Fast Elimination of Redundant Linear Equations and Reconstruction of Recombination-Free Mendelian Inheritance on a Pedigree

Jing Xiao[1], Lan Liu[2], Lirong Xia[1], Tao Jiang[2,3]
[1] Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2] Department of Computer Science and Engineering, University of California, Riverside, CA
[3] Currently visiting at Tsinghua University, Beijing, China

Abstract

Computational inference of haplotypes from genotypes has attracted a great deal of attention in the computational biology community recently, partially driven by the international HapMap project. In this paper, we study the question of how to efficiently infer haplotypes from genotypes of individuals related by a pedigree, assuming that the hereditary process was free of mutations (*i.e.* the Mendelian law of inheritance) and recombinants. The problem has recently been formulated as a system of linear equations over the finite field of $F(2)$ and solved in $O(m^3n^3)$ time by using standard Gaussian elimination, where m is the number of loci (or markers) in a genotype and n the number of individuals in the pedigree. We give a much faster algorithm with running time $O(mn^2 + n^3 \log^2 n \log\log n)$. The key ingredients of our construction are (i) a new system of linear equations based on some spanning tree of the pedigree graph and (ii) an efficient method for eliminating redundant equations in a system of $O(mn)$ linear equations over $O(n)$ variables. Although such a fast elimination method is not known for general systems of linear equations, we take advantage of the underlying pedigree graph structure and recent progress on low-stretch spanning trees.

Keywords: haplotype inference, pedigree analysis, system of linear equations, low-stretch spanning tree

1 Introduction

For centuries, human beings have fought the battle against deadly diseases, such as diabetes, cancer, stroke, heart disease, depression, and asthma. Genetic factors are believed to play a significant role for preventing, diagnosing, and treating these diseases. In recent years, *gene mapping*, whose goal is to establish connections between diseases and some specific genetic variations, has become one of the most active areas of research in human genetics. In October 2002, a multi-country collaboration, namely, the international *HapMap* project was launched [18]. One of the main objectives of the HapMap project is to identify the *haplotype* (*i.e.*, the states of genetic markers from a single chromosome) structure of humans and common haplotypes among various populations. This information will greatly facilitate the mapping of many important disease-susceptible genes. However, the human genome is a *diploid* (*i.e.*, its chromosomes come in pairs with one being paternal and the other maternal) and, in practice, haplotype data are not collected directly, especially in large-scale sequencing projects mainly due to cost considerations. Instead, *genotype* data (*i.e.*, the states of genetic markers from all chromosomes, without specifying which chromosome giving rise to each particular marker state) are collected routinely. Hence, combinatorial algorithms and statistical methods for the inference of haplotypes from genotypes, which is also commonly referred to as *phasing*, are urgently needed and have been intensively studied.

This paper is concerned with the inference of haplotypes from genotypes of individuals related by a *pedigree*, which describes the parent-offspring relationship among the individuals. Figure 1 gives an illustrative example of pedigree, genotype, haplotype, as well as *recombination* where the haplotypes of a parent recombine to produce a haplotype of her child. (See the appendix for more detailed definitions of these concepts.) Pedigree data is often collected in family-based gene association/mapping studies in addition to genotype data. It is generally believed that haplotypes inferred from pedigrees are more accurate than those from population data. Moreover, some family-based statistical gene association tests such as TDT (*i.e. Transmission Disequilibrium Test*) and its variants (*e.g.* [30, 37] among others) require access to haplotype information of each member in a pedigree.

By utilizing some reasonable biological assumptions, such as the *Mendelian law of inheritance*, *i.e.* one haplotype of each child is inherited from the father while the other is inherited from the mother free of mutations, and the *minimum-recombination principle* which says that genetic recombination is rare for closely linked markers and thus haplotypes with fewer recombinants should be preferred in haplotype inference [28, 29], several combinatorial approaches for inferring haplotypes from genotypes on a pedigree have been proposed recently and shown to be powerful and practical [28, 21, 22, 23, 29, 34, 35]. These methods essentially propose polynomial-time heuristics or exponential-time exact algorithms for the so called the *minimum-recombinant haplotype configuration (MRHC)* problem, which requires a haplotype solution for the input pedigree with the minimum number of recombinants (*i.e.* recombination events) and is known to be NP-hard [21]. (See the appendix for a more formal definition of the MRHC problem).

A closely related problem, called the *zero-recombinant haplotype configuration (ZRHC)* problem where we would like to enumerate all haplotype solutions that require no recombinant (if such solutions exist), was studied in [21] under a more stringent biological assumption that the pedigree is also recombination-free. (See the appendix for a more formal definition of the ZRHC problem). ZRHC is interesting because recent genetic research has shown that human genomic DNAs can be partitioned into long blocks (called *haplotype blocks*) such that recombination within each block is rare or even nonexistent [6, 10, 19], especially when restricted to a single pedigree [22, 23]. An efficient algorithm for ZRHC could also be useful for solving the general MRHC problem as a subroutine, when the number of recombinants is expected to be small. We note in passing that recent work on haplotype inference for population data based on perfect phylogenies also assumes the data is recombination-free [11, 12]. Observe that, when the solution for ZRHC is not unique, it would really be useful to be able to enumerate all the solutions instead of finding only one feasible solution, so that the solutions can be examined in subsequent analysis (*e.g.* likelihood distribution of haplotypes [22, 23], linkage between different haplotype blocks, etc.) by geneticists.

The Algorithmic Problem and Our Result. The ZRHC problem can also be stated abstractly as a simple inheritance reconstruction problem as follows. We have a pedigree connecting n individuals where each individual j has two haplotypes (*i.e.* strings) defined on m marker loci $a_{j,1} \cdots a_{j,m}$ and $b_{j,1} \cdots b_{j,m}$ inherited from j's father and mother, respectively. That is, if the parents of j are individuals j_1 and j_2, then $a_{j,1} \cdots a_{j,m} \in \{a_{j_1,1} \cdots a_{j_1,m}, b_{j_1,1} \cdots b_{j_1,m}\}$ and $b_{j,1} \cdots b_{j,m} \in \{a_{j_2,1} \cdots a_{j_2,m}, b_{j_2,1} \cdots b_{j_2,m}\}$. The haplotypes are unknown, but the genotype of each individual j is given to us in the form of string $\{a_{j,1}, b_{j,1}\} \cdots \{a_{j,m}, b_{j,m}\}$.

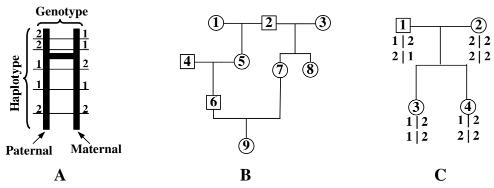

Figure 1: A. The structure of a pair of chromosomes from a mathematical point of view. In the figure, each numeric value (1 or 2) represents a marker state or an *allele*. The haplotype inherited from the father (or the mother) is called *paternal haplotype* (or *maternal haplotype*, respectively). The paternal and maternal haplotypes are thus strings 22112 and 11212, and they form the genotype $\{1,2\}\{1,2\}\{1,2\}\{1,1\}\{2,2\}$, which is a string of unordered pairs of alleles at each locus. B. An illustration of a pedigree with 9 members with a *mating loop*, where circles represent females and boxes represent males. Children are shown under their parents with line connections. For example, individuals 7 and 8 are children of individuals 2 and 3. Individuals without parents, such as individuals 1 and 2 are called *founders*. A pedigree with no mating loops are called *tree pedigree* conventionally. C. An example of recombination event where the haplotypes of individual 1 recombine to produce the paternal haplotype of individual 3. The numbers inside the circles/boxes are individual IDs. Here, a "|" is used to indicate the phase of the two alleles at a marker locus, with the left allele being paternal and the right maternal. Both loci of individual 2 and the 2nd locus of individual 4 are *homozygous* while all the other loci in the pedigree are *heterozygous*.

We would like to reconstruct all the haplotype solutions that could have resulted in the genotypes.

Li and Jiang presented an $O(m^3n^3)$ time algorithm for ZRHC by formulating it as a system of $O(mn)$ linear equations with mn variables over the finite field of $F(2)$ and applying Gaussian elimination [21]. Although this cubic time algorithm is reasonably fast, it is inadequate for large scale pedigree analysis where both m and n can be in the order of tens or even hundreds, and we may have to examine many pedigrees and haplotype blocks. There are, for example, over five million SNP markers in the public database dbSNP [18]. This challenge motivates us to find more efficient algorithms for ZRHC. Several attempts have been made recently in [3, 25], but the authors failed to prove the correctness of their algorithms in all cases, especially when the input pedigree has mating loops. Chan *et al.* proposed a linear-time algorithm in [2], but the algorithm only works for pedigrees without mating loops (*i.e.* the tree pedigrees).

In this paper, we present a much faster algorithm for ZRHC with running time $O(mn^2+n^3\log^2 n\log\log n)$. Our construction begins with a new system of linear equations over $F(2)$ for ZRHC. Although the system still has $O(mn)$ variables and $O(mn)$ equations, we prove that it can be reduced to an effectively equivalent system with $O(mn)$ equations and at most $2n$ variables, by exploring the underlying pedigree graph structure. By using standard Gaussian elimination, this already gives an improved algorithm with running time $O(mn^3)$. We then show how to reduce the number of equations further to $O(n\log^2 n\log\log n)$ (assuming that $m \geq \log^2 n\log\log n$, which usually holds in practice), by giving an $O(mn)$ time method for eliminating redundant equations in the system. Although such a fast elimination method is not known for general systems of linear equations, we again take advantage of the underlying pedigree graph structure and recent progress on low-stretch spanning trees in [8]. In particular, the low-stretch spanning tree result helps upper bound the number of equations that need be kept in the elimination process. We also show that our algorithm actually runs in time $O(mn^2+n^3)$ time when the input pedigree is a tree pedigree with no mating loops (which is often true for human pedigrees) or when there is a locus that is heterozygous across the entire pedigree. Moreover, our algorithm produces a general solution [1] to the original system of linear equations at the end that represents all feasible solutions to the ZRHC problem.

