Linear-Time Reconstruction of Zero-Recombinant Mendelian Inheritance on Pedigrees without Mating Loops

LAN LIU[1] TAO JIANG[1]
lliu@cs.ucr.edu jiang@cs.ucr.edu

[1] *Department of Computer Science and Engineering, University of California, Riverside, CA, USA*

With the launch of the international HapMap project, the haplotype inference problem has attracted a great deal of attention in the computational biology community recently. In this paper, we study the question of how to efficiently infer haplotypes from genotypes of individuals related by a pedigree *without mating loops*, assuming that the hereditary process was free of mutations (*i.e.* the Mendelian law of inheritance) and recombinants. We model the haplotype inference problem as a system of linear equations as in [10] and present an (optimal) linear-time (*i.e.* $O(mn)$ time) algorithm to generate a *particular solution* * to the haplotype inference problem, where m is the number of loci (or markers) in a genotype and n is the number of individuals in the pedigree. Moreover, the algorithm also provides a *general solution* † in $O(mn^2)$ time, which is optimal because the size of a general solution could be as large as $\Theta(mn^2)$. The key ingredients of our construction are (i) a fast consistency checking procedure for the system of linear equations introduced in [10] based on a careful investigation of the relationship between the equations (ii) a novel linear-time method for solving linear equations without invoking the Gaussian elimination method. Although such a fast method for solving equations is not known for general systems of linear equations, we take advantage of the underlying loop-free pedigree graph and some special properties of the linear equations.

Keywords: haplotype inference, pedigree analysis, mating loop, tree-pedigree, linear-time algorithm, system of linear equation, general solution.

1. Introduction

In October 2002, a multi-country collaboration, namely, the international *HapMap* project was launched [5]. One of the main objectives of the HapMap project is to identify the haplotype structure of humans and common haplotypes among various populations. This information will greatly facilitate the mapping of many important disease-susceptible genes. However, due to the diploid structure of the human genome, in practice, genotype data, instead of haplotype data are collected routinely, especially in large-scale sequencing projects mainly to save experimental costs. Hence, combinatorial algorithms as well as 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 in the literature.

This paper is concerned with the inference of haplotypes from genotypes of individuals related by a *pedigree*, which describes the parent-offspring relation-

---

*A particular solution of any linear system is an assignment of numerical values to the variables in the system which satisfies the equations in the system.

†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. A general solution for ZRHC is very useful in practice because it allows the end user to efficiently enumerate all solutions for ZRHC and performs tasks such as random sampling.

ship among the individuals. Two biological assumptions will be made, namely, 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 *zero-recombinant principle* which says that genetic recombination is very rare or nonexistent for closely linked markers and thus in haplotype inference we prefer haplotype configurations with zero recombinants if they exist [3, 9]. The problem of inferring zero-recombinant haplotypes from given genotypes under the Mendelian law, where we would like to enumerate all haplotype solutions that require no recombinants (if such solutions exist), is called the *zero-recombinant haplotype configuration (ZRHC)* problem and was studied recently in [6, 10]. When the input pedigree is a tree pedigree with no mating loops (which is often true for human pedigrees), the ZRHC problem is called the *Loop-Free ZRHC* problem. 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 a more formal definition of the above biological concepts and computational problems).

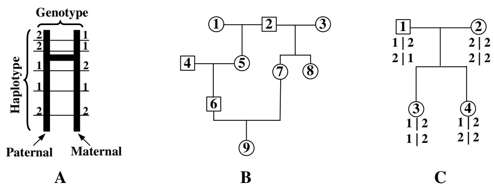

Fig. 1. (A). The structure of a pair of chromosomes from a mathematical point of view. In the figure, each numeric value (1 or 2) represents a marker state or an *allele*. The haplotype inherited from the father (or the mother) is called *paternal* haplotype (or *maternal* haplotype, respectively). The 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 without mating loops is called a *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*.

