way compatibility expression on any triplet of columns i, j and k, we obtain the following set of equations:

$$\begin{aligned} (1 \oplus f_{ij} \oplus f_{ik}) + (1 \oplus f_{ji} \oplus f_{jk}) &= 1 \\ (1 \oplus f_{ji} \oplus f_{jk}) + (1 \oplus f_{kj} \oplus f_{ki}) &= 1 \\ (1 \oplus f_{kj} \oplus f_{ki}) + (1 \oplus f_{ij} \oplus f_{ik}) &= 1 \end{aligned} \tag{5.3}$$

In total there will be $m(m-1)(m-2)/2$ such equations, since there are $m(m-1)(m-2)/6$ possible ways to choose i, j and k. The incomplete matrix will admit a perfect phylogeny only if there is an assignment of 0 or 1 to each variable that satisfies all these equations. In the special situation in which at least two out of the four variables in each expression can be assigned a value, the problem can be reduced to the 2-SAT problem, and can be solved in polynomial time.

For any pair of columns (i, j), if $|I(i, j)| = 3$, then both f_{ij} and f_{ji} will be known. When $|I(i, j)| = 2$, either f_{ij} or f_{ji} will be known, or one of them can be expressed as the other or the complement of the other. For example, $I(i, j) = \{00, 01\} \Rightarrow \mathcal{F}(i, j) \in \{10, 11\} \Rightarrow f_{ij} = 1$. Similarly, $I(i, j) = \{00, 11\} \Rightarrow \mathcal{F}(i, j) \in \{10, 01\} \Rightarrow f_{ij} = \overline{f_{ji}}$.

When $|I(i, j)| = 1$, f_{ij} and f_{ji} will be related by a disjunction. For example: $I(i, j) = \{00\} \Rightarrow \mathcal{F}(i, j) \in \{10, 01, 11\} \Rightarrow f_{ij} + f_{ji} = 1$. For the matrix M to admit perfect phylogeny, the above disjunctions have to be satisfied in addition to the equations 5.3.

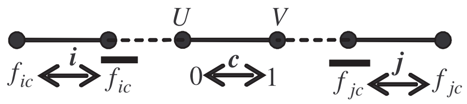


Figure 5.4: For any site labeling an edge (U, V), the state of any other site i at both the vertices U and V will be $\overline{f_{ic}}$

5.3.4 Properties of the forbidden pairs

It is convenient to represent the forbidden pairs using an $m \times m$ matrix F, where $F[i, j] = f_{ij}$ $\forall$ (i, j), $i \neq j$. The diagonal of the matrix, i.e., $F[i, i]$ $\forall$ i, is not defined. When $|I(i, j)| = 3$, both f_{ij} and f_{ji} can be assigned a value of 0 or 1. When $|I(i, j)| < 3$, we might be able to define one of the two variables (f_{ij}, f_{ji}), or introduce an equality or disjunction relationship between the two variables.

In any phylogeny T, we denote the node-label of a node V using the notation $\mathcal{L}(V)$. $\mathcal{L}(V)$ is a length-m haplotype vector. The following are some interesting properties of F.

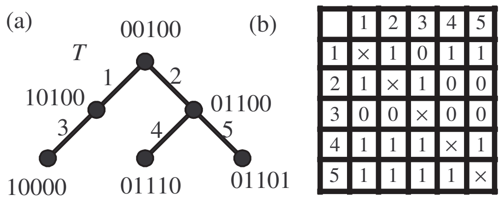
5.3.4.1 Each column in F represents two node labels in T

Assume the matrix M admits a perfect phylogeny T. Therefore, every site i in M labels a unique edge in T. Let a column c label an edge (U, V), where U and V are nodes in T. We show how to construct the node labels for U and V from F. Without loss of generality, let $\mathcal{L}(U)[c] = 0$, and $\mathcal{L}(V)[c] = 1$. For any column i in T, the state of the column i at both the nodes U and V will be $\overline{f_{ic}}$. This is irrespective of which 'side' of c the column i appears in T. Therefore, if we know f_{ic} for every site i, we will be able to build the node labels for both the vertices U and V. i.e., *if every entry (except F[c,c], which is not defined) in the column c in F is known, we can construct the node labels for the two nodes that define the edge labeled with column c.*

Let H_c be a vector formed by transposing the column c in F. The node labels for U and V can be constructed by setting $\mathcal{L}(U)[i] = \mathcal{L}(V)[i] = \overline{H_c[i]}\ \forall i \neq c$, and setting $\mathcal{L}(U)[c] = 0$ and $\mathcal{L}(V)[c] = 1$. Hence, if we can assign a value to every entry in a column in F, we can convert the IPP problem into the IDP problem. Note that the rich-data hypothesis need not be satisfied on the matrix M for us to be able to fill a column c in F completely. The algorithm for the IPP problem we present in Section 5.3 makes use of this property of F.

5.3.4.2 Each node label in T can be derived from a column in F

Another interesting property of F is that each node label in T can be obtained directly from some column in F. As described above, each column c in F describes the two node labels $\mathcal{L}(U)$, $\mathcal{L}(V)$ where (U, V) is the edge labeled by the site c in T. Therefore, we can derive the node label of any node X in T from any column i that labels an edge incident on X. There can be at most $m + 1$ nodes in T, and we can obtain $2m$ node-labels from F. The number of times a node-label is repeated in these $2m$ node-labels gives the degree of the corresponding node in T. For example, refer to Figure 5.5. A phylogeny T is shown in Figure 5.5a, and the



	1	2	3	4	5
1	×	1	0	1	1
2	1	×	1	0	0
3	0	0	×	0	0
4	1	1	1	×	1
5	1	1	1	1	×

Figure 5.5: (a) A perfect phylogeny T; (b) The forbidden matrix F for T

corresponding forbidden matrix F is shown in Figure 5.5b. The node label $\{01100\}$ can be derived from any of the three columns c_2, c_4, or c_5 in F. As the node with the label $\{01110\}$ is a leaf in the tree, it can be derived only one column (column c_4) in F. Also, notice that any row that is all-0 or all-1 in F is a site that is incident on a leaf in T. All three leaves of the tree satisfy this property.

5.4 Realizability Conditions for the IPPH problem

The input to the IPPH problem is a matrix $A = \{0, 1, 2, ?\}^{n\times m}$. Each row in the matrix A represents a genotype. Each genotype contains the conflated information about two haplotypes. Let H_1 and H_2 be two haplotype vectors that are conflated to produce a genotype G. If H_1 and H_2 have the same allele in a site i, i.e, if $H_1[i] = H_2[i] = a$, $a \in \{0, 1\}$, then $G[i] = a$. On the other hand, if the site i is heterozygous in G, $G[i] = 2$.