Related Work on Solving Systems of (Sparse) Linear Equations. The search for efficient algorithms for solving systems of linear equations is a classical problem in linear algebra. Besides Gaussian elimination, methods based on fast matrix multiplication algorithms have been proposed and could achieve an asymptotic speed of $O(n^{2.376})$ on n equations with n unknowns [7, 33]. However, these methods are only of theoretical interest since they are hard to implement and do not outperform Gaussian elimination unless n is very large. Moreover, they assume that the coefficient matrix is of full rank, which is an unreasonable assumption in ZRHC (considering the linear systems derived for ZRHC so far).

[1] A general solution of any linear system is denoted by the span of a basis in the solution space to its associated homogeneous system, offset from the origin by a vector, namely by any particular solution.

Observe that the linear system given in [21] for ZRHC is actually very sparse since each of its equations has at most four variables. Thus, a plausible way to speed-up is to utilize fast algorithms for solving sparse linear systems. The Lanczos and conjugate gradient algorithms [15], and the Wiedemann algorithm [36] are some of the best known algorithms for solving sparse linear system over finite fields. The Wiedemann algorithm runs in (expected) quadratic time (which is in fact slower than our algorithm when applied to linear systems for ZRHC) while the Lanczos and conjugate gradient algorithms are only heuristics [27]. However, they use randomization and do not find all solutions. Furthermore, the algorithms cannot check if the system has no solution [14]. A randomized algorithm with quadratic expected time for certifying inconsistency of linear systems is given in [14].

The rest of our paper is organized as follows. We will describe a new system of linear equations for ZRHC and some useful graphs derived from a pedigree in section 2. The $O(mn^3)$ time algorithm is presented in section 3, and the $O(mn^2 + n^3 \log^2 \log \log n)$ time algorithm is given in section 4. Some concluding remarks are given in section 5. Due to page constraint, we omit the proofs of some theorems, lemmas and corollaries. (See the appendix for the details of the proofs).

2 A System of Linear Equations for ZRHC and the Pedigree Graph

In this section, we first present a new formulation of ZRHC in terms of linear equations, and define some graph structures which will be used in our algorithm.

2.1 The Linear System

Throughout this paper, n denotes the number of the individuals (or members) in the input pedigree and m the number of marker loci. Without loss of generality, suppose that each allele in the given genotypes is numbered numerically as 1 or 2 (*i.e.* the markers are assumed to be *bi-allelic*, which make the hardest case for MRHC/ZRHC[21]), and the pedigree is free of genotype errors (*i.e.* the two alleles at each locus of a child can always be obtained from her respective parents). Hence, we can represent the genotype of member j as a ternary vector $\mathbf{g}_j$ as follows: $g_j[i] = 0$ if locus i of member j is homozygous with both alleles being 1's, $g_j[i] = 1$ if the locus is homozygous with both alleles being 2's, and $g_j[i] = 2$ otherwise (*i.e.* the locus is heterozygous). For any heterozygous locus i of member j, we use a binary variable $p_j[i]$ to denote the *phase* at the locus as follows: $p_j[i] = 0$ if allele 1 is paternal, and $p_j[i] = 1$ otherwise. When the locus is homozygous, the variable is set to $g_j[i]$ for some technical reasons (so that the equations below involving $p_j[i]$ will hold). Hence, the vector $\mathbf{p}_j$ describes the paternal and maternal haplotypes of member j. Observe that the vectors $\mathbf{p}_1, \ldots, \mathbf{p}_n$ represent a complete haplotype configuration of the pedigree. In fact, the sparse linear system in [21] was based on these vectors. Also for technical reasons, define a vector $\mathbf{w}_j$ for member j such that $w_j[i] = 0$ if its i-th locus is homozygous and $w_j[i] = 1$ otherwise.

Suppose that member j_r is a parent of member j. We introduce an auxiliary binary variable $h_{j_r,j}$ to indicate which haplotype of j_r is passed to j. If j_r gives its paternal haplotype to j, then $h_{j_r,j} = 0$; otherwise $h_{j_r,j} = 1$. Suppose that j is a non-founder member with her father and mother being j_1 and j_2, respectively. We can define two linear (constraint) equations over $F(2)$ to describe the inheritance of paternal and maternal haplotypes at j respectively, following the Mendelian law of inheritance and zero-recombinant assumption:

$$\mathbf{p}_{j_1} + h_{j_1,j} \cdot \mathbf{w}_{j_1} = \mathbf{p}_j \qquad \text{and} \qquad \mathbf{p}_{j_2} + h_{j_2,j} \cdot \mathbf{w}_{j_2} = \mathbf{p}_j + \mathbf{w}_j \tag{1}$$

If we let $\mathbf{d}_{j_1,j}$ be the vector $\mathbf{0}$ and $\mathbf{d}_{j_2,j} = \mathbf{w}_j$ otherwise, then the above equations can be unified into a single equation as:

$$\mathbf{p}_{j_r} + h_{j_r,j} \cdot \mathbf{w}_{j_r} = \mathbf{p}_j + \mathbf{d}_{j_r,j} \qquad (r = 1,\ 2) \tag{2}$$

Formally, we can express the ZRHC problem as a system of linear equations:

$$\begin{cases} p_k[i] + h_{k,j} \cdot w_k[i] = p_j[i] + d_{k,j}[i] & 1 \le i \le m,\ 1 \le j, k \le n, k \text{ is a parent of } j \\ p_j[i] = g_j[i] & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] \ne 2 \\ w_j[i] = 1 & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] = 2 \\ w_j[i] = 0 & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] \ne 2 \\ d_{k,j}[i] = w_j[i] & 1 \le i \le m,\ 1 \le j, k \le n,\ k \text{ is the mother of } j \\ d_{k,j}[i] = 0 & 1 \le i \le m,\ 1 \le j, k \le n,\ k \text{ is the father of } j \end{cases} \tag{3}$$

where $g_j[i], w_j[i], d_{k,j}[i]$ are all constants depending on the input genotypes, and $p_j[i], h_{k,j}$ are the unknowns. Note that, the number of p-variables is exactly mn and the number of g-variables is at most $2n$ since every child has two parents and there are at most n children in the pedigree.

Remark: Observe that for any member j, if the member itself or one of its parents are homozygous at locus i, $p_j[i]$ is fixed based on Equation (3). In the rest of the paper, we will assume that all such variables $p_j[i]$ are *pre-determined* (without any conflict), and use them as "anchor points" to define some new constraints about the h-variables.

2.2 The Pedigree Graph and Locus Graphs

To conform with standard graph theory notations, we transform the input pedigree into a graph, called the *pedigree graph*, by connecting each parent directly to her children, as shown in Figure 2 (B). Although the edges in the pedigree represent the inheritance relationship between a parent and a child and are directed, we will think of the pedigree graph, and more importantly, the subsequent locus graphs, as undirected in future definitions and constructions, since each edge (j, k) of the pedigree graph (and locus graphs) will be used to represent the constraint between the vectors $\mathbf{p}_j$ and $\mathbf{p}_k$ (*i.e.* the phases at j and k) via the variable $h_{j,k}$, which is symmetric as can be seen from Lemma 1 and Corollaries 2 and 3 below. [2]

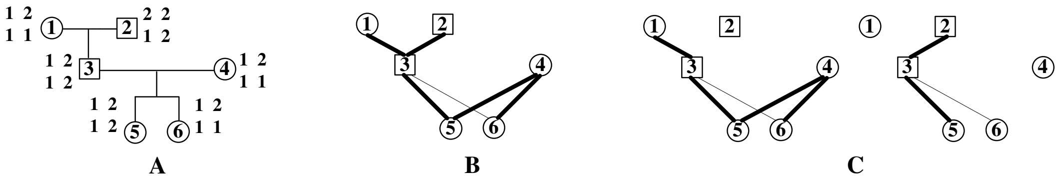

Figure 2: A. An example pedigree with genotype data. Here, the alleles at a locus are ordered according to their id numbers instead of phase (which is unknown). B. The pedigree graph with a spanning tree. The tree edges are highlighted. Observe that there lies a *cycle* of length 4 in the given tree pedigree graph. C. The locus graphs. The left graph is for the first locus, which has a cycle, while the right graph is for the second locus. The locus forests are highlighted.

Clearly, such a pedigree graph $G = (V, E)$ may be cyclic due to mating loops or multiple children shared by a pair of parents. Let $\mathcal{T}(G)$ be any spanning tree of G. $\mathcal{T}(G)$ partitions the edge set E into two subsets: *tree edges* and *non-tree edges*. For simplicity, the non-tree edges will be called *cross edges*. Let $E^{\mathbb{X}}$ denote the set of cross edges. Since $|E| \le 2n$ and the number of edges in $\mathcal{T}(G)$ is $n-1$, we have $|E^{\mathbb{X}}| \le n+1$. Figure 2 (B) gives an example of tree edges and cross edges.

For any fixed locus i, the value $w_k[i]$ can be viewed as the weight of each edge $(k, j) \in E$, where k is a parent of j. We construct the *i-th locus graph* G_i as the subgraph of G induced by the edges with weight 1. Formally, $G_i = (V, E_i)$, where $E_i = \{(k, j)|\ k \text{ is a parent of } j, w_k[i] = 1\}$. The i-th locus graph G_i induces a subgraph of the spanning tree $\mathcal{T}(G)$. Since the subgraph is a forest, it will be referred to as the *i-th locus forest* and denoted by $\mathcal{T}(G_i)$. Figure 2 (C) shows the locus graphs and the loucs forests of the given pedigree.