The Algorithmic Problem, Previous Results and Our Result. The ZRHC problem can also be stated abstractly as a simple inheritance reconstruction problem as follows [10]. We have a pedigree connecting n individuals where each individual j has a genotype given in the form of string $\{a_{j,1}, b_{j,1}\} \cdots \{a_{j,m}, b_{j,m}\}$ defined on m marker loci. We would like to reconstruct two haplotypes (*i.e.* strings) $a_{j,1} \cdots a_{j,m}$ and $b_{j,1} \cdots b_{j,m}$ for each individual j such that the inferred haplotypes obey the Mendelian law and the zero-recombinant principle. That is, suppose that haplotypes $a_{j,1} \cdots a_{j,m}$ and $b_{j,1} \cdots b_{j,m}$ are inherited from j's father j_1 and mother j_2 respectively, then we have $a_{j,1} \cdots a_{j,m} \in \{a_{j_1,1} \cdots a_{j_1,m},\ b_{j_1,1} \cdots b_{j_1,m}\}$ and $b_{j,1} \cdots b_{j,m} \in \{a_{j_2,1} \cdots a_{j_2,m},\ b_{j_2,1} \cdots b_{j_2,m}\}$.

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 $F(2)$ and applying Gaussian elimination [6]. 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 [5]. Recently, Xiao *et al.* [10] presented a much faster algorithm for ZRHC with running time $O(mn^2 + n^3 \log^2 n \log\log n)$ by introducing a new system of $O(mn)$ linear equations over $F(2)$ with mn variables and reducing it to an effectively equivalent system with only $O(n \log^2 n \log\log n)$ equations and at most $2n$ variables.

For Loop-Free ZRHC, the method proposed by Xiao *et al.* [10] runs in time $O(mn^2 + n^3)$ to produce a general solution and in time $O(mn + n^3)$ to produce a particular solution. Chan *et al.* [4] proposed a linear-time (*i.e.* $O(mn)$ time) algorithm to find a particular solution. It is unclear how the algorithm can be extended to produce general solutions efficiently. Moreover, the algorithm has four different stages and needs to maintain various consistency conditions (called SNP-consistency, Mendelian consistency, and endgame-consistency) and avoid some inconsistency conditions (such as the family problem) at these stages (see [4] for the details of the (in)consistency conditions). As a result, the analysis of the algorithm (concerning both correctness and complexity) is quite involved.

In this paper, we present a linear-time algorithm to generate a particular solution for Loop-Free ZRHC. Moreover, the algorithm can also provide a general solution in $O(mn^2)$ time, which is optimal because the size of a general solution could be as large as $\Theta(mn^2)$. [a] Our construction is based on the system of linear equations/constraints introduced in [10], a careful examination of the relationship between the constraints using the so called *constraint graphs* (to be defined later on), and a novel linear-time method for solving linear systems without invoking Gaussian elimination. In the algorithm, we first build a *one-to-one* mapping from the constraints to the edges in the pedigree graph, then solve the constraints by a single BFS traversal on the pedigree graph. We also give a rigorous proof of the correctness and tight analysis of the time complexity of our algorithm.

The rest of the paper is organized as follows. In section 2, we describe a system of linear equations for ZRHC and some useful graphs derived from the input pedigree introduced in [10] to make the paper self-contained. A linear-time algorithm to check if the linear equations derived from an input tree pedigree are consistent (*i.e.* if solutions for the linear system and thus ZRHC exist) and solve the linear system to attain a general solution is presented in section 3. The method of obtaining a particular solution by the algorithm is also described in this section. Some concluding remarks are given in section 4. Due to page limit, all proofs, some detailed biological definitions and an example execution of the main algorithm will be shown in the full version [1].

2. A System of Linear Equations and Equivalent Linear Constraints

In this section, we present a formulation of ZRHC in terms of linear equations and define some graph structures that will be used in our algorithm. The linear system and definitions were all introduced in [10], and are included here for the convenience of the reader.

2.1. *The Linear System*

Throughout this paper, n denotes the number of the individuals (or members) in the input pedigree and m denotes the number of marker loci. Without loss of generality,

[a]An instance of the Loop-Free ZRHC problem that has a general solution of size $\Theta(mn^2)$ is given in the appendix.

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 ZRHC [6, 10]), 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 2 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 [6] 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 j is a non-founder member with her father and mother being j_1 and j_2, respectively. We define $\mathbf{d}_{j_r,j} (r = 1, 2)$ as follows: $\mathbf{d}_{j_1,j}$ be the vector $\mathbf{0}$ and $\mathbf{d}_{j_2,j} = \mathbf{w}_j$.