Formally stated, the IPPH problem is to determine if there is a $2n\times m$ complete haplotype matrix M so that:

1. M admits a perfect phylogeny

2. For every row r in A, there are two rows (r, r') in M such that $M[r,i] = \overline{M[r',i]}$ for every positions i in which $A[r,i] = 2$, and $M[r,i] = M[r',i] = A[r,i]$ in every position i in which $A[r,i] \in \{0,1\}$.

The IPP problem can be viewed as a special case of the IPPH problem in which there are no '2's in the matrix A. Therefore, the discussion and results in Section 5.3 are applicable for the IPPH problem too. The only difference is that the definition of the induced rows is slightly different, and an additional set of constraints apply on triplets of columns that are all '2' in the same row. For a genotype matrix A, a row ab, $a \in \{0,1\}$, $b \in \{0,1\}$ is in $I(i,j)$ for a pair of columns (i,j) if there is a row r in A such that $A[r,i] = a$ and $A[r,j] = b$, or $A[r,i] = a$ and $A[r,j] = 2$, or $A[r,i] = 2$ and $A[r,j] = b$. The definitions of $U(i,j)$ or $\mathcal{F}(i,j)$ do not change, as they are defined in terms of $I(i,j)$.

A triplet of columns (i,j,k) are said to be a *companion* triplet if there is a row r in A such that $A[r,i] = A[r,j] = A[r,k] = 2$. Since all the three columns i, j and k are heterozygous, in any perfect phylogeny T for A, i, j and k must mutate in the path between the two haplotypes for the row r. Hence, any companion triplet of columns must form a path topology, as shown in Figure 5.1b. There are three ways in which the columns i, j and k can label three edges in an un-directed path, each corresponding to the columns i, j or k labeling the 'inner' edge in the path. This restriction on a companion triplet of columns can be expressed in terms of the forbidden pairs as:

$$\overline{E_i}E_jE_k + E_i\overline{E_j}E_k + E_iE_j\overline{E_k} = 1 \tag{5.4}$$

where E_i, E_j and E_k are as described in Section 5.3.2. It can easily be seen that $\overline{E_i}E_jE_k = 1$ *iff* the columns i, j and k form a path with i in the middle. Similarly the other two terms in Equation 5.4 correspond to j being in the middle and k being in the middle. Equation 5.4 can be simplified to the following form:

$$f_{ij} \oplus f_{ji} \oplus f_{jk} \oplus f_{kj} \oplus f_{ik} \oplus f_{ki} \oplus (f_{ij} \oplus f_{ik})(f_{ji} \oplus f_{jk})(f_{ki} \oplus f_{kj}) = 1 \tag{5.5}$$

The matrix A will admit a perfect phylogeny *iff* Equation 5.3 is satisfied on every non-companion triplet of columns and Equation 5.5 is satisfied on every companion triplet of columns. An alternative way of arriving at Equation 5.5 is through the *phase* relationships [VM05, VM06] between pairs of columns. If a pair of columns (i, j) are both '2' in a genotype, the pair of columns in can be expanded as either $\{(0,0),(1,1)\}$ or $\{(0,1),(1,0)\}$ in the two haplotypes for the genotype. It has been previously established [BGL03, VM05, VM06] that every genotype in A in which the columns i and j are '2' must be expanded the same way if A is to admit a perfect phylogeny. This relationship between a pair of columns is defined as the *phase* between the two columns [VM05, VM06]. The phase between the pair of columns (i, j) is represented as $P(i, j)$. In terms of the forbidden pairs, the phase between a pair of columns (i, j) can be expressed as $P(i, j) = 1 \oplus f_{ij} \oplus f_{ji}$.

Assume that A admits a perfect phylogeny T. If three columns i, j and k are all '2' in some row r of the matrix A, the pairwise phase relationships will have some interdependencies, very similar to those introduced in [VM05]. Let H_1 and H_2 be the two haplotypes that combine to produce the row r in A. Since i, j and k are all '2' in row r, H_1 and H_2

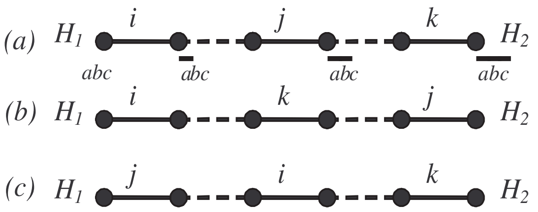


Figure 5.6: (a), (b) and (c): The three possible relative arrangements of the columns i, j and k that are all '2' in row r of A, in any perfect phylogeny T for A

differ in all three columns i, j and k. In T, the path between the vertices labeled with H_1 and H_2 must contain all the three edges labeled with i, j and k. Theorem 5.3 establishes the interdependencies between the pairwise relationships. Theorem 5.3 for the rooted PPH problem was first introduced by Bafna et. al. [BGL03] using different terminology. Here, we present a more general version of the theorem that is applicable to the un-rooted version of the problem.

Theorem 5.3 *In any genotype matrix A that allows a perfect phylogeny, if three columns i, j and k are all '2' in some row r, then the pairwise phase relationships are related by the expression $P(i,j) \oplus P(j,k) = P(i,k)$.*

Proof Let us consider the arrangement in Figure 5.6-(a). Let the two alleles for columns i, j and k be $\{a, \overline{a}\}$, $\{b, \overline{b}\}$ and $\{c, \overline{c}\}$, respectively. Let a, b, c be the alleles on the left of edges labeled i, j and k in Figure 5.6-(a). Therefore, the vertex labels will be abc(labeling $H1$), $\overline{a}bc$ (any vertex between the edges i and j), $\overline{a}\overline{b}c$ (any vertex between the edge j and k), and $\overline{a}\overline{b}\overline{c}$ (labeling $H2$). Clearly, $\mathcal{F}(i,j) = (a, \overline{b})$, $\mathcal{F}(i,k) = (a, \overline{c})$, and $\mathcal{F}(j,k) = (b, \overline{c})$. Hence,

$$P(i,j) \oplus P(j,k) = 1 \oplus f_{ij} \oplus f_{ji} \oplus 1 \oplus f_{jk} \oplus f_{kj}$$

$$\Rightarrow \qquad P(i,j) \oplus P(j,k) = a \oplus \overline{b} \oplus b \oplus \overline{c}$$

$$\Rightarrow \qquad P(i,j) \oplus P(j,k) = 1 \oplus a \oplus \overline{c} = P(i,k)$$

Similarly, it can be shown that relationships hold for the situations in Figure 5.6-(b) and Figure 5.6-(c). ◇

Theorem 5.3 is a generalization of 3.1. Adding Theorem 5.3 to Equation 5.3 gives Equation 5.5.