The locus graphs can be used to identify some implicit constraints on the h-variables as follows. First, we need "symmetrizing" the h-variables and d-constants: for any edge $(k, j) \in E$, define $h_{k,j} = h_{j,k}$ and $\mathbf{d}_{k,j} = \mathbf{d}_{j,k}$.

[2] The reader can also verify that the direction of an edge will not affect the graph traversal and the ensuing treatment of constraint equations to be discussed in the next two sections.

Lemma 1 *For any path $\mathcal{P} = j_0, \ldots, j_k$ in locus graph G_i connecting vertices j_0 and j_k, we have*

$$p_{j_0}[i] + p_{j_k}[i] + \sum_{r=0}^{k-1} h_{j_r,j_{r+1}} + d_{j_r,j_{r+1}}[i] = 0$$

Proof: The equation follows easily from Equation (3) and an induction on the length k of the path. ■

Note that the above constraint remains the same no matter in which direction path $\mathcal{P}$ is read, since the addition is over field $F(2)$ and the h-variables and d-constants are symmetric. From the lemma, we can see that for a cycle in G_i, the summation of all the h-variables corresponding to the edges on the cycle is a constant. The constant is said to be *associated* with the cycle.

Corollary 2 *For any cycle $\mathcal{C} = j_0, \ldots, j_k, j_0$ in G_i, there exists a binary constant b defined as $b = \sum_{r=0}^{k} d_{j_r,j_{r+1 \mod k+1}}[i]$ such that $\sum_{r=0}^{k} h_{j_r,j_{r+1 \mod k+1}} = b$.*

Proof: This follows from Lemma 1 and the fact that $p_{j_0}[i] + p_{j_0}[i] = 0$. ■

From Lemma 1, we can easily see that if the p-variables at the endpoints of a path are pre-determined, then the summation of all the h-variables corresponding to the edges on the path is a constant. The constant is said to be *associated* with the path. We construct constraints on h-variables as follows. Again, notice that the following constant b does not depend on the direction that path $\mathcal{P}$ is read.

Corollary 3 *Suppose that $\mathcal{P} = j_0, \ldots, j_k$ is a path in G_i connecting vertices j_0 and j_k, and the variables $p_{j_0}[i]$ and $p_{j_k}[i]$ are pre-determined. Then there exists a binary constant b defined as $b = p_{j_0}[i] + p_{j_k}[i] + \sum_{r=0}^{k-1} d_{j_r,j_{r+1}}[i]$ such that $\sum_{r=0}^{k-1} h_{j_r,j_{r+1}} = b$.*

3 An $O(mn^3)$ Time Algorithm for ZRHC

Since the number of h-variables is at most $2n$, our key idea is to first derive a system of $O(mn)$ linear equations on the h-variables, using paths and cycles in G_i and the pre-determined p-variables as mentioned in the last section, and then find a *general* solution to the system by using Gaussian elimination so that all inherent freedom in Equation (3) is kept. This new system of equations about the h-variables is clearly necessary for Equation (3). The crux of the construction is to show that it is also sufficient, and thus the p-variables can be determined from the values of the h-variables by a simple traversal of the locus graphs.

3.1 Linear Constraints on the h-Variables

We will introduce constraint equations to "cover" all the edges in each locus graph. As mentioned above, these equations connect the p-variables and will suffice to help determine their values. Note that, since the edges broken in each locus graph involve pre-determined p-variables, we do not have to introduce constraints to cover them. The constraints can be classified into two categories with respect to the spanning tree $\mathcal{T}(G)$: constraints for cross edges and constraints for tree edges.

Cross Edge Constraints. Adding a cross edge e to the spanning tree $\mathcal{T}(G)$ yields a cycle $\mathcal{C}$ in the pedigree graph G. Let $length(\mathcal{C})$ denote the length of cycle $\mathcal{C}$. Suppose that the edge e exists in the i-th locus graph G_i, and consider two cases of the cycle $\mathcal{C}$ with respect to graph G_i.

Case 1: The cycle exists in G_i. We introduce a constraint along the cycle as in Corollary 2. This constraint is called a *cycle constraint*. The set of such cycle constraints for edge e in all locus graphs is denoted by $C^{\mathbb{C}}(e)$, *i.e.*

$$C^{\mathbb{C}}(e) = \{(b, e) \mid b \text{ is associated with the cycle in } \mathcal{T}(G_i) \cup \{e\}, 1 \leq i \leq m\}$$

The set of cycle constraints for all cross edges is denoted by $C^{\mathbb{C}} = \biguplus_{e \in E^{\mathbb{X}}} C^{\mathbb{C}}(e)$.

Case 2: Some of the edges of the cycle do not exist in G_i. This means that the cycle $\mathcal{C}$ is broken into several disjoint paths in G_i by the pre-determined vertices. Since e exists in G_i, exactly one of these paths, denoted

Procedure CROSS_EDGE_CONSTRAINTS
input: locus graphs $G_{[1..m]}$ and the spanning tree $\mathcal{T}(G)$
output: cross edge constraint sets $C^{\mathbb{C}}$, $C^{\mathbb{P}}$
begin
 $C^{\mathbb{C}} = C^{\mathbb{P}} = \emptyset$;
 for each cross edge e
  Suppose that $\mathcal{C}$ is the cycle in $\mathcal{T}(G) \cup \{e\}$;
  $C^{\mathbb{P}}(e) = C^{\mathbb{C}}(e) = \emptyset$;
  for each locus i
   if $\mathcal{C}$ is connected in G_i
    Let $b = \sum_{(k,j)\in \mathcal{C}} d_{k,j}[i]$;
    $C^{\mathbb{C}}(e) = C^{\mathbb{C}}(e) \uplus \{(e, b)\}$;
   else
    Suppose $\mathcal{P}$ is the path containing e in $\mathcal{T}(G_i) \cup \{e\}$;
    Let vertices j_1 and j_2 be the endpoints of $\mathcal{P}$;
    Define $b = p_{j_1}[i] + p_{j_2}[i] + \sum_{(k,j)\in \mathcal{P}} d_{k,j}[i]$;
    $C^{\mathbb{P}}(e) = C^{\mathbb{P}}(e) \uplus \{(j_1, j_2, b, e)\}$;
  $C^{\mathbb{C}} = C^{\mathbb{C}} \uplus C^{\mathbb{C}}(e)$;
  $C^{\mathbb{P}} = C^{\mathbb{P}} \uplus C^{\mathbb{P}}(e)$;
end.

Procedure TREE_EDGE_CONSTRAINTS
input: locus forests $\mathcal{T}(G_{[1..m]})$ and a fixed tree edge e_0
output: tree edge constraint set $C^{\mathbb{T}}$
begin
 $C^{\mathbb{T}} = \emptyset$;
 for each locus i
  for each connected component $\mathcal{T}$ in $\mathcal{T}(G_i)$
   if $\mathcal{T}$ has no pre-determined vertices
    Arbitrarily pick a vertex j_0 in $\mathcal{T}$ as the seed of $\mathcal{T}$;
   else
    Arbitrarily pick a pre-determined vertex j_0 in $\mathcal{T}$ as the seed of $\mathcal{T}$;
   for each pre-determined vertex $j_1 \neq j_0$ in $\mathcal{T}$
    Let $\mathcal{P}$ be the path between j_0 and j_1;
    Define $b = p_{j_0}[i] + p_{j_1}[i] + \sum_{(k,j)\in \mathcal{P}} d_{k,j}[i]$;
    $C^{\mathbb{T}} = C^{\mathbb{T}} \uplus \{(j_0, j_1, b, e_0)\}$;
end.

Figure 3: The procedure for generating constraints.

Algorithm ZRHC_PHASE
[**IMPROVED ZRHC_PHASE**]
input: pedigree $G = (V, E)$ and genotype $\{\mathbf{g}_j\}$
output: a general solution of $\{\mathbf{p}_j\}$
begin
 Step 1. *Preprocessing*
  Construct a [**low-stretch**] spanning tree $\mathcal{T}(G)$ on G;
  Let e_0 be an arbitrary tree edge;
  for each locus i
   Generate the locus graph G_i;
   Generate the locus forest $\mathcal{T}(G_i)$;
   Identify the pre-determined nodes;
 Step 2. *Constraint generation*
  CROSS_EDGE_CONSTRAINTS($G_{[1..m]}, \mathcal{T}(G)$, $C^{\mathbb{C}}$, $C^{\mathbb{P}}$);
  TREE_EDGE_CONSTRAINTS($\mathcal{T}(G_{[1..m]})$, $C^{\mathbb{T}}$, e_0);
 [**Step 2′.** ***Redundant Constraint Elimination***
  COMPACT_CONSTRAINTS($C^{\mathbb{C}}$, $C^{\mathbb{P}}$, $C^{\mathbb{T}}$, e_0);]
 Step 3. *Solve the h-variables*
  Apply Gaussian elimination on $C^{\mathbb{C}} \uplus C^{\mathbb{P}} \uplus C^{\mathbb{T}}$ to get a general solution of the h-variables;
 Step 4. *Solve the p-variables by propagation*
  for each locus i
   for each connected component $\mathcal{T}$ in $\mathcal{T}(G_i)$
    if $\mathcal{T}$ has no pre-determined vertices
     Set the p-variable of the seed as a free variable and treat it as a determined value;
    Traverse $\mathcal{T}$ by BFS starting from the seed;
     for each edge (j, k) in $\mathcal{T}$
      if $p_j[i]$ is determined but $p_k[i]$ is undetermined
       $p_k[i] = p_j[i] + h_{j,k} + d_{j,k}[i]$;
 return $\{\mathbf{p}_j\}$;
end.