Utilizing the above definitions, we can formally express the ZRHC problem as a system of linear equations (refer to [10] for the details of deriving the linear system) over $F(2)$:

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

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 h-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 (1). 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* (to be defined below), 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. [b]

[b]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.

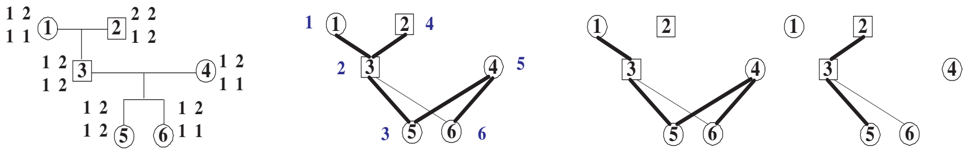

Fig. 2. (A). An example pedigree with genotype data. Here, the alleles at a locus are ordered according to their id numbers instead of phase (which is unknown). (B). The pedigree graph with a spanning tree. The tree edges are highlighted. Observe that there lies a *cycle* of length 4 in the given tree pedigree graph. The root of the tree is v_1. The number (in blue) next to a vertex is the index generated by a BFS traversal of the spanning tree. For instance, v_3 is v_2's predecessor in the BFS traversak where $index(v_3) = 2$ and $index(v_2) = 4$, while v_2 is v_3's biological father. (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. We also set an arbitrary vertex as the *root* of $\mathcal{T}(G)$. We start a BFS traversal on $\mathcal{T}(G)$ from the root, and index the vertices in E based on the order of their first-time occurrences in the traverse. For a vertex j, we denote by $index(j)$ the index of j. For instance, $index(r) = 1$ if vertex r is the root of $\mathcal{T}(G)$. If the last edge on the path from the root to vertex j is (k, j), k is called the *predecessor* of j, and denoted by $pred(j)$. [c] Obviously, $index(pred(k)) < index(j)$ for every non-root vertex j. $\mathcal{T}(G)$ 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 $|E| \leq 2n$ and the number of edges in $\mathcal{T}(G)$ is $n-1$, we have $|E^{\mathbb{X}}| \leq n+1$. Figure 2 (B) gives an example of tree edges, cross edges and an indexing on vertices.

For any fixed locus i, the value $w_k[i]$ can be viewed as the *label* 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 label 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 spanning 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 locus 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 $\mathbf{d}_{k,j} = \mathbf{d}_{j,k}$.

As discussed in [10], we can see that for any 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*. Formally, given a cycle $\mathcal{C} = j_0, \ldots, j_k, j_0$ in G_i, the constant b is defined as $b = \sum_{r=0}^{k} d_{j_r, j_{r+1 \mod k+1}}[i]$. We can easily verify $\sum_{r=0}^{k} h_{j_r, j_{r+1 \mod k+1}} = b$ [10].

Moreover, 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* [10]. Formally, given

[c] Conventionally, k is called the *father* of j in graph theory. In order to distinguish the concept of father in graph theory and the concept of biological father, we use the term *predecessor* in this paper.

a path $\mathcal{P} = j_0, \ldots, j_k$ in G_i connecting vertices j_0 and j_k, if the variables $p_{j_0}[i]$ and $p_{j_k}[i]$ are pre-determined, the binary constant b is defined as $b = p_{j_0}[i] + p_{j_k}[i] + \sum_{r=0}^{k-1} d_{j_r,j_{r+1}}[i]$. It is easy to verify that $\sum_{r=0}^{k-1} h_{j_r,j_{r+1}} = b$ [10]. Again, notice that the constant b does not depend on the direction that path $\mathcal{P}$ is read.

2.3. *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. 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. 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, which 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 $C^{\mathbb{C}} = \biguplus_{e \in E^{\mathbb{X}}} C^{\mathbb{C}}(e)$. [d]

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 as $\mathcal{P}$, contains e. Observe that both endpoints of $\mathcal{P}$ are pre-determined and the 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. 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 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. 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. 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.

[d]Here the operator $\uplus$ denotes disjoint union and is used to save running time in our algorithm.