Hence, the only difference between the solutions for IPP and IPPH problems is the additional set of expressions as defined by the Equation 5.5 on all possible triplets of columns that are '2' in the same row. If any four of the six variables in all equations given by equations 5.3 and 5.5 are known, the IPPH problem can be solved in polynomial time. An obvious algorithm that checks every triplet to see if this is the case takes $O(m^3+nm^2)$ time. $O(nm^2)$ time will be necessary to obtain $\mathcal{F}(i,j)$ for each pair of columns, and $O(m^3)$ to evaluate the $O(m^3)$ expressions given by the equations 5.3 and 5.5.

5.5 Algorithms

An obvious solution for IPP and IPPH problems will be to obtain an assignment for every entry in F that satisfies the equations 5.3 and 5.5 and build the perfect phylogeny from F. However, this approach might be impractical, since there can be quite a few un-assigned entries in F. Our approach, instead, is to apply equations 5.3 and 5.5 in order to fill F to the fullest extent possible from information available in the matrix. In fact, all we need is to have one complete column in F. Using this complete column in F, we can convert the un-rooted versions of the problems into rooted versions of the same problem.

5.5.1 An algorithm for the IPP problem

In this Section, we present a practical algorithm for the IPP problem. For simplicity of illustration, we assume that each column in the input matrix M is polymorphic. As described earlier, non-polymorphic columns in M are un-informative, and will not label any edges in the perfect phylogeny for M. If there is a row r in M such that the vector $M[r]$ does not have any missing entries, then we can treat the vector $M[r]$ as the root, and solve the problem as an IDP problem, using the $\tilde{O}(nm)$ algorithm described in [PPS04]. In the following, we assume that no such complete row is directly available from the data. The algorithm first constructs the forbidden matrix F from M and applies the 3-way compatibility expression on triplets of columns to assign a value to as many entries in F as possible. Once a complete column in F is available, the root of the phylogeny can be derived, as described in Section 5.3.4. The IPP problem is then solved as the IDP problem.

The first step in the algorithm is to determine the set of induced pairs $I(i,j)$ for each pair of columns i and j. Constructing $I(i,j)$ for every pair of columns in the matrix takes $O(nm^2)$ time. The next step is to construct the $m \times m$ forbidden matrix F for M. When $|I(i,j)| = 3$, we can define (assign a value of 0 or 1) both f_{ij} and f_{ji}. When $|I(i,j)| < 3$, we might be able define one of the two variables (f_{ij}, f_{ji}), or introduce an equality or disjunction

1. Construct the matrix F from M by building $I(i,j)$ and inferring $\mathcal{F}(i,j)$ from $I(i,j)$. When $1 \leq |I(i,j)| < 3$, relate f_{ij} and f_{ji} by a disjunction or an equality so that all the restrictions imposed by $I(i,j)$ on $\mathcal{F}(i,j)$ are accounted for. If a column c in F is complete, derive the root from column c, and solve the problem as an instance of the IDP problem. Other wise, proceed to step 2.
2. Apply $R(i,j,k) = 1$ on triplets of columns from M until a column in F is complete or until no new assignments/equalities/disjunctions can be derived.
3. If a column c in F is complete, derive the root from c, and solve the problem as an IDP problem. Otherwise select a column c with the fewest un-assigned entries. Let p be the number of un-assigned entries in c.
    (a) For each of the 2^p possible ways in which the column c in F can be completed:
        i. Derive the root r from column c, and solve the problem as an IDP problem. If the problem can be solved as an IDP problem rooted at r, report the solution and halt.
4. report that the matrix M does not admit a perfect phylogeny.

Figure 5.7: The algorithm for the IPP problem

relationship between the two variables. A high-level description of the algorithm is given in Figure 5.7.

5.5.1.1 Obtaining all possible information from F

If there is no complete column in F, we apply condition $R(i,j,k) = 1$ on triplets of columns to fill the matrix F further. We continue to do this until either a full column is known, or until no further information can be obtained from F. For example, for any three columns c, i and j, if $f_{ci} = 0$ and $f_{cj} = 1$, then f_{ic} must be equal to f_{ij}, and f_{jc} must be set equal to f_{ji}, because of Equation 5.3. If any of the two variables f_{ic} and f_{ij} are known, the other could be assigned the same value as f_{ij}. If neither variable is known, one of them (say f_{ij}) is 'redirected' to the other(f_{ic}). i.e., all references to f_{ij} can be replaced with f_{ic}. There are multiple other scenarios where previously unknown variables can be assigned a value. In general, if two of the four variables in any of the three expressions in Equation 5.3 are known, it might be possible to infer some information about the others. This step of obtaining all the possible information from F can be implemented to run in $O(m^3)$ time.

5.5.1.2 Uniqueness of the solution

A given incomplete matrix M will have a unique perfect phylogeny if there is a unique way of filling F so that Equation 5.3 is satisfied on every triplet of columns. Each complete matrix F that satisfies Equation 5.3 has a unique perfect phylogeny T. This is because a complete matrix F refers to a hypothetical matrix M' in which $I|(i,j)| = 3$ for all pairs of columns. The incomplete matrix M consists of a subset of rows from M'.

5.5.2 Algorithm for the IPPH problem

As the rooted version of the IPPH problem is also NP-complete, just obtaining the root does not result in a solution. Therefore, the approach is to obtain all the information that can be obtained by applying Equations 5.3 and 5.5 on triplets of columns until no further information can be obtained. In practice, this leads to a situation in which most if the forbidden matrix F is filled. From F, we construct a $2m \times m$ haplotype matrix M by deriving two haplotypes from each column in F as described in Section 5.3.4.2. If the matrix A admits a perfect phylogeny T, then T must be among the IPP solutions for the matrix

M. Therefore, we can enumerate M all IPP solutions for the matrix M, and check if any of these solutions satisfy the Equation 5.5 on every companion triplet in A. In any practical instance of the problem, there will be very few solutions for IPP solutions for M. As a result, most practical instances of the IPPH problem can be solved in polynomial time. This algorithm is expected to be faster than the algorithm presented in [KS05], as their algorithm iterates through all possible phase relationships in A. Further, their algorithm has to further enumerate through all possible root vectors for A, whereas in our algorithm, the root might be directly available from the forbidden matrix F. Even when the root is not directly available from F, the number of candidate roots which are tested by our algorithm are expected to be fewer. This is because a column in F is more likely to be complete than a row in A.

A high-level description of the algorithm is presented in Figure 5.8.