Figure 4: The $O(mn^3)$ time algorithm ZRHC_PHASE, and the $O(mn^2 + n^3 \log^2 n \log\log n)$ time algorithm IMPROVED_ZRHC_PHASE. The additional instructions in IMPROVED_ZRHC_PHASE are highlighted by bold font in square brackets.

In order to save running time, we use disjoint union (*i.e.* $\uplus$) in algorithms ZRHC_PHASE and IMPROVED_ZRHC_PHASE.

as $\mathcal{P}$, contains e. Observe that both endpoints of $\mathcal{P}$ are pre-determined and thus Corollary 3 could give us a constraint concerning the h-variables along the path. Such a constraint will be called a *path constraint*. The set of such path constraints for e in all locus graphs G_i is denoted by $C^{\mathbb{P}}(e)$, *i.e.*

$$C^{\mathbb{P}}(e) = \left\{ (k, j, b, e) \,\middle|\, \begin{array}{l} \text{in } \mathcal{T}(G_i) \cup \{e\}, b \text{ is associated with the path containing } e \\ \text{connecting two pre-determined vertices } k \text{ and } j, 1 \leq i \leq m \end{array} \right\}$$

The set of path constraints for all cross edges is denoted by $C^{\mathbb{P}} = \biguplus_{e \in E^{\mathbb{X}}} C^{\mathbb{P}}(e)$.

Tree Edge Constraints. By Corollary 3, there is an implicit constraint concerning the h-variables along each path between two pre-determined vertices in the same connected component of $\mathcal{T}(G_i)$. Therefore, for each connected component $\mathcal{T}$ of $\mathcal{T}(G_i)$, we arbitrarily pick a pre-determined vertex in the component as the *seed* vertex, and generate a constraint for the unique path in $\mathcal{T}(G_i)$ between the seed and each of the other pre-determined vertices in the component, as in Corollary 3. Such a constraint will be called a *tree constraint*. Notice that if there exists any component having no pre-determined vertices, then locus i must be heterozygous across the entire pedigree and $\mathcal{T}(G_i)$ is actually a spanning tree. Such a locus will be referred to as an *all-heterozygous* locus. For such a locus i, we arbitrarily pick a vertex in $\mathcal{T}(G_i)$ as the seed, but will not generate at tree constraints.

To conform with the notation of path constraints and for the convenience of presentation, we arbitrarily pick a tree edge denoted as e_0, and write the set of tree constraints at all loci as

$$C^{\mathbb{T}} = \left\{ (k, j, b, e_0) \,\middle|\, \begin{array}{l} \text{in a connected component of } \mathcal{T}(G_i) \text{ with seed } k, b \text{ is associated with} \\ \text{the path connecting vertices } k \text{ and a predetermined vertex } j, 1 \leq i \leq m \end{array} \right\}$$

Note that e_0 is the same for all the tree constraints, and will be used as an *indictor* to distinguish tree constraints from path constraints defined by cross edges.

Again, we need symmetrize path constraints and tree constraints: given any constraint (k, j, b, e) generated for a path connecting two pre-determined vertices k and j in a locus graph, define $(k, j, b, e) = (j, k, b, e)$. The above constructions of $C^{\mathbb{C}}$, $C^{\mathbb{P}}$ and $C^{\mathbb{T}}$ are more formally described as pseudo-code in Figures 3. We can easily see that

Lemma 4 $|C^{\mathbb{C}}| + |C^{\mathbb{P}}| + |C^{\mathbb{T}}| = O(mn)$.

3.2 Solving the Linear System for ZRHC Using the New Constraints

We now describe how to solve the system in Equation (3) in $O(mn^3)$ time. The pseudo-code for solving the system is formally given as algorithm **ZRHC_phase** in Figure 4. Here, we first construct the cycle, path and tree constraints on the h-variables, and pick a vertex as the seed for every connected component in the locus forests $\mathcal{T}(G_i)$, as described in the last subsection. Then we solve these constraints by using Gaussian elimination to obtain a general solution of the h-variables, which may contain some *free* h-variables. Next, for each connected component with no pre-determined vertices, we set the p-variable of the seed as a *free* variable and treat it as a determined value. Finally, we perform a (*e.g.* breadth-first) traversal on the spanning forest $\mathcal{T}(G_i)$ of each locus graph G_i. For each connected component of $\mathcal{T}(G_i)$, we start from the seed and propagate its p-variable value to the undetermined vertices in the component by using the solution for the h-variables, which will result in functions of the *free* h-variables and at most one *free* p-variable. Note that in the last step of the algorithm, $p_k[i]$ is expressed as a linear combination of the free variables in $p_j[i]$ and the free h-variables with an appropriate constant term.

To show the correctness of the algorithm, we need only show that the solution found by the algorithm is a feasible solution for Equation (3) and vice the versa. Since we determine the p-variables based on the linear system for the h-variables derived from Equation (3), any feasible solution to Equation (3) will be included in the (general) solution found by our algorithm. In other words, we do not lose any degrees of freedom in the solution process. Hence, it suffices to show that our solution satisfies Equation (3).

Lemma 5 *The p-variables and h-variables determined by algorithm **ZRHC_Phase** satisfy the linear system in Equation (3).*

Theorem 6 *The running time of algorithm **ZRHC_Phase** is $O(mn^3)$.*

4 Speeding up the Algorithm by Fast Elimination of Redundant Equations

The bottleneck in the above algorithm is step 3 where we have to spend $O(mn^3)$ time to solve a system of $O(mn)$ equations over $O(n)$ variables. Clearly, most of the equations are redundant and can be expressed as linear combinations of other equations. The question is how to detect and eliminate these redundant equations (without using Gaussian elimination, of course). To our best knowledge, there are no methods that would eliminate redundant equations for any system of linear equations over any field faster than Gaussian elimination asymptotically in the worst case. Here, we give such a method taking advantage of the underlying pedigree structure. We first give a general method to compact path and tree constraints that correspond to paths on a cycle in the pedigree graph.

Let $\mathcal{C}$ be a cycle in the pedigree graph G induced by cross edge e_1. For convenience, we say that a path/tree constraint is on the cycle $\mathcal{C}$ if it corresponds to a path/edge on $\mathcal{C}$. The following lemma shows that the path/tree constraints on a cycle can be greatly compacted, and is the key to our algorithm to eliminate redundant constraints.

Lemma 7 *Given a set C of path/tree constraints on cycle $\mathcal{C}$, we can reduce C to an equivalent constraint set of size at most $2 \cdot length(\mathcal{C})$ in time $O(|C|)$.*

An immediate application of Lemma 7 is to remove redundancy from each path constraint set $C^{\mathbb{P}}(e)$, since the path constraints in $C^{\mathbb{P}}(e)$ are all on the cycle induced by e.

Corollary 8 *Given the path constraint set $C^{\mathbb{P}}(e)$, we can reduce it to an equivalent constraint set of size at most $2 \cdot length(\mathcal{C})$ in time $O(|C^{\mathbb{P}}(e)|)$, where $\mathcal{C}$ is the cycle induced by cross edge e.*

We can also use Lemma 7 to remove redundant tree constraints. Note that the construction in the proof of Lemma 7 still works if the constraints in C are all tree constraints involving no cross edge e_1. Moreover, the resultant set $\widehat{C}$ contains only constraints of the form (k_0, j, b, e_0). This implies that $|\widehat{C}| \le n$. Therefore, the following corollary holds.

Corollary 9 *Given the tree constraint set $C^{\mathbb{T}}$, we can reduce it to an equivalent constraint set of size at most n, in $O(|C^{\mathbb{T}}|)$ time.*

4.1 Elimination of Redundant Cycle Constraints

Each cross edge e induces a unique cycle $\mathcal{C}$. Since every constraint in $C^{\mathbb{C}}(e)$ concerns the same set of h-variables corresponding to the edges on $\mathcal{C}$, each $C^{\mathbb{C}}(e)$ contains only one independent constraint. Moreover, these constraints are consistent with each other if and only if their associated constants are identical, which can be checked in $O(m)$ time. Because the total number of cross edges are at most $n+1$, we have

Lemma 10 *Given the cycle constraint set $C^{\mathbb{C}}$, we can reduce it to an equivalent constraint set of size at most $n+1$ in $O(mn)$ time.*

4.2 Elimination of Redundant Path Constraints

We will show how to reduce the path constraints $C^{\mathbb{P}}$ on a general pedigree to an equivalent set of path constraints with size $O(n \log^2 n \log\log n)$ in $O(mn)$ time (assuming $\log^2 n \log\log n < m$). Furthermore, for tree pedigrees (*i.e.* pedigrees with no mating loops) we can make the equivalent constraint set as small as $O(n)$. For pedigrees with an all-heterozygous locus across the entire pedigree, we can first transform $C^{\mathbb{P}}$ into an equivalent tree constraint set with size $O(mn)$, and then will remove its redundancy via Corollary 9. We first start with the special cases.

Elimination of Redundant Path Constraint on Tree Pedigrees. Observing that the length of each (simple) cycle in the pedigree graph of a tree pedigree is a constant (*i.e.* 4, of which an example is given in Figure 2(B)), we can upper bound the total number of path constraints as follows.