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 $(k, j, b, e) = (j, k, b, e)$. We can easily see that $|C^{\mathbb{C}}| + |C^{\mathbb{P}}| + |C^{\mathbb{T}}| \leq 2mn$, since $|C^{\mathbb{P}}| + |C^{\mathbb{C}}| \leq m|E^{\mathbb{X}}| = m(n+1)$ and $|C^{\mathbb{C}}| \leq m(|V| - 1) = m(n-1)$. Moreover, the construction of the locus graphs, the constraints $C^{\mathbb{C}}$, $C^{\mathbb{P}}$ and $C^{\mathbb{T}}$ can be done in $O(mn)$ time (the detailed pseudo-code for the constructions is given in [10]).

3. A Linear-Time Algorithm for Solving Loop-Free ZRHC

As pointed out in [10], ZRHC has a solution if and only if the above linear system for the h-variables has a solution. Now we focus on how to solve the linear system to obtain solutions to the h-variables. We will check the consistency of the path/cycle constraints and tree constraints first, perform some transformation on the constraints to obtain an equivalent but smaller system of constraints, and then solve the h-variables for tree edges and cross edges with respect to the spanning tree $\mathcal{T}(G)$.

3.1. *Consistency Checking*

First, we transform the path/cycle constraints to eliminate redundancy. The tree constraints will be dealt with in a similar way.

Path/Cycle Consistency Checking and Redundancy Removal. Due to page constraint, we leave the proof of the following lemma and corollary in the full version [1]. The pseudo-code for path/cyle constraint consistency checking is given in Fig. 3.

Lemma 3.1. *Given the path constraint set $C^{\mathbb{P}}(e)$ and the cycle constraint cycle $C^{\mathbb{C}}(e)$ for a cross edge e in a tree pedigree, we can reduce them to an equivalent constraint set of size at most one, in $O(|C^{\mathbb{C}}(e) \uplus C^{\mathbb{P}}(e)|)$ time.*

As a natural extension of Lemma 3.1, we have the following corollary:

Corollary 3.1. *Given the path constraint set $C^{\mathbb{P}}$ and the cycle constraint cycle $C^{\mathbb{C}}$ for a tree pedigree, we can reduce them to an equivalent constraint set of size at most $|E^{\mathbb{X}}|$, in $O(|C^{\mathbb{C}} \uplus C^{\mathbb{P}}|)$ time.*

Tree Constraint Consistency Checking and Redundancy Removal. To clearly demonstrate the relationship among the constraints in $C^{\mathbb{T}}$, we define a *constraint graph* G^* as follows. [e] For each vertex in the pedigree graph, we create a vertex in G^*. For each tree constraint $(k, j, b, e_0) \in C^{\mathbb{T}}$, we introduce an edge connecting vertices k and j in G^* with weight b. We denote by $E(G^*)$ the edge set in G^*. The weight of a path $\mathcal{P}$ (or a cycle $\mathcal{C}$) in the constraint graph is defined as the summation (in $F(2)$) over the weights of the edges along path $\mathcal{P}$ (or cycle $\mathcal{C}$), which is denoted as $W(\mathcal{C})$ (or $W(\mathcal{P})$, respectively). We choose a spanning forest $\mathcal{T}(G^*)$ on the constraint graph G^*. In each connected component $\mathcal{T}$ of $\mathcal{T}(G^*)$, the vertex with the smallest index is designated as the root of $\mathcal{T}$. (We set the roots in this way to facilitate solving the h-variables in subsection 3.2). $\mathcal{T}(G^*)$ partitions the edge set $E(G^*)$ into two subsets: *tree edges* and *cross edges*. Conventionally, a cycle containing exactly one cross edge is called a *basic cycle*. Figure 5 illustrates an example constraint graph and its basic cycles.

[e]Note that a constraint graph can be a weighted multigraph.