5.6 Results

The algorithms were implemented in C++. The algorithms were tested on simulated data. First, haplotype matrices that admit perfect phylogenies using the program MS [Hud02]. For the IPPH case, we combine consecutive rows in the haplotype matrix to form genotypes. We then create incomplete haplotype/genotype matrices from these complete matrices by converting each entry in the matrix to a '?' with a fixed masking probability p. Therefore, any entry in the matrix has the same probability of being masked, and each entry is independently

1. Construct the matrix F from A. Derive all relationships between pairs of variables (f_{ij}, f_{ji}) from $I(i,j)$.
2. For every triplet of columns (i,j,k), apply $R(i,j,k) = 1$, and derive additional assignments/relationships. If i, j and k are all '2' in some row, apply Equation 5.5 on (i,j,k) to obtain additional assignemnts/relationships. Continue doing so until no new relationships are obtained.
3. If a column c in F is complete, derive the root from c. Otherwise select a column c with the fewest un-assigned entries. Let p be the number of un-assigned entries in c.
   (a) For each of the 2^p possible ways in which the column c in F can be completed:
      - Derive the root r from column c. Form an instance of a $2m \times m$ incomplete haplotype matrix M by deriving incomplete haplotypes from each column in F as described in Section 5.3.4.2.
      - Construct all possible IDP solutions for the matrix M. If there is any IDP solution for M that satisfies the phase relationships in matrix A given by the Equation 5.5 on every triplet of companion columns, report it and halt. If M does not have and IDP solution or if none of the IDP solutions for M satisfy Equation 5.5 on every triplet of companion columns, report that the matrix A does not admit a perfect phylogeny.

Figure 5.8: The algorithm for the IPPH problem

subjected to masking. The incomplete haplotype/genotype matrices created in this fashion are inputs to the IPP/IPPH algorithms.

Practical data sets that admit perfect phylogenies may not involve more than 30-50 loci. Therefore, we tested our algorithm for values of m up to one hundred. We tried masking probabilities ranging from 0.1 to 0.5, with increments of 0.1. We repeated the experiment 100 times for each problem size and each value of p.

5.6.1 Results for the IPP algorithm

Interestingly, the input data sets never satisfied the rich data hypothesis on all pairs of sites. For the 50×50 problem size with a masking probability of 0.1, a complete haplotype was

Table 5.1: Percentage of input data sets in which a complete column is directly available from F

$n \times m$	$p = 0.1$	$p = 0.2$	$p = 0.3$	$p = 0.4$	$p = 0.5$
50×50	98	97	91	81	54
50×100	100	97	80	53	14
100×50	100	100	100	98	90
100×100	100	99	99	95	68

Table 5.2: Percentage of input data sets in which a complete column was available from F after applying $R(i, j, k) = 1$ on triplets of columns

$n \times m$	$p = 0.1$	$p = 0.2$	$p = 0.3$	$p = 0.4$	$p = 0.5$
50×50	100	100	99	94	82
50×100	100	100	97	85	66
100×50	100	100	100	100	99
100×100	100	100	100	100	98

directly available from the input data 5% of the time. For the 100 × 50 problem size with a masking probability of 0.1, a complete haplotype was directly available in all the 100 input data sets. The complete haplotypes were not directly available in all the other test cases.

In Table 5.1, we show the percentage of time a complete column was available from F directly, before applying the 3-way compatibility expression on triplets of columns.

In Table 5.2, we show the percentage of time a complete column was available after $R(i, j, k) = 1$ was applied on triplets of columns. It is evident from the results that even with 50% missing data, the root can be effectively inferred from the matrix F in most situations. In the cases in which a complete column was not available in F even after applying $R(i, j, k) = 1$ on triplets of columns, there were at most two unknown values in the most complete column. Therefore the maximum number of root vectors tested (number of IDP instances tried) for any data set never exceeded 4.

Table 5.3: Performance on a pentium 3.2 Ghz pc - all times are in seconds, and are averages over 100 matrices

$n \times m$	$p = 0.1$	$p = 0.2$	$p = 0.3$	$p = 0.4$	$p = 0.5$
50×50	0.014	0.013	0.012	0.011	0.011
50×100	0.045	0.040	0.038	0.041	0.054
100×50	0.024	0.023	0.020	0.018	0.017
100×100	0.077	0.068	0.057	0.050	0.048

Table 5.4: Accuracy of the results - Percentage of loci incorrectly recovered

test case	$p = 0.1$	$p = 0.2$	$p = 0.3$	$p = 0.4$	$p = 0.5$
50×50	0.923	0.458	0.752	1.114	2.510
50×100	0.121	0.277	0.486	0.781	1.213
100×50	0.208	0.356	0.598	0.908	1.346
100×100	0.091	0.216	0.370	0.567	0.853

The performance of the algorithm in terms of speed is shown in Table 5.3. All the times are averages over 100 runs. The standard deviation for the run times varied greatly, and was as high as 20% for some test cases. It can be seen that the time taken is less than 0.1 seconds for all problem sizes. Also, it can be seen that time taken for a given problem size did not vary much for different masking probabilities.

The accuracy of the recovered haplotypes is shown in Table 5.4. The measure presented here is the percentage of loci in each haplotype on average that are incorrectly recovered, as compared to the original complete haplotype. It can be seen that the error rate varies almost linearly with the masking probability p.

Table 5.5: Results of the IPPH algorithm on 200 × 30 matrices. All values are averages over 100 test runs. For calculation purposes, the algorithm is considered to have *failed* for test runs that took more than 10 seconds.

	$p = 0.1$	$p = 0.2$	$p = 0.3$	$p = 0.4$	$p = 0.5$
No. of test runs completed	97	98	97	95	94
Average time (seconds)	0.08	0.14	0.08	0.24	0.07
Median time (seconds)	0.06	0.06	0.06	0.06	0.05
Percentage of incorrectly recovered loci	0.3	0.65	1.09	1.66	2.46

5.6.2 Results for the IPPH algorithm

Tests were carried out on matrices with 200 genotypes with 30 SNP loci. As in the IPP case, the masking probability p is varied from 0.1 to 0.5. As can be expected, the IPPH problem is considerably harder than IPP problem. Though the algorithm takes less than a second on most instances of the problem, it took more than 15 minutes on a few instances of the problem. The accuracy of the recovered haplotypes was comparable to that for the IPP algorithm. Detailed analysis of the results is shown in Table 5.5.

5.6.3 Discussion

New, faster algorithms for both the IPP an IPPH problems have been presented in this chapter. Through empirical analysis on simulated data, it was demonstrated that these algorithms are very fast and highly accurate. The accuracy of the algorithms even on data with 50% missing entries shows that these algorithms can be used even for input matrices for which a large fraction of the data is missing.

The algorithm for the undirected IPP problem we presented here might be useful in a lot of other applications like building consensus trees. These problems will be investigated in the future.

CHAPTER 6

CONCLUSION

6.1 Block Partitioning Based on Perfect Phylogeny

Different methods have been proposed for block-partitioning of the human genome [ZSW03, DZZ05, ZQL04, ZDC02]. Most of these methods assume that the phased haplotype data is available [ZDC02, ZSW03]. Though some of these methods can deal with genotype data (for instance, [DZZ05]), all these methods involve a two-step process: the haplotypes are first inferred, and then the block partitioning is performed on the haplotypes. The disadvantage of this two-step process is that the haplotype inference procedure does not take into account the block structure of the human genome. Using perfect phylogeny, we can use the block structure itself to infer the haplotypes from the genotype data. Highly accurate block-partitioning can be achieved using this single-step procedure. This section presents an outline of block partitioning based on perfect phylogeny.