Lemma 11 *Given the path constraint set $C^{\mathbb{P}}$ on a tree pedigree, we can reduce it to an equivalent path constraint set of size $O(n)$ in $O(mn)$ time.*

```
Procedure COMPACT_PT_CONST
input: a set C of path/tree constraints
        on cycle 𝒞 induced by cross edge e_1,
        and a fixed tree edge e_0
output: a compact constraint set Ĉ ≡ C
begin
 Construct the constraint graph G* for C;
 Ĉ = ∅;
 for each connected component S of G*
  Pick an arbitrary vertex k_0 ∈ S as the root of S;
  Traverse S by BFS starting from k_0;
   while there exists unvisited vertices in G*
    Visit an unvisited vertex, say k, in the BFS order;
    for each constraint c = (k_0, j, b, e) in C
     Ĉ = Ĉ ⊎ {c};
    for each constraint c = (k, j, b, e) in C
               s.t. vertex k ≠ k_0 is visited before j
     for each constraint c' = (k_0, k, b', e') in Ĉ
      b'' = b + b';
      if {e} ∪ {e'} = {e_0, e_1}
       e'' = e_1;
      else
       e'' = e_0;
      Construct a new constraint c'' = (k_0, j, b'', e'');
      if there exists a constraint (k_0, j, b'' + 1, e'') ∈ Ĉ
       exit "The input genotypes are inconsistent.";
      if c'' ∉ Ĉ
       Ĉ = Ĉ ⊎ {c''};
 return Ĉ;
end.
```

Figure 5: The procedure for compacting path and tree constraints on a cycle.

```
Procedure COMPACT_CONSTRAINTS
input: C^C, C^P, C^T, and a fixed tree edge e_0
output: compact C^C, C^P and C^T
begin
 Step 1. Removing redundant cycle constraints
  for each cross edge e
   Pick an arbitrary constraint, say c = (e, b), from C^C(e);
   if there exists a constraint (e, b + 1) ∈ C^C(e)
    exit "The input genotypes are inconsistent.";
   C^C = C^C − C^C(e) ⊎ {c};
 Step 2. Processing path constraints
  if G is a tree pedigree
   for each cross edge e
    Ĉ^P(e) = ∅;
    for each constraint c = (k, j, b, e) ∈ C^P(e)
     if there exists a constraint (k, j, b + 1, e) ∈ C^P(e)
      exit "The input genotypes are inconsistent.";
     Ĉ^P(e) = Ĉ^P(e) ⊎ {c};
    C^P = C^P − C^P(e) ⊎ Ĉ^P(e);
  else if G has an all-heterozygous locus
   for each cross edge e
    Let (b', e) be the cycle constraint for e in C^C;
    for each constraint c = (k, j, b, e) in C^P(e)
     Construct a new constraint c' = (k, j, b − b', e_0);
     C^T = C^T ⊎ {c'} ;
  else (i.e. G is a general pedigree)
   for each cross edge e
    Ĉ^P(e) = COMPACT_PT_CONST(C^P(e), e_0);
    C^P = C^P − C^P(e) ⊎ Ĉ^P(e);
 Step 3. Removing redundant tree constraints
  C^T = COMPACT_PT_CONST(C^T, e_0);
end.
```

Figure 6: The procedure for removing redundant constraints.

Transformation of Path Constraints on Pedigrees with an All-Heterozygous Locus. Observe that for a pedigree with an all-heterozygous locus i, each cross edge induces a cycle that exists in the locus graph G_i and has a cycle constraint in the (reduced) set $C^{\mathbb{C}}$. This allows us to transform all the path constraints into tree constraints given the cycle constraints as follows.

Corollary 12 *Given the path constraint set $C^{\mathbb{P}}$ on a pedigree with an all-heterozygous locus, we can construct a equivalent tree constraint set of size $O(mn)$ in $O(mn)$ time.*

Elimination of Redundant Path Constraints on a General Pedigree. Now we consider how to compact path constraints in the general case. As shown in Corollary 8, given a cross edge e inducing cycle $\mathcal{C}$, we can compact the constraints in $C^{\mathbb{P}}(e)$ so that at most $2 \cdot length(\mathcal{C})$ constraints are kept. Clearly, the compact $C^{\mathbb{P}}$ has size at most $O(n^2)$ since the number of cross edges is at most $n+1$ and the length of a cycle containing a cross edge is at most n. This bound can be improved by observing that the total length of all cycles in G is related to the average *stretch* [8] of G with respect to the spanning tree $\mathcal{T}(G)$. Hence, we can obtain a sharper upper bound on $|C^{\mathbb{P}}|$ by using a low-stretch spanning tree $\mathcal{T}(G)$ as constructed in [8].

We first give a formal definition of the *stretch* of an unweighted connected graph with respect to a spanning tree. Given a spanning tree $\mathcal{T}$ on an unweighted connected graph $G = (V, E)$ (*e.g.* the pedigree graph), we define the stretch of an edge $(k, j) \in E$, denoted as $streth_{\mathcal{T}}(k, j)$, to be the length of the unique path (*i.e.* the number of edges on the path) in $\mathcal{T}$ between k and j. The average stretch of G with respect to $\mathcal{T}$ is then defined as $avg\text{-}stretch_{\mathcal{T}}(E) = \frac{1}{|E|} \sum_{(k,j)\in E} stretch_{\mathcal{T}}(k, j)$.

Lemma 13 *Given a pedigree G, we can build a low-stretch spanning tree $\mathcal{T}(G)$ in $O(n \log n)$ time such that $|C^{\mathbb{P}}| = O(n \log^2 n \cdot \log\log n)$ after compacting.*

4.3 Elimination of Redundant Tree Constraints and the Final Algorithm

After we process (*i.e.* compact or transform) the path constraints, we eliminate redundant tree constraints and obtain a compact tree constraint set containing at most n constraints as shown in Corollary 9. The complete algorithm for eliminating redundant cycle, path and tree constraints is given in Figure 6 as procedure **Compact_Constraints**. The next theorem summarizes the above discussion.

Theorem 14 *Given the constraint sets $C^{\mathbb{C}}$, $C^{\mathbb{P}}$ and $C^{\mathbb{T}}$ on a pedigree, we can reduce them to an equivalent constraint set of size $O(n \cdot \log^2 n \log\log n)$ in $O(mn)$ time. In particular, for tree pedigrees and pedigrees with an all-heterozygous locus, the equivalent constraint set has size $O(n)$.*

We can incorporate the above redundant constraint elimination procedure **Compact_Constraints** into the $O(mn^3)$ time algorithm for ZRHC to obtain an improved algorithm **Improved_ZRHC_Phase** as shown in Figure 4. (An example of how **Improved_ZRHC_Phase** works is given in the appendix). The following theorem is obvious given Theorem 14.

Theorem 15 *The algorithm **Improved_ZRHC_Phase** solves the ZRHC problem correctly on any pedigree in $O(mn^2 + n^3 \log^2 n \log\log n)$ time. Moreover, it solves ZRHC on tree pedigrees or pedigrees with an all-heterozygous locus in $O(mn^2 + n^3)$ time.*

5 Concluding Remarks

It remains interesting if the time complexity for ZRHC on general pedigrees can be improved to $O(mn^2 + n^3)$ or lower. Another open question is how to use the algorithm to solve MRHC on pedigrees that require only a small (constant) number of recombinants.

6 Acknowledgements

We would like to thank Jing Li and Cliff Zhang for many useful discussions on ZRHC. The research is supported in part by NSF grant CCR-0309902, NIH grant LM008991-01, NSFC grant 60528001, National Key Project for Basic Research (973) grant 2002CB512801, and a fellowship from the Center for Advanced Study, Tsinghua University. Also, the authors wish it to be known that the first two authors should be regarded as joint First Authors of this paper.

References

[1] G. R. Abecasis, S. S. Cherny, W. O. Cookson and L. R. Cardon, Merlin–rapid analysis of dense genetic maps using sparse gene flow trees. *Nat Genet*, 30(1):97-101, 2002.

[2] M. Y. Chan, W. Chan, F. Chin, S. Fung, and M. Kao Linear-Time Haplotype Inference on Pedigrees without Recombinations *Proc. of the 6th Annual Workshop on Algorithms in Bioinformatics (WABI06)*, to appear, 2006.

[3] F. Chin and Q. Zhang. Haplotype inference on tightly linked markers in pedigree data. *Unpublished manuscript*, 2005.

[4] F. Chin, Q. Zhang, and H. Shen. k-Recombination Haplotype Inference in Pedigrees. *Proc. of the International Workshop on Bioinformatics Research and Applications (ICCS)*, 985-993, 2005.

[5] K. Doi, J. Li and T. Jiang. Minimum recombinant haplotype configuration on tree pedigrees. *Proc. of the 3rd Annual Workshop on Algorithms in Bioinformatics (WABI03)*, 339-353, 2003.

[6] M. J. Daly, J. D. Rioux, S. F. Schaffner, T. J. Hudson, and E. S. Lander. High-resolution haplotype structure in the human genome. *Nat Genet*, 29(2):229–32, 2001.

[7] D. Coppersmith, and S. Winograd. Matrix multiplication via arithmetic programming. *J. Symb. Comput.* 9:251-280, 1990.

[8] M. Elkin, Y. Emeky, D. A. Spielman, S. Teng. Lower-Stretch Spanning Trees *Proc. 37th ACM Symposium on Theory of computing(STOC'05)*, 494-503, 2005.

[9] E. Eskin, E. Halperin, and R. M. Karp. Large scale reconstruction of haplotypes from genotype data. *In Proc. RECOMB'03*, pages 104–113, 2003.

[10] S. B. Gabriel, S. F. Schaffner, H. Nguyen, J. M. Moore, J. Roy, B. Blumenstiel, J. Higgins, M. DeFelice, A. Lochner, M. Faggart, S. N. Liu-Cordero, C. Rotimi, A. Adeyemo, R. Cooper, R. Ward, E. S. Lander, M. J. Daly, and D. Altshuler. The structure of haplotype blocks in the human genome. *Science*, 296(5576):2225–9, 2002.

[11] D. Gusfield. Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. *In Proc. RECOMB'02*, pages 166–175, 2002.

[12] D. Gusfield. An overview of combinatorial methods for haplotype inference. *Lecture Notes in Computer Science (2983): Computational Methods for SNPs and Haplotype Inference.*, 9–25, 2004.

[13] D. F. Gudbjartsson, K. Jonasson, M. L. Frigge, and A. Kong. Allegro, a new computer program for multipoint linkage analysis. *Nat Genet*, 25(1):12–13, 2000.