Procedure Constraints_consistency
input: $C^{\mathbb{C}}$, $C^{\mathbb{P}}$, $C^{\mathbb{T}}$, and a fixed tree edge e_0
output: compact $C^{\mathbb{C}}, C^{\mathbb{P}}$ and $C^{\mathbb{T}}$

begin

Step 1. *Consistency checking for path/cycle constraints*

for each cross edge e
  Pick an arbitrary constraint, say (j, k, b, e), from $C^{\mathbb{C}}(e)$;
  if there exists a constraint $(j, k, b + 1, e) \in C^{\mathbb{C}}(e)$
    exit "Input genotypes are inconsistent.";
  $C^{\mathbb{P}} = \{(j, k, b, e)\}$;

for each cross edge e
  Pick an arbitrary constraint, say (b, e), from $C^{\mathbb{C}}(e)$;
  if there exists a constraint $(b+1,e) \in C^{\mathbb{C}}(e)$
    exit "Input genotypes are inconsistent.";
  $C^{\mathbb{C}} = \{(b, e)\}$;
  if $C^{\mathbb{P}} \neq \emptyset$ (*i.e.* $C^{\mathbb{P}}$ contains exactly one constraint)
    Suppose the constraint $\in C^{\mathbb{P}}$ is (j,k,b',e);
    $C^{\mathbb{T}} = \{(j, k, b' + b, e_0)\}$;
    $C^{\mathbb{P}} = \emptyset$;

Step 2. *Consistency checking for tree constraints*

Construct the constraint graph G^* for $C^{\mathbb{T}}$;
$\widehat{C} = \emptyset$;

for each connected component S of G^*
  Generate the spanning tree $\mathcal{T}(S)$ of S;
  Suppose vertex k_0 is the vertex with the smallest index;
  Let vertex k_0 be the root of $\mathcal{T}(S)$;

  Traverse $\mathcal{T}(S)$ by BFS starting from k_0;
  for each vertex $k \in S$
    $visited(k) = false$;
  $W(k_0) = 0$; $visited(k_0) = true$;
  for each tree edge (j,k) with respect to $\mathcal{T}(S)$
    Suppose the edge represents constraint (j, k, b, e_0)
    Suppose $visited(j)=true, visited(k)=false$;
    $W(k) = W(j) + b$; $visited(k) = true$;
    $\widehat{C} = \widehat{C} \uplus \{(k_0, k, W(k), e_0)\}$;

  for each non-tree edge (j, k) w.r.t. $\mathcal{T}(S)$
    Suppose the edge represents constraint (j, k, b, e_0)
    if $W[j] + W[k] + b = 1$ (*i.e.* $\exists$ a 1-weight basic cycle)
      exit "Input genotypes are inconsistent.";

  $C^{\mathbb{T}} = \widehat{C}$;

  return $C^{\mathbb{C}}, C^{\mathbb{P}}$ and $C^{\mathbb{T}}$

end.

Fig. 3. The procedure for checking the consistency among the constraints.

Algorithm Loop_Free_ZRHC_Phase
input: a tree pedigree $G = (V, E)$ and genotypes $\{\mathbf{g}_j\}$
output: a general solution of $\{\mathbf{p}_j\}$

begin

Step 1. *Preprocessing*
  Choose an arbitrary vertex r as the root of a spanning tree $\mathcal{T}(G)$ for the pedigree graph G;
  Index the vertices by a BFS traversal on $\mathcal{T}(G)$;

Step 2. *Constraint generation*
  e_0 is an arbitrary tree edge w.r.t. $\mathcal{T}(G)$;
  Generate $C^{\mathbb{C}}$, $C^{\mathbb{P}}$ and $C^{\mathbb{T}}$;

Step 2′. *Redundant Constraint Elimination*
  Constraints_Consistency($C^{\mathbb{C}}$, $C^{\mathbb{P}}$, $C^{\mathbb{T}}$, e_0);
  Define $C = C^{\mathbb{C}} \uplus C^{\mathbb{P}} \uplus C^{\mathbb{T}}$;