The fundamental idea behind block-partitioning based on perfect phylogeny is to divide each chromosome into non-overlapping blocks that admit a perfect or near-perfect phylogeny. The first task in doing so would be to identify all contiguous regions that admit a perfect or near-perfect phylogeny. Each block should be assigned a score based on the desired optimization criteria. A non-overlapping subset of these blocks should then be selected to maximize the overall score, thereby achieving *optimal* block partitioning. Each of these steps is described below in detail.

As in the previous chapters, the input genotype data can be expressed as an $n \times m$ matrix A. The rows in A represent the genotypes and the columns represent the SNPs.

6.1.1 Identification of blocks

In the context of perfect phylogeny based block partitioning, a block is defined as a contiguous region of the chromosome that admits a perfect or a near-perfect phylogeny. Incase of near-perfect phylogeny, the phylogeny should involve no more than ρ number of recombination events and h number of homoplasy events.

A block of length l that begins at a locus i is represented by the tuple $(i, i + l - 1)$. A block $(i, i + l - 1)$ is left-maximal if the block cannot be extended to the left any further. i.e., $(i, i + l - 1)$ is left-maximal if $(i - 1, i + l - 1)$ does not admit a near-perfect phylogeny with parameters ρ and h. Similarly, a block $(i, i + l - 1)$ is right-maximal if it can not be extended to the right any further. A block $(i, i + l - 1)$ is maximal if it is both left-maximal and right-maximal. The first task in obtaining the blocks is to find all the maximal blocks in the data of length at least 2. At any position i, the algorithm starts with a right-maximal block of length at least 1, and tries to extend the block to the left, until it finds a maximal block. At each position i, all the blocks starting at position i with length longer than a certain minimum length are identified.

6.1.2 Block scoring

Each block should be assigned a *score* which takes into consideration the following factors:

1. Number of SNPs covered by the block

2. Actual length of the chromosome (in base pairs) covered by the block

3. The number of distinct haplotypes in the block

4. The number of recombination events and homoplasy events necessary to construct a phylogeny for all the haplotypes in the block

5. The number of tagSNPs necessary to uniquely identify each haplotype in the block

Different optimization criterion will be necessary for different applications. The scoring system should be adjustable in order to accommodate flexible weightage to each of the factors listed above. The scoring system can be represented by W, where $W_{(i,i+l-1)}$ indicates the score of the block $(i, i+l-1)$.

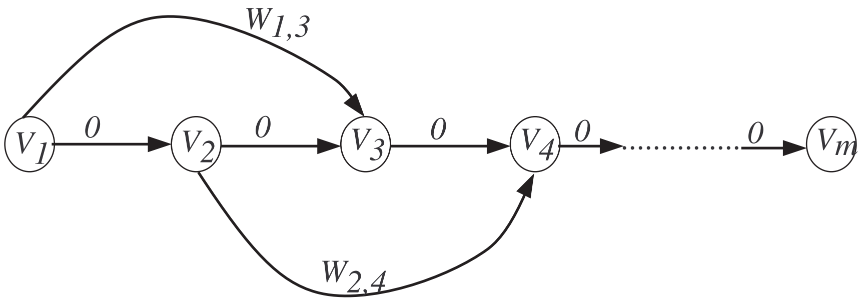


Figure 6.1: The graph $G = (V, E, W)$

6.1.3 Optimal block partitioning

Once a scoring system is in place, the problem of finding the optimal block partitioning reduces to the problem of selecting the set of non-overlapping blocks with the maximum weight. This can be done using the same algorithm as in [VMR03]. The algorithm is briefly described in the following. A directed acyclic graph $G = (V, E, W)$ is constructed. Each vertex $v_i \in V$ represents a locus i. A directed edge $(v_i, v_j) \in E$ if the tuple $(i, i - j)$ is a block. The weight of the edge (v_i, v_j) is the weight of the corresponding block. The edges in the maximum weighted path from the vertex v_1 to v_m give the optimal block-partitioning of the given genotypes. The longest weighted path can be calculated in $O(m^2)$ time. If the size of the longest block is M, the longest weighted path can be calculated in just $O(mM)$ time. Figure 6.1 shows a schematic representation of the graph $G = (V, E, W)$.

6.2 Application to Real Genotype Data

Results from applying perfect phylogeny based methods to real genotype data are encouraging. Eskin et al [HE04]have presented block partitioning results using the perfect phylogeny based HAP program on real genotype data presented in [DRS01]. The data in [DRS01] is from 103 SNPs from a 500 kb region of chromosome 21. The data is from a total of 387 individuals in 129 mother-father-child trios. A significant portion of the data (10.03%) is missing. Eskin et al [HE04] used this data to compare the performance of the HAP program with PHASE [SSD01] and HAPLOTYPER [NQX03], by taking the block-partitioning presented in [DRS01] as the reference. They showed that the accuracy of HAP is comparable to that of PHASE.

Marcini et al [MCP06] presented a comprehensive analysis of the performance of different phasing algorithms on genotype data and haplotype data. Their analysis on simulated data showed that HAP is 1000 times faster than PHASE. However the error rate for HAP was 3.7%, where as the error rate for PHASE was 2.33%. This high error rate is due to the simplistic approach of HAP that constructs only perfect phylogenies. HAP does not explicitly handle imperfect phylogenies. It handles imperfect phylogenies implicitly by ignoring violations of the four-gamete rule as long as they occur with a frequency less than a certain threshold.

More biologically meaningful treatment of the imperfect phylogenies by explicitly allowing homoplasy events and recombinations will only improve the accuracy of perfect phylogeny based methods, and it is quite possible that these methods might achieve better accuracy than all of the existing methods. However some open problems have to be addressed before these imperfect phylogeny based methods on real genotype data, as explained in Section 6.3.

6.3 Future Work

6.3.1 Constructing Phylogenies with Recombination Cycles

An important open problem is that of constructing phylogenies with a limited number of recombinations events. Gusfield et al [GEL03] presented an efficient algorithm for constructing phylogenies with recombination events on a set of haplotypes when certain restrictions apply on how the recombinations can occur. Their algorithm deals with constructing *galled* trees, where the recombination cycles are node-disjoint with each other. Their algorithm for this problem is $O(nm + m^3)$, where n is the number of haplotypes and m is the number SNPs.

It is not yet known if the galled tree construction problem on genotype data is solvable in polynomial time. Song et al [SWG05] posed a much simpler version of the problem - to determine if the given set of genotypes admit a phylogeny with a single recombination cycle. Even this simple version of the problem is an open problem.