[14] M. Giesbrecht, A. Lobo, and B. Saunders. Certifying inconsistency of sparse linear systems. *International Symposium on Symbolic and Algebraic Computation* pp. 113-119, 1998.

[15] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopins Univeristy Press, Baltimore, Maryland, third edition, 1996.

[16] L. Helmuth. Genome research: Map of the human genome 3.0. *Science*, 293(5530):583–585, 2001.

[17] B. V. Halldórsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph and S. Istrail. A survey of computational methods for determining haplotypes. *Lecture Notes in Computer Science (2983): Computational Methods for SNPs and Haplotype Inference.*, 26–47, 2004.

[18] The international HapMap Consortium. International HapMap Project. *Nature* 426:789-796, 2003.

[19] L. Helmuth. Genome research: Map of the human genome 3.0. *Science*, 293(5530):583–585, 2001.

[20] L. Kruglyak, M. J. Daly, M. P. Reeve-Daly and E. S. Lander. Parametric and nonparametric linkage analysis: a unified multipoint approach. *Am J Hum Genet*, 58(6):1347–63, 1996.

[21] J. Li and T. Jiang. Efficient rule-Based haplotyping algorithm for pedigree data. *Proc. 7th Annual Conference on Research in Computational Molecular Biology (RECOMB'03)*, 197-206, 2003.

[22] J. Li and T. Jiang. An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming. *Proc. RECOMB'04*, 20-29, 2004.

[23] J. Li and T. Jiang. Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by Integer Linear Programming. *Journal of Computational Biology* 12(6), 719-739, 2005.

[24] S. Lin and T. P. Speed. An algorithm for haplotype analysis. *J Comput Biol*, 4(4):535–46, 1997.

[25] X. Li, Y. Chen and J. Li. An efficient algorithm for the zero-recombinant haplotype cofiguration problem. *Unpublished manuscript*, 2006.

[26] L. Liu, X. Chen, J. Xiao and T. Jiang. Complexity and approximation of the minimum recombination haplotype configuration problem. *Proc. 16th International Symposium on Algorithms and Computation (ISAAC'05)*, 370-379, 2005.

[27] B. LaMacchia and A. Odlyzko. Solving large sparse linear systems over finite fields. *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology. Lecture Notes In Computer Science* 537, pp. 109 - 133, 1990.

[28] J. R. O'Connell. Zero-recombinant haplotyping: applications to fine mapping using SNPs. *Genet Epidemiol*, 19 Suppl 1:S64–70, 2000.

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

[30] H. Seltman, K. Roeder, and B.D. Devlin. Transmission/disequilibrium test meets measured haplotype analysis: family-based association analysis guided by evolution of haplotypes. *Am J Hum Genet*, 68(5):1250–1263, 2001.

[31] E. Sobel, K. Lange, J. O'Connell, and D. Weeks. Haplotyping algorithms. *T. Speed and M. Waterman, eds., Genetic Mapping and DNA Sequencing, IMA Vol in Math and its App*, 81:89–110, 1996.

[32] E. Sobel and K. Lange. Descent graphs in pedigree analysis: applications to haplotyping, location scores, and marker-sharing statistics. *Am J Hum Genet*, 58(6):1323–37, 1996.

[33] V. Strassen. Gaussian Elimination is not Optimal *Numer. Math.* 13, pp. 354-356, 1969

[34] P. Tapadar, S. Ghosh, and P. P. Majumder. Haplotyping in pedigrees via a genetic algorithm. *Hum Hered*, 50(1):43–56, 2000.

[35] E. M. Wijsman. A deductive method of haplotype analysis in pedigrees. *Am J Hum Genet*, 41(3):356–73, 1987.

[36] D. Wiedemann. Solving sparse linear euquations over finite fields. *IEEE Trans. Inf. Theory*, IT-32:52-62, 1986

[37] S. Zhang *et al.* Transmission/Disequilibrium test based on haplotype sharing for tightly linked markers. *Am J Hum Genet*, 73(3):566–579, 2003.

Appendix

A. Related Biological Definitions

The genome of an organism consists of *chromosomes* that are double strand DNAs. Locations on a chromosome can be labelled using *markers*, which are small segments of DNA with some specific features. A physical position of a marker on a chromosome is called a *marker locus* and a marker state is called an *allele*. In *diploid* organisms, chromosomes come in pairs. The status of two alleles at a particular marker locus of a pair of chromosomes is called a *marker genotype*. The genotype information at a locus will be denoted using a set, *e.g.* $\{a, b\}$. If the two alleles a and b are the same, the genotype is *homozygous*. Otherwise, it is *heterozygous*. A *haplotype* consists of all alleles, one from each locus, that are on the same chromosome. Figure 1(A) illustrates the above concepts, where alleles are represented by their numerical IDs.

A pedigree can be defined formally as follows.

Definition 16 *A pedigree graph is a weakly connected directed acyclic graph (DAG) $G = (V, E)$, where $V = M \cup F$, M stands for the male nodes, F stands for the female nodes. The in-degree of each node is 0 (founders) or 2 (non-founders). If the in-degree of a node is 2, one edge must start from a male node (called father) and the other edge from a female node (called mother) and the node itself is a child of its parents (father and mother).*

A *mating loop* consists of two distinct paths from a node x to a node y. Figure 1(B) illustrates an example pedigree with a mating loop. The Mendelian law of inheritance states that the alleles of a child must come from the alleles of its parents at each marker locus (*i.e.* assuming no mutations within a pedigree). In other words, the two alleles at each locus of the child have different origins: one is from its father (which is called the *paternal* allele) and the other from its mother (which is called the *maternal* allele). Usually, a child inherits a complete haplotype from each parent. However, *recombination* may occur, where the two haplotypes of a parent get shuffled due to a crossover of chromosomes and one of the shuffled copies is passed on to the child. Such an event is called a recombination event and its result is called a *recombinant*. Since markers are usually short DNA sequences, we assume that recombination only occurs between markers. Figure 1(C) illustrates an example where the paternal haplotype of member 3 is the result of a recombinant. Paternal allele and maternal allele at each locus is separated by a "|" in this figure.

We use the term *haplotype configuration* to describes not only the paternal and maternal haplotypes of an individual, but also the grandpaternal or grandmaternal origin of each allele on the haplotypes. Observe that the number of recombinants required in a pedigree can be easily computed once the haplotype configuration of each member of the pedigree is given. The MRHC problem is defined as follows.

Definition 17 *(MRHC) Given a pedigree and genotype information for each member of the pedigree, find a haplotype configuration for the pedigree that requires the minimum number of recombinants.*

Namely, the ZRHC problem is a special case of MRHC with the following definition.

Definition 18 *(ZRHC) Given a pedigree and genotype information for each member of the pedigree, find a haplotype configuration for the pedigree that requires no recombinant (if such solution exist).*

B. An Example for Algorithm Improved_ZRHC_phase

The example in the following Figure 7 aims to demonstrate how algorithm **Improved_ZRHC_phase** works (see Figure 6 for the pseudo-code of the algorithm). The input pedigree with genotype data is shown in Figure 7(A) and the corresponding pedigree graph is in Figure 7(B).

Step 1, we generate the locus graphs (or forests) illustrated in Figure 7(C) and identify the pre-determined vertices in Figure 7(D). Moreover, we arbitrarily pick a tree edge, say $e_{1,4}$, as the *indicator* to distinguish tree constraints from path constraints, which is defined in subsection 3.1.

Step 2, we generate cycle, path and tree constraints as follows. For example, given the cycle $\mathcal{C} = v_2v_5v_3v_6v_2$ in the second locus graph of Figure 7(D), we denote the cycle constraint $h_{2,5} + h_{3,5} + h_{3,6} + h_{2,6} = 0$ by the form $(0, e_{2,6})$. Afterwards, we have $C^{\mathbb{C}} = \{(0, e_{2,6}), (0, e_{2,6})\}$, $C^{\mathbb{P}} = \{(v_4, v_9, 0, e_{4,9}), (v_9, v_8, 1, e_{4,9})\}$ and $C^{\mathbb{T}} = \{(v_6, v_8, 0, e_{1,4}), (v_6, v_9, 1, e_{1,4}), (v_4, v_8, 1, e_{1,4})\}$.