Step 3. *Solve the h-variables*
Construct the mapping $f: C \mapsto E'$ in Equ.(3);
Traverse G in BFS order;
$W[r] = 0$;
for each tree edge $(pred(k), k) \in E$
  if $(pred(k), k) \notin E'$
    Define $h_{pred(k),k}$ as a free variable;
    $W(k) = h_{pred(k),k} + W(pred(k))$;
  if $(pred(k), k) \in E'$
    Suppose $f^{-1}((pred(k),k)) = (j,k,b,e_0)$ $(index(j) < index(k))$;
    $h_{pred(k),k} = b + W(j) + W(pred(k))$;
    $W(k) = b + W(j)$;

for each cross edge $(i, j) \in E$
  if $(i, j) \notin E'$
    Define $h_{i,j}$ as a free variable;
  if $(i, j) \in E'$
    if $f^{-1}((i, j))$ is a path constraint, say, (k_1, k_2, b, e)
      $h_{i,j} = b + W(k_1) + W(k_2) + W(i) + W(j)$;
    else (*i.e.* $f^{-1}((i, j))$ is a cycle constraint, say,(b, e))
      $h_{i,j} = b + W(i) + W(j)$;

Step 4. *Solve* $\{\mathbf{p}_j\}$ *by propagation*
  for each locus i
    for each component $\mathcal{T}$ in $\mathcal{T}(G_i)$
      if $\mathcal{T}$ has no pre-determined vertices
        Set the seed's p-variable as a free variable and treat it as determined;
      Traverse $\mathcal{T}$ by BFS 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.

Fig. 4. The algorithm for solving the linear system for Loop-Free ZRHC.

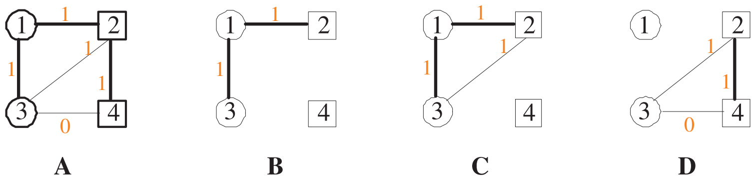

Fig. 5. A constraint graph and its basic cycles. (A). Given a set of tree constraints $C^{\mathbb{T}} = \{(v_1, v_2, 1, e_0), (v_1, v_3, 1, e_0), (v_2, v_3, 1, e_0), (v_2, v_4, 1, e_0), (v_3, v_4, 0, e_0)\}$, we can construct the constraint graph G^* shown in (A), where the number on an edge in G^* is the weight of the edge, *i.e.* the b-constant in the constraint represented by the edge. For example, $(v_1, v_3, 1, e_0)$ is represented by edge (v_1, v_3) with weight 1 in the constraint graph G^*. We choose a spanning tree $\mathcal{T}(G^*)$, where the tree edges of $\mathcal{T}(G^*)$ are highlighted. (B). A path connecting v_2 and v_3 on the spanning tree $\mathcal{T}(G^*)$. The weight of the path (*i.e.* the summation over weights of the edges along the path) is 0. (C). A cycle consisting of v_1, v_2 and v_3 with weight 1. The cycle contains only one cross edge in $E(G^*)$, and is thus a basic cycle. (D). A cycle consisting of v_2, v_3 and v_4 with weight 0. The cycle contains two cross edges in $E(G^*)$, and is thus not a basic cycle.

In the rest of this subsection, we will consider some basic properties of the constraint graph, which will be used to check the consistency on tree constraints. The proofs of the following lemmas are given in the appendix.

Lemma 3.2. *For any three vertices i, j and k in a connected component of the spanning forest $\mathcal{T}(G^*)$, the weight of the path connecting vertices j and k is equal to the total weight of the path connecting vertices i and j and the path connecting vertices i and k.*

Lemma 3.3. *In a constraint graph, the weight of every cycle can be represented as a summation over the weights of some basic cycles.*

The next corollary follows from Lemma 3.3 easily.

Corollary 3.2. *In a constraint graph, every cycle has weight 0 if and only every basic cycle has weight 0.*

Now we present our key lemma, which suggests how to use the constraint graph constructed from $C^{\mathbb{T}}$ to check the consistency among the constraints in $C^{\mathbb{T}}$.

Lemma 3.4. *$C^{\mathbb{T}}$ has a solution if and only if every cycle in the corresponding constraint graph has weight 0.*

Corollary 3.2 and Lemma 3.4 imply the following theorem.