Recombination events events are very common in the human genome. In many cases, deviations from perfect phylogeny are likely to be due to recombination events. Hence, solving this problem will be essential in improving the accuracy of the perfect phylogeny based haplotype inference methods.

6.3.2 Constructing Imperfect Phylogenies on Incomplete Data

Finally, a strategy is necessary for constructing imperfect phylogenies on incomplete genotype data. Current high throughput genotyping methods produce data that is highly accurate and complete. It was reported [Int05] that the HapMap phase-I data is 99.7% accurate and 99.3% complete. The 0.7% missing data is still significant and the imperfect phylogeny construction algorithms should be modified to effectively handle this missing data

6.3.3 Incorporating Statistical Methods

Even with the ability to construct imperfect phylogenies, perfect phylogeny based methods may still fail in some regions of the human genome. Using statistical algorithms in these regions might be necessary to achieve better accuracy.

On the other hand, statistical methods might incorporate phylogeny based methods for improving their speed and accuracy. Statistical methods might use combinatorial approaches to quickly arrive at a *starting point*, and improve upon the results through further statistical analysis.

6.4 Conclusion

Efficient algorithms for haplotype inference based on perfect phylogeny have been developed as part of this dissertation work. Analysis on simulated data shows that these algorithms are fast and highly accurate. These algorithms can be used for block partitioning of the human genome.

The algorithms presented here are also applicable for general phylogeny construction problems. Specifically, these near-perfect phylogeny construction algorithms presented in Chapter 4 will be useful for constructing phylogenies on mtDNA and nrY SNP data. It is possible to extend these algorithms to apply on multi allelic character data like the nrY microsatellite data.

GLOSSARY

Some of the following definitions have been taken from online glossary [Dav06].

Linkage Disequilibrium

Linkage disequilibrium is the non-random association between two or more characters. A set of loci are said to be in linkage disequilibrium if the observed frequency distribution of the haplotypes over the given loci is different from the frequency distribution expected from the individual allele frequencies at each locus.

Consider two bi-allelic SNP loci A and B, with the alleles (A_1,A_2) and (B_1,B_2) respectively. Let the observed frequencies of the alleles A_1 and A_2 at locus A be p and $1 - p$, and the observed frequencies of the alleles B_1 and B_2 at locus B be q and $1 - q$, respectively. Let the frequencies of the four haplotypes over the two loci (A_1B_1, A_1B_2, A_2B_1, A_2B_2) be represented by (f_{11}, f_{12}, f_{21}, f_{22}). Clearly, the expected values for (f_{11}, f_{12}, f_{21}, f_{22}) are (pq, $p(1 - q)$, $(1 - p)q$, $(1 - p)(1 - q)$). The two loci are said to be in linkage *equilibrium* if the observed frequencies of the haplotypes match these expected frequencies. The two loci are in linkage disequilibrium otherwise.

There are various measures for linkage disequilibrium. Linkage disequilibrium measure D is given by $D = (f_{11}f_{22} - f_{12}f_{21})$. It can be shown [LK60] that any observed set of frequencies can be expressed in terms of D as $f_{11} = pq + D$, $f_{12} = p(1-q) - D$, $f_{21} = (1-p)q - D$ and $f_{22} = (1-p)(1-q) + D$. At linkage equilibrium, D will be equal to zero.

The measure D is sensitive to allele frequencies, so the normalized measure $D' = \frac{D}{D_{\text{max}}}$ can be used instead, where D_{max} is the theoretical maximum value of D.

Another commonly used measure for linkage disequilibrium is r^2, given by $r^2 = \frac{D^2}{pq(1-p)(1-q)}$.

Microsatellites

A microsatellite consists of tandem repeats of a specific short sequence (2-5 bp) of DNA. A microsatellite marker can be expressed as $(\text{P})_n$, where is P is a DNA sequence of length 3-5 bp, and n is the repeat count. The repeat count varies from individual to individual.

Minimum Allele Frequency, MAF

The frequency of the second most frequent allele in a SNP location.

Restriction Factor Length Polymorphisms (RFLPs)

Variation within the DNA sequences of organisms of a given species that can be identified by fragmenting the sequences using restriction enzymes. Variations in the population result in variations in the lengths of the fragments produced by a set of restriction enzymes. RFLPs can be used to measure the diversity of a gene in a population.

LIST OF REFERENCES

[AF94] Richa Agarwala and David Fernandez-Baca. "A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed." *SIAM J. Computing*, **23**:1216–1224, 1994.

[AJL03] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. *Molecular Biology of the Cell.* Garland Science, NewYork, NY, 4 edition, 2003.

[BB04] Vineet Bafna and Vikas Bansal. "The number of recombination events in a sample history: conflict graph and lower bounds." *IEEE Trans on Comput Biol and Bioinform*, **1**(2):78–90, 2004.

[BFW92] H. Bodlaender, M. Fellows, and T. Warnow. "Two strikes against perfect phylogeny." In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, pp. 273–283. Springer Verlag, Lecture Notes in Computer Science, 1992.

[BGH04] Vineet Bafna, Dan Gusfield, Sridhar Hannenhalli, and Shibu Yooseph. "A note on efficient computation of haplotypes via perfect phylogeny." *J Comput Biol.*, **11**(5):858–66, 2004.

[BGL02] Vineet Bafna, Dan Gusfield, Giuseppe Lancia, and Shibu Yooseph. "Haplotyping as Perfect Phylogeny: A direct approach." Technical Report CSE-2002-21, Department of Computer Science, The University of California at Davis, July 2002.

[BGL03] V Bafna, D Gusfield, G Lancia, and S Yooseph. "Haplotyping as perfect phylogeny: a direct approach." *J Comput Biol.*, **10**(3–4):323–340, 2003.

[CG02] Ren Hua Chung and Dan Gusfield. "PPH - A program for deducing haplotypes that fit a perfect phylogeny." Technical Report CSE-2002-27, Department of Computer Science, The University of California at Davis, 2002.

[Cla90] Andrew G. Clark. "Inference of Haplotypes from PCR-amplified Samples of Diploid Populations." *Mol. Biol. Evol.*, **7**:111–122, 1990.

[Cla04] AG Clark. "The role of haplotypes in candidate gene studies." *Genet Epidemiol.*, **27**:321–333, 2004.

[CR06] Alicia Chang and Malcolm Ritter. "Cousins at Risk of Cancer Give Up Stomachs." `http://www.washingtonpost.com/wp-dyn/content/article/2006/06/18/AR2006061800251.html`, June 18, 2006.

[Dav06] Richard E. Davis. "Bioinformatics Glossary." `http://www.library.csi.cuny.edu/~davis/Bio_326/bioinfo_glossary.html`, 2006.