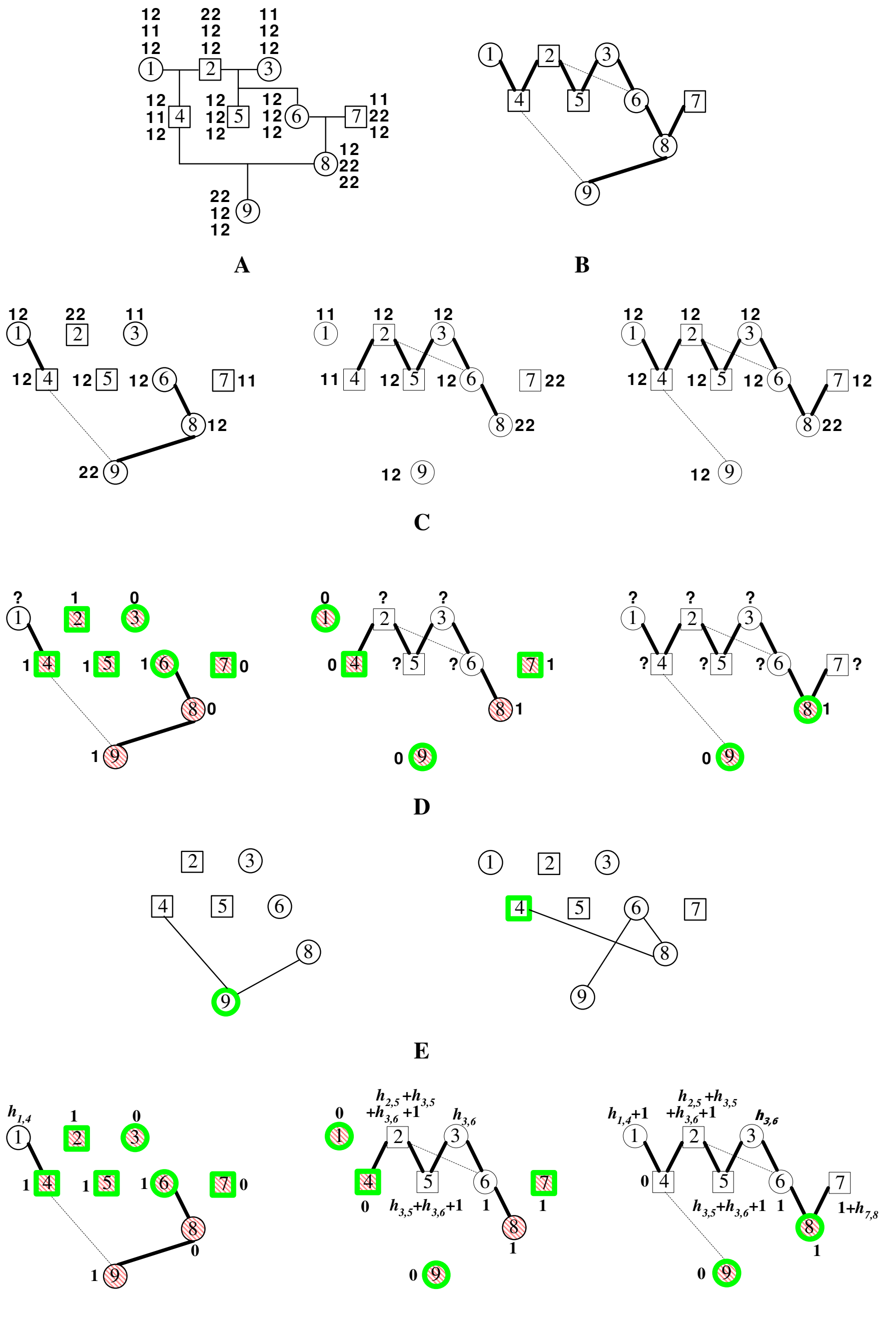

Figure 7: A. An example input pedigree with genotype data. Here, the alleles at a locus are ordered according to their id numbers instead of phase (which is unknown). B. The pedigree graph with a spanning tree. The tree edges are highlighted. C. Locus graphs for the three loci, respectively. The locus forests are highlighted. D. The pre-determined nodes. On the locus graphs, the pre-determined nodes have the corresponding p-values (*i.e.* the number 0 or 1 near the nodes) with the red shadow highlight, while the undetermined nodes have question marks (*i.e.*"?"). The seeds in the graphs are highlighted by thick green border. E. The constraint graphs. The left graph is for removing redundant path constraints generated by the cross edge $e_{4,9}$, while the right graph is for tree constraints. The roots (see Lemma 7 for the definition) in the graphs are highlighted by thick green border. F. The locus graphs with propagated p-values. The notations are the same as those in Figure 7(D), except that the question marks on the undermined nodes are replaced by their resolved p-values. The resolved p-values are expressed as a linear combination of the free variables in $p_j[i]$ and the free h-variables with an appropriate constant term.

Step 2′, we first remove redundant cycle constraint and get $C^{\mathbb{C}} = \{(0, e_{2,6})\}$. Now we need to take care path constraints. For an instance, given the path constraints $(v_4, v_9, 0, e_{4,9})$ and $(v_9, v_8, 1, e_{4,9})$ induced by the cross edge $e_{4,9}$, we draw the constraint graph illustrated in Figure 7(E) to help removing redundant path constraints, and get $C^{\mathbb{P}} = \{(v_4, v_9, 0, e_{4,9}), (v_9, v_8, 1, e_{4,9})\}$ based on Lemma 7. Then, we construct the constraint graph shown in Figure 7(E) for tree constraints and obtain $C^{\mathbb{T}} = \{(v_4, v_8, 1, e_{1,4}), (v_4, v_6, 1, e_{1,4}), (v_4, v_9, 0, e_{1,4})\}$, according to Lemma 7.

Step 3, we apply Gaussian elimination to the following linear system, which is equivalent to $C^{\mathbb{C}} \biguplus C^{\mathbb{P}} \biguplus C^{\mathbb{T}}$.

$$\begin{cases} h_{2,5} + h_{3,5} + h_{3,6} + h_{2,6} = 0 & \textit{i.e.}\text{ the cycle constraint } (0, e_{2,6}) \\ h_{4,9} = 0 & \textit{i.e.}\text{ the path constraint } (v_4, v_9, 0, e_{4,9}) \\ h_{4,9} + h_{2,4} + h_{2,5} + h_{3,5} + h_{3,6} + h_{6,8} = 1 & \textit{i.e.}\text{ the path constraint } (v_9, v_8, 1, e_{4,9}) \\ h_{2,4} + h_{2,5} + h_{5,3} + h_{3,6} + h_{6,8} = 1 & \textit{i.e.}\text{ the tree constraint } (v_4, v_8, 1, e_{1,4}) \\ h_{2,4} + h_{2,5} + h_{5,3} + h_{3,6} = 1 & \textit{i.e.}\text{ the tree constraint } (v_4, v_6, 1, e_{1,4}) \\ h_{2,4} + h_{2,5} + h_{5,3} + h_{3,6} + h_{6,8} + h_{8,9} = 0 & \textit{i.e.}\text{ the tree constraint } (v_4, v_9, 0, e_{1,4}) \end{cases}$$

Then we obtain the following general solution,

$$\begin{cases} h_{4,9} = 0 \\ h_{6,8} = 0 \\ h_{8,9} = 1 \\ h_{2,6} = h_{2,5} + h_{3,5} + h_{3,6} \\ h_{2,4} = h_{2,5} + h_{3,5} + h_{3,6} + 1 \end{cases}$$

where $h_{2,5}, h_{3,5}, h_{3,6}, h_{1,4}$ and $h_{7,8}$ are free variables.

Step 4, we solve the unknown p-values by propagation from the seeds and obtain all p-values as shown in Figure 7(F).

C. Proofs of Theorems, Corollaries and Lemmas

Proof of Lemma 5: Denote the solution found by our algorithm as $\mathbf{p}$ and $\mathbf{h}$. We need only care about the first equations in Equation (3). If $w_k[i]$ in the equation is 0, the equation only involves two pre-determined p-values, which holds trivially since step 1 of our algorithm explicitly takes care of pre-determined p-values. Otherwise, each such equation corresponds to an edge in some locus graph. Let $e = (j_1, j_2)$ be an edge in locus graph G_i. It represents an equation $p_{j_1}[i] + h_{j_1,j_2} = p_{j_2}[i] + d_{j_1,j_2}[i]$, *i.e.* the first equation on edge (j_1, j_2) in Equation (3).

Given any two vertices j_s and j_t in a same connected component of $\mathcal{T}(G_i)$, we denote by $\mathcal{P}(j_s, j_t)$ the unique path in the component connecting j_s and j_t. Suppose that vertex j_0 is the seed of the component. The key observation here is that for any vertex j_t, no matter vertex j_t is pre-determined or not, our solution satisfies

$$p_{j_t}[i] \;=\; p_{j_0}[i] \;+ \sum_{(k,\; j) \,\in\, \mathcal{P}(j_0, j_t)} (h_{k,j} + d_{k,j}[i]) \tag{4}$$

More precisely, if vertex j_t is pre-determined, Equation (4) holds because of the tree constraint for path $\mathcal{P}(j_0, j_t)$ defined in step 2 of the algorithm; otherwise, $p_{j_t}[i]$ is set to be the value in Equation (4) during the traversal from the seed j_0 to j_t in step 4. More generally, we can build a relationship between the p-values of two vertices j_s and j_t from Equation (4) as follows:

$$p_{j_t}[i] \;=\; p_{j_s}[i] \;+ \sum_{(k,\; j) \,\in\, \mathcal{P}(j_t, j_s)} (h_{k,j} + d_{k,j}[i]) \tag{5}$$

Notice that, as long as j_s and j_t are in the same connected component of $\mathcal{T}(G_i)$, the above equation is satisfied by our solution. With the help of Equation (5), we now show that our solution satisfies all equations represented by tree edges and cross edges.

Case 1: e is a tree edge. Clearly, vertices j_1 and j_2 belong to the same connected component of $\mathcal{T}(G_i)$. We replace j_s and j_t by j_1 and j_2 respectively in Equation (5) and obtain $p_{j_1}[i] \;=\; p_{j_2}[i] + \sum_{(k,\; j) \,\in\, \mathcal{P}(j_1, j_2)} (h_{k,j} + d_{k,j}[i]) \;=\; p_{j_2}[i] \;+ h_{j_1,j_2} + d_{j_1,j_2}[i]$, which means that our solution satisfies the equation represented by the tree edge e.

Case 2: e is a cross edge. Denote by $\mathcal{C}$ the cycle in $\mathcal{T}(G_i) \cup \{e\}$. There are two subcases.

Case 2.1: $\mathcal{C}$ exists in G_i. In this case, we have a cycle constraint for $\mathcal{C}$ in G_i. Suppose that the cycle constraint is $0 = \sum_{(k,j)\in\mathcal{C}}(h_{k,j} + d_{k,j}[i])$. Observe that vertices j_1 and j_2 are in the same connected component of $\mathcal{T}(G_i)$, thus we have $p_{j_1}[i] = p_{j_2}[i] + \sum_{(k,\ j) \in \mathcal{P}(j_1,j_2)}(h_{k,j} + d_{k,j}[i])$ as shown in Equation (5). Observe that cycle $\mathcal{C}$ can be decomposed into path $\mathcal{P}(j_1, j_2)$ and cross edge $e = (j_1, j_2)$. Therefore, summing up the above two equations, we can get $p_{j_1}[i] = p_{j_2}[i] + h_{j_1,j_2} + d_{j_1,j_2}[i]$, which means that our solution satisfies the equation represented by the cross edge e.