Theorem 3.1. *$C^{\mathbb{T}}$ has a solution if and only if every basic cycle in the corresponding constraint graph has weight 0.*

Therefore, we can check the consistency of $C^{\mathbb{T}}$ by detecting the existence of 1-weight basic cycles. Notice that every basic cycle contains exactly one cross edge in $E(G^*)$. We can calculate the weights of basic cycles by first calculating the weight of the path connecting every pair of vertices in each connected component of the spanning forest $\mathcal{T}(G^*)$, and then checking if the basic cycle induced by each cross edge has weight 0. By Lemma 3.2, in every connected component $\mathcal{T}$ of the forest $\mathcal{T}(G^*)$, the weight of the path connecting each pair of vertices can be obtained from the weights of the paths connecting the root of $\mathcal{T}$ and each of involved vertices. In the following, we denote by $W(k)$ the weight of the path between the root and a vertex k in a connected component of $\mathcal{T}(G^*)$.

The detailed pseudo-code for checking the consistency of $C^{\mathbb{T}}$ is given in Figure 3. Besides consistency checking, the procedure also generates an equivalent constraint set $\widehat{C}$ for $C^{\mathbb{T}}$ based on $\mathcal{T}(G^*)$, which is defined as follows,

$$\widehat{C} = \left\{ (k_0, k, W(k), e_0) \,\middle|\, \begin{array}{l} k \in V,\ k_0 \text{ is the root of the connected component} \\ \text{containing } k \text{ in } \mathcal{T}(G^*), \text{ and } k \neq k_0 \end{array} \right\} \tag{2}$$

Clearly, $|\widehat{C}| \leq |V| - 1 = n - 1$. This discussion is summarized formally in the following lemma. [f]

Lemma 3.5. *Given the tree constraint set $C^{\mathbb{T}}$, we can reduce it to an equivalent constraint set of size at most $(n-1)$ in $O(|C^{\mathbb{T}})|$ time.*

The following corollary holds from Corollary 3.1, Lemma 3.5, and the fact $|C^{\mathbb{C}}| + |C^{\mathbb{P}}| + |C^{\mathbb{T}}| \leq 2mn$.

Corollary 3.3. *Given the linear constraint set $C = C^{\mathbb{C}} \uplus C^{\mathbb{P}} \uplus C^{\mathbb{T}}$ for a tree pedigree, we can reduce it to an equivalent constraint set of size at most $|E|$ in $O(mn)$ time, where E is the set of the edges in the pedigree graph and $|E| \leq 2n$.*

3.2. *Solving the h-Variables Efficiently*

Now we return to the pedigree graph G and show how to solve the h-variables for tree edges and for cross edges separately. Our idea is as follows. We first construct a one-to-one mapping from the constraints to the edges in the pedigree graph, and then assign values to the h-variables consistent with the constraints by a BFS traversal on the pedigree graph.

Constructing a Mapping from Constraints to Edges. We want to construct a mapping f from the (reduced) constraints $C = C^{\mathbb{C}} \uplus C^{\mathbb{P}} \uplus C^{\mathbb{T}}$ to a subset $E' \subseteq E$ of the edges satisfying the following two conditions:

- *Condition 1:* For every tree/path constraint (or every cycle constraint), there exists exactly one edge of E' on the path (or the cycle, respectively) corresponding to the constraint.
- *Condition 2:* f is one-to-one.

We define the mapping $f : C \mapsto S'$ as follows, where $C = C^{\mathbb{T}} \uplus C^{\mathbb{P}} \uplus C^{\mathbb{C}}$ and the subset S' is implied by the definition.

$$f(c) = \begin{cases} (pred(k), k) & \text{if } c = (j, k, b, e_0) \in C^{\mathbb{T}} \text{ and } index(j) < index(k) \ (i.e.\ c \text{ is a tree constraint}) \\ (i, j) & \text{if } c = (k_1, k_2, b, e) \in C^{\mathbb{P}}, \text{ where } e = (i, j) \ (i.e.\ c \text{ is a path constraint}) \\ (i, j) & \text{otherwise } c = (b, e) \in C^{\mathbb{C}}, \text{ where } e = (i, j) \ (i.e.\ c \text{ is a cycle constraint}) \end{cases} \tag{3}$$

Clearly, the mapping for every path (or cycle) constraint satisfies condition 1. Consider a tree constraint $(j, k, b, e_0) \in C^{\mathbb{T}}$ and suppose that $index(j) < index(k)$. Because we construct the indexing from top to bottom in $\mathcal{T}(G)$, vertex j must be at the same or a higher level than vertex k. Hence, $(pred(k), k)$ must be on the path connecting vertices j and k in $\mathcal{T}(G)$. As a result, condition 1 holds for the mapping f. The proof of the mapping f is one-to-one is omitted here and can be found in the full version [1].