[DBG01] JA Douglas, M Boehnke, E Gillanders, JM Trent, and SB Gruber. "Experimentally-derived haplotypes substantially increase the efficiency of linkage disequilibrium studies." *Nature Genetics*, **28**:361–364, 2001.

[DJS86] W.H.E Day, D.S. Johnson, and D. Sanko. "The computational complexity of inferring rooted phylogenies by parsimony." *Mathematical Biosciences*, **81**:33–42, 1986.

[DRS01] Mark J. Daly, John D. Rioux, Stephen F. Schaffner, Thomas J. Hudson, and Eric S. Lander. "High-resolution haplotype structure in the human genome." *Nature Genetics*, **29**(2):229–32, Oct 2001.

[DZZ05] K Ding, K Zhou, J Zhang, J Knight, X Zhang, and Shen Y. "The effect of haplotype-block definitions on inference of haplotype-block structure and htSNPs selection." *Mol. Biol. Evol.*, **22**(1):148–59, 2005.

[EHK03] Eleazar Eskin, Eran Halperin, and Richard M. Karp. "Large Scale Reconstruction of Haplotypes from Genotype Data." In *Proceedings of RECOMB*, 2003.

[Fel04] Joseph Felsenstein. *Inferring Phylogenies.* Sinauer Associates, Sunderland, MA, USA, 2004.

[Fit71] Walter M. Fitch. "Toward defining the course of evolution: Minimum change for a specified tree topology." *Systematic Zoology*, **20**:406–416, 1971.

[FL03] David Fernandez-Baca and Jens Lagergren. "A polynomial time algorithm for near-perfect-phylogeny." *SIAM Journal of Computing*, **32**(5):1115–1127, 2003.

[GEL03] Dan Gusfield, Satish Eddhu, and Charles Langley. "Efficient reconstruction of phylogenetic networks with constrained recombination." In *Proceedings of CSB2003*, pp. 363–374, Stanford, CA, August 2003.

[GH04] D Gusfield and D Hickerson. "A fundamnetal, efficiently-computed lower bound on the number of recombinations needed in phylogenetic networks." Technical report, University of California at Davis, 2004.

[GNS04] Jens Gramm, Till Nierhoff, Roded Sharan, and Till Tantau. "Haplotyping with Missing Data via Perfect Path Phylogenies." In *Proceedings of the second RECOMB datellite workshop on Computational methods for SNPs and haplotypes*, pp. 35–46, 2004.

[GSN02] SB Gabriel, SF Schaffner, H Nguyen, JM Moore, J Roy, B Blumensteil, J Higgins, M DeFelice, A Lochner, M Faggart, SN Liu Cordero, C Rotimi, A Adeyemo, R Cooper, R Ward, ES Lander, MJ Daly, and D Altshuler. "The structure of haplotype blocks in human genome." *Science*, **296**:2225–2229, 2002.

[Gus91] Dan Gusfield. "Efficient Algorithms for Inferring Evolutionary Trees." *Networks*, **21**:19–28, 1991.

[Gus97] Dan Gusfield. *Algorithms on strings, trees and sequences.* The Press Syndicate of Univeristy of Cambridge, NewYork, USA, 1997.

[Gus01] Dan Gusfield. "Inference of haplotypes from samples of diploid populations: Complexity and algorithms." *J Comput Biol.*, **8**(3):305–323, 2001.

[Gus02] Dan Gusfield. "Haplotyping as Perfect Phylogeny: conceptual framework and efficient solutions." In *Proceedings of RECOMB*, 2002.

[GW02] David B. Goldstein and Michael E. Weale. "Poplation genomics: Linkage Disequilibrium holds the key." *Cur Biol*, **11**:576–579, 2002.

[Hap03] HapMapConsortium. "The International HapMap Project." *Nature*, **426**:789–796, December 2003.

[HE04] Eran Halperin and Eleazar Eskin. "Haplotype reconstruction from genotype data using Imperfect Phylogeny." *Bioinformatics*, **20**(12):1842–1849, 2004.

[HK85] R Hudson and N Kaplan. "Statistical propertied of the number of recombination events in the history of a sample of DNA sequences." *Genetics*, **111**:147–165, 1985.

[HK04] Eran Halperin and Richard Karp. "Perfect Phylogeny and Haplotype Assignment." In *Proceedings of RECOMB*, 2004.

[HSN05] DA Hinds, LL Stuve, GB Nilsen, E Halperin, E Eskin, DG Ballinger, KA Frazer, and DR Cox. "Whole-genome patterns of common DNA variation in three human populations." *Science*, **307**:1072–9, 2005.

[Hud01] R Hudson. *Handbook of statistical genetics*, chapter Linkage disequilibrium and recombination, pp. 309–324. Wiley & Sons, New York, 2001.

[Hud02] R. Hudson. "Generating samples under the Wright-Fisher neutral model of genetic variation." *Bioinformatics*, **18**:337–338, 2002.

[Int05] InternationalHapMapConsortium. "A haplotype map of the human genome." *science*, **437**:1299–1320, 2005.

[KS05] Gad Kimmel and Ron Shamir. "The Incomplete Perfect Phylogeny Problem." *J Bioinform Comput Biol.*, **3**(2):1–25, 2005.

[KW97] Sampath Kannan and Tandy Warnow. "A Fast Algorithm for the Computation and Enumeration of Perfect Phylogenies when the Number of Character States is Fixed." *SIAM J. Computing*, **26**:1749–1763, 1997.

[Law06] Lawrence Berkeley National Laboratory. "All about breast cancer genes." `http://www.lbl.gov/Education/ELSI/Frames/cancer-genes-f.html`, 2006.

[LJ03] Jing Li and Tao Jiang. "Efficient inference of haplotypes from genotypes on a pedigree." *J Bioinform Comput Biol.*, **1**(1):41–69, 2003.

[LK60] R. C. Lewontin and Ken ichi Kojima. "The evolutionary dynamics of complex polymorphisms." *Evolution*, **42**:458–472, 1960.

[LZ01] K Zhang L Wang and L Zhang. "Perfect phylogenetic networks with recombinations." *J Comput Biol.*, **8**:69–78, 2001.

[LZH04] E LindHolm, J Zhang, SE Hodge, and DA Greenberg. "The realizability of haplotyping inference in nuclear families: misassignment rates for SNPs and microsatellites." *Hum Hered.*, **57**(3):117–27, 2004.

[MCP06] Jonathan Marchini, David Cutler, Nick Patterson, Matthew Stephens, Eleazar Eskin, Eran Halperin, Shin Lin, Zhaohui S. Qin, Heather M. Munro, Goncalo R. Abecasis, and Peter Donnelly. "A comparison of phasing algorithms for trios and unrelated individuals." *Am J Hum Genet.*, **78**:437–450, 2006.