Case 2.2.: $\mathcal{C}$ is broken into serval disjoint paths in G_i by pre-determined vertices. Suppose that the unique path containing e is $\mathcal{P}$. Without loss of generality, suppose that vertices k_1 and k_2 are the endpoints of $\mathcal{P}$, and $\mathcal{P}$ has the form $\mathcal{P}(k_1, j_1); e; \mathcal{P}(j_2, k_2)$. Since vertices k_1 and k_2 are pre-determined, our algorithm defines a path constraint for $\mathcal{P}$ in G_i, ***i.e.***, $p_{k_1}[i] = p_{k_2}[i] + \sum_{(k,\ j) \in \mathcal{P}}(h_{k,j} + d_{k,j}[i])$. Observe that vertices k_1 and j_1 are in the same connected component of $\mathcal{T}(G_i)$. Therefore, $p_{k_1}[i] = p_{j_1}[i] + \sum_{(k,\ j) \in \mathcal{P}(k_1,j_1)}(h_{k,j} + d_{k,j}[i])$ based on Equation (5). Similarly, vertices j_2 and k_2 are in the same connected component of $\mathcal{T}(G_i)$ and $p_{j_2}[i] = p_{k_2}[i] + \sum_{(k,\ j) \in \mathcal{P}(j_2,k_2)}(h_{k,j} + d_{k,j}[i])$. Since path $\mathcal{P} = \mathcal{P}(k_1, j_1); e; \mathcal{P}(j_2, k_2)$, summing up the above three equations yields equality $p_{j_2}[i] = p_{j_1}[i] + h_{j_1,j_2} + d_{j_1,j_2}[i]$, which means that our solution satisfies the equation represented by the cross edge e.

In conclusion, the solution produced by algorithm **ZRHC_Phase** satisfies the equations on all edges, and thus the lemma holds. ■

Proof of Theorem 6: Step 1 needs $O(n)$ time for each locus, which takes $O(mn)$ time in total. In step 2, we need at most $O(n)$ time for generating a constraint. Since there are at most $O(mn)$ constraints, it takes at most $O(mn^2)$ time. In step 3, the system has $O(n)$ h-variables and $O(mn)$ constraints. So, Gaussian elimination requires $O(mn^3)$ time. In step 4, every edge is just visited once in the traversal. Since each h-variable is expressed as a linear combination of at most n free h-variables at the end of step 3, every p-variable is expressed as at most n free h-variables and at most one free p-variable on the same locus. Thus, this steps takes $O(mn^2)$ time altogether. Therefore, the entire process takes $O(mn^3)$ time. ■

Proof of Lemma 7: Recall that e_0 represents the fixed tree edge introduced in subsection 3.1 for defining the tree constraints. We use $\widehat{C}$ to denote the equivalent constraint set (to be constructed). Initially, we set $\widehat{C} = \emptyset$.

For convenience, we say a path/tree constraint *connects* vertices j and k, if the constraint has the form (k, j, b, e). To depict a more clear picture of the relationship between constraints in C, we define a *constraint graph* G^* [3] as follows. For each vertex k of the cycle $\mathcal{C}$, we create a vertex in G^*. For each path/tree constraint in C connecting k and j, we build an edge connecting k and j in G^*. Observe that the connected components in G^* naturally partition the constraints in C into disjoint subsets. We will compact the constraints in each of these disjoint subsets separately, and put the resultant equivalent constraint sets into $\widehat{C}$. More precisely, for each connected component of G^*, we pick an arbitrary vertex as the *root* of the component and construct new constraints connecting the root and the other vertices in the component. The details of the construction will be given in the next paragraph. Here, the term root is meant to be synonymous to the term seed defined in subsection 3.1, although each seed is defined for a single locus whereas a root may be used to deal with constraints concerning multiple loci.

Now, we give the details of how to construct $\widehat{C}$. Consider each connected component S of G^*. Suppose that its root is k_0. We process the constraints of C induced by S in the order of increasing distance between the root and the vertices connected by the constraints. In other words, we traverse S by BFS starting from the root. Suppose that we are now visiting vertex j. For each edge (k_0, j), we directly put into $\widehat{C}$ the constraints which have resulted in (k_0, j) in G^*. For each edge (k, j) where $k (k \neq k_0)$ is visited before j, our construction guarantees that $\widehat{C}$ has constraints connecting the root k_0 and k. Suppose one of such constraints is $c' = (k_0, k, b', e')$. Let $c = (k, j, b, e) \in C$ denote the constraint that has resulted in the edge (k, j) in G^*. We

[3] Note that a constraint graph might actually be a multigraph, but this will not affect the correctness of Lemma 7.

generate a new constraint $c'' = (k_0, j, b'', e'')$, where $b'' = b + b'$ and e'' is defined as follows:

$$e'' = \begin{cases} e_1 & \text{if } \{e\} \cup \{e'\} = \{e_0, e_1\} \\ e_0 & \text{otherwise} \end{cases}$$

Because c can be represented as the summation of c'' and $c' \in \widehat{C}$, c is equivalent to c'' given $\widehat{C}$. Then we add c'' to $\widehat{C}$. Here, again the edges e_0 and e_1 are used to indicate tree and path constraints, respectively. Observe that, the above combination of constraints c' and c will result in a constraint c'' involving only tree edges, if and only if both or none of them correspond to paths containing e. Otherwise, the resultant constraint c'' will correspond to some path across e_1.

The BFS creates an equivalent constraint in $\widehat{C}$ for every constraint in C, and thus $\widehat{C} \equiv C$. Recall that each constraint in $\widehat{C}$ has the form (k_0, j, b, e), where j is a vertex in $\mathcal{C}$, e is either e_0 or e_1, and k_0 is the root of the connected component of G^* containing j. It is easy to see that two constraints (k_0, j, b', e) and (k_0, j, b'', e) that differ only in the associated b-constants are consistent with each other if and only if $b' = b''$. Hence, $\widehat{C}$ has at most $2 \cdot length(\mathcal{C})$ different constraints, or otherwise input linear system has no feasible solutions. The construction can be done in $O(|C|)$ time, as more formally described by procedure **Compact_PT_Const** in Figure 5. Hence, the lemma holds. ■

Proof of Lemma 11: By Corollary 8, we can reduce $C^{\mathbb{P}}(e)$ for each cross edge e to an equivalent set of at most 8 path constraints in $O(m)$ time. Since there are at most $n + 1$ cross edges, the set $C^{\mathbb{P}}$ can be reduced to an equivalent set of size $O(n)$ in $O(mn)$ time and thus the lemma holds. ■

Proof of Corollary 12: Let us first focus on a cross edge e. Suppose that the cycle $\mathcal{C}$ is induced by e in $\mathcal{T}(G) \cup \{e\}$, the compact $C^{\mathbb{C}}(e)$ is $\{(b', e)\}$, and e_0 is the fixed tree edge defined in subsection 3.1. Notice that the cycle $\mathcal{C}$ has two disjoint paths connecting each pair of two vertices on the cycle. One of them consists of only tree edges and may be used to define a tree constraint, while the other contains the cross edge and may be used to represent a path constraint. Therefore, with the help of the cycle constraint (b', e), we can transform the path constraint set $C^{\mathbb{P}}(e)$ into a tree constraint set $C(e)$ of the same size as follows:

$$C(e) = \{(k, j,\ b - b',\ e_0) \mid (k, j,\ b,\ e) \in C^{\mathbb{P}}(e)\}$$

It is not hard to see that $C(e) \cup C^{\mathbb{C}}(e) \equiv C^{\mathbb{P}}(e) \cup C^{\mathbb{C}}(e)$, and $C(e)$ can be constructed in $|C^{\mathbb{P}}(e)|$ time.

Let $C = \cup_{e \in E^{\mathbb{X}}} C(e)$. Obviously, $C \equiv C^{\mathbb{P}}$ and $|C| = O(mn)$ since $|C^{\mathbb{P}}| = O(mn)$. Hence, the lemma holds. ■

Proof of Lemma 13: In [8], Elkin *et al.* showed that every unweighted connected graph $G = (V, E)$ contains a spanning tree, into which each edge of the graph can be embedded with an average stretch of $O(\log^2 n \log\log n)$. Moreover, this tree can constructed in $O(|E| \log |V|)$ time. In our situation, such a low-stretch spanning tree $\mathcal{T}(G)$ can be built in $O(n \log n)$ time because $|E| \leq 2n$ and $|V| = n$. The following inequality establishes the relationship between $|C^{\mathbb{P}}|$ and the average stretch of E with respect to $\mathcal{T}(G)$. From

Corollary 8, we have

$$
\begin{aligned}
|C^{\mathbb{P}}| &= \sum_{e \in E^{\mathbb{X}}} |C^{\mathbb{P}}(e)| \\
&\leq \sum_{\mathcal{C} \text{ induced by } e \in E^{\mathbb{X}}} 2 \cdot length(\mathcal{C}) \\
&= \sum_{e\,=\,(k,j) \in E^{\mathbb{X}}} 2 \cdot \big(stretch_{\mathcal{T}}(k,j) + 1\big) \\
&\leq 2 \sum_{(k,j) \in E} stretch_{\mathcal{T}}(k,j) + 2n \\
&\leq 2 \cdot |E| \cdot avg\text{-}stretch_{\mathcal{T}}(E) + 2n \\
&\leq 2 \cdot (2n) \cdot O(\log^2 n \log\log n) + 2n \\
&= O(n \cdot \log^2 n \log\log n)
\end{aligned}
$$

■