[MKE02] OG McDonald, EY Krynetski, and WE Evans. "Molecular haplotyping of genomic DNA for multiple single-nucleotide polymorphisms located kilobases apart using long-range polymerase chain reaction and intramolecular ligation." *Pharmacogenetics*, **12**:93–99, 2002.

[MTB96] S Michalatos-Beloin, SA Tishkoff, KL Bentley, and KK Kidd G Ruano G. "Molecular haplotyping of genetic markers 10 kb apart by allele-specific long-range PCR." *Necleic Acids Res.*, **24**:4841–4843, 1996.

[Nat06] National Institutes of Health. "NHLBI TO LAUNCH FRAMINGHAM GENETIC RESEARCH STUDY." `http://www.nhlbi.nih.gov/new/press/06-02-06.htm`, February 6, 2006.

[Niu04] T Niu. "Algorithms for inferring haplotypes." *Genet Epidemiol.*, **27**(4):334–347, 2004.

[NQX03] Tianhua Niu, Zhaohui S Qin, Xiping Xu, and Jun S Liu. "Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms." *Am J Hum Genet.*, **70**:157–169, 2003.

[OHE02] J Odeberg, K Holmberg, P Eriksson, and M Uhlen. “Molecular haplotyping by pyrosequencing.” *Biotechniques*, **33**:1104–1108, 2002.

[pau06] “PAUP.” `http://paup.csit.fsu.edu/`, 2006.

[PBH03] N Patil, AJ Berno, DA Hinds, WA Barrett, JM Doshi, CR Hacker, CR Kautzer, DH Lee, C Marjoribanks, DP McDonough, BT Nguyen, MC Norris, JB Sheehan, N Shen, D Stern, RP Stokowski, DJ Thomas, MO Trulson, KR Vyas, KA Frazer, SP Fodor, and DR Cox. “Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21.” *Science*, **294**:1719–1723, 2003.

[phy06] “PHYLIP.” `http://evolution.genetics.washington.edu/phylip.html`, 2006.

[PPS04] Itsik Peer, Tal Pupko, Ron Shamir, and Roded Sharan. “Incomplete Directed Perfect Phylogeny.” *SIAM Journal on Computing*, **33**(3):590–607, 2004.

[QB02] D Qian and L Beckmann. “Minimum-recombinant haplotyping in pedigrees.” *Am J Hum Genet.*, **70**(6):1434–1445, 2002.

[RCB01] David E. Reich, Michele Cargill, Stacey Bolk, James Ireland, Pardis C. Sabeti, Daniel J. Richter, thomas Lavery, Rose Kouyoumjian, Shelli F. Farhadian, Ryk Ward, and Eric S. Lander. “Linkage Disequilibrium in the human genome.” *nature*, **411**:199–204, 2001.

[San75] David Sankoff. “Minimal mutation trees of sequences.” *SIAM Journal of Applied Mathematics*, **28**:35–42, 1975.

[SDB05] Srinath Sridhar, Kedar Dhamdhere, Guy E. Blelloch, Eran Halperin, R. Ravi, and Russell Schwartz. “FPT Algorithms for Binary Near-Perfect Phylogenetic Trees.” Technical Report CMU-CS-05-181, Computer Science Department, Carnegie Mellon University, School, September 2005.

[SM97] Joao Setubal and Joao Medanis. *Introduction to Computational Molecular Biology*. PWS Publishing Company, Boston, MA, 1997.

[SR75] David Sankoff and Pascale Rousseau. “Locating the vertices of a Steiner tree in arbitrary space.” *Mathematical Programming*, **9**:240–246, 1975.

[SRR90] JC Stephens, J Rogers, and G Ruano. “Theoretical underpinning of the single-molecule-dilution (SMD) method of direct haplotype resolution.” *Am J Hum Genet.*, **46**(6):1149–1155, 1990.

[SSD01] M. Stephens, N.J. Smith, and P. Donnelly. “A new statistical method for haplotype reconstruction from population data.” *Am J Hum Genet.*, **68**:978–989, 2001.

[Ste92] Michael Steel. “The Complexity of Reconstructing Trees from Qualitative Characters and Subtrees.” *Journal of Classification*, **9**:91–116, 1992.

[SWG05] Yun S. Song, Yufeng Wu, and Dan Gusfield. “Algorithms for Imperfect Phylogeny Haplotyping (IPPH) with a single Homoplasy or Recombination Event.” In *Proceedings of WABI 2005*, pp. 152–164, 2005.

[VM05] Ravi Vijaya Satya and Amar Mukherjee. “An Efficient algorithm for perfect phylogeny haplotyping.” In *Proceedings of CSB2005*, pp. 103–110, Stanford, CA, August 2005.

[VM06] Ravi Vijaya Satya and Amar Mukherjee. “An optimal algorithm for perfect phylogeny haplotyping.” *Journal of Computational Biology*, **13**(4):897–928, 2006.

[VMR03] Ravi Vijaya Satya, Amar Mukherjee, and Uday Kumar Ranga. “A pattern matching algorithm for codon optimization and CpG motif-engineering.” In *CSB 2003*, pp. 294–305, 2003.

[WGC00] AT Woolley, C Guillemette, C Li Cheung, DE Housman, and CM Lieber. “Direct haplotyping of kilobase-size DNA using carbon nanotube probes.” *Nat Biotechnol*, **18**:760–763, 2000.

[Wiu04] Carsten Wiuf. “Inference on recombination and block structure using unphased data.” *Genetics*, **166**(1):537–545, 2004.

[WP03] JD Wall and JK Pritchard. “Haplotype blocks and linkage disequilibrium in the human genome.” *Nat Rev Genet.*, **4**(8):587–597, 2003.

[ZDC02] K Zhang, M Deng, T Chen, MS Waterman, and FA Sun. “A dynamic programming algorithm for haplotype block partitioning.” *Proceedings of the National Academy of Science of United States of America*, **99**:7335–7339, 2002.

[ZLH01] XB Zhong, Pm Lizardi, XH Huang, PL Bray-Ward, and DC Ward. “Visualization of oligonucleotide probes and point mutations in interphase nuclei and DNA fibers using rolling circle DNA amplification.” *Proc Natl Acad Sci USA*, **98**:3940–3945, 2001.

[ZQL04] K Zhang, ZS Qin, JS Liu, T Chen, MS Waterman, and F Sun. “Haplotype block partitioning and tag SNP selection using genotype data and their applications to association studies.” *Genome Res.*, **14**(5):908–16, 2004.

[ZSW03] Kui Zhang, Fengzhu Sun, Michael S. Waterman, and Ting Chen. “Dynamic programming algorithms for haplotype block partitioning: Applications to human chromosome 21 haplotype data.” In *RECOMB 2003*, Berlin, Germany, 2003.