[f] A similar lemma is given in [10] too.

Once the mapping f is defined, we can solve the constraints as follows. We set each h-variable corresponding to an edge in $E - E'$ as a free variable, and represent the h-variable corresponding to an edge in E' as a linear combination of the free variables based on the constraint mapped to the edge so that the constraint is satisfied.

Assigning Values to the h-Variables We need assign values to the h-variables so that all the path/cycle/tree constraints are satisfied. Given a vertex j, we denote by $W[j]$ the sum of the h-variables along the path between the root of the spanning tree $\mathcal{T}(G)$ and vertex j. We will treat h-variables for tree edges and cross edges separately, utilizing the mapping f.

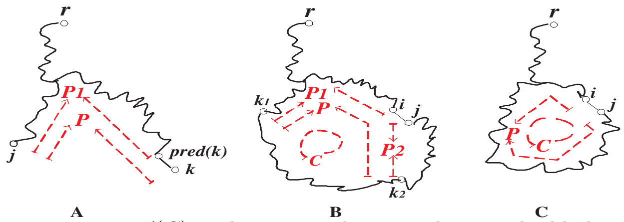

Fig. 6. The spanning tree $\mathcal{T}(G)$ with root r. The tree edges are highlighted. (A) The path $\mathcal{P}$ connecting vertices j and k on $\mathcal{T}(G)$ for a given tree constraint (j, k, b, e_0), and the path $\mathcal{P}_1$ connecting vertices j and $pred(k)$. (B) The cycle $\mathcal{C}$ induced by adding $e = (i, j)$ to the spanning tree $\mathcal{T}(G)$ for given a path constraint (k_1, k_2, b, e), and the paths $\mathcal{P}$, $\mathcal{P}_1$, and $\mathcal{P}_2$ connecting vertices k_1 and k_2, vertices k_1 and i, and vertices j and k_2, respectively. Notice that $\mathcal{P}_1$ and $\mathcal{P}_2$ do not contain e while $\mathcal{P}$ does. (C) The cycle $\mathcal{C}$ induced by adding $e = (i, j)$ to the spanning tree $\mathcal{T}(G)$ for a given cycle constraint (b, e) and the path $\mathcal{P}$ connecting the vertices i and j. Note that $\mathcal{P}$ does not contain e.

- **Handling h-variables for tree edges.** We solve the h-variables by a BFS traversal on $\mathcal{T}(G)$ starting from the root r. First, we set $W[r] = 0$. If we encounter a tree edge $(pred(k), k)$ in $(E - E')$, we set $h_{pred(k),k}$ as a free variable, and set $W(k) = h_{pred(k),k} + W(pred(k))$. Otherwise, we encounter a tree edge $(pred(k), k)$ in E' (assuming $f^{-1}((pred(k), k)) = (j, k, b, e_0)$ and $index(j) < index(k)$), and we set $h_{pred(k),k}$ as:
$$h_{pred(k),k} = b + W(j) + W(pred(k)) \tag{4}$$
Note that since $index(j) < index(k)$ and $index(pred(k)) < index(k)$, $W(j)$ and $W(pred(k))$ are known at this time. We also set $W(k) = b + W(j)$. An illustration of the vertices j, $pred(k)$ and k is given in Figure 6(A). After the BFS traversal, we determine $W[j]$ for every vertex j. We will use $W[j]$ to solve the h-variables for cross edges.
- **Handling h-variables for cross edges:** If we encounter a cross edge $(i, j) \in (E - E')$, we set $h_{i,j}$ as a free variable. Otherwise, we encounter a cross edge $(i, j) \in E'$, and have $|C^{\mathbb{C}}(e) \uplus C^{\mathbb{P}}(e)| = 1$ based on Lemma 3.3 and Equation (3). We assign $h_{i,j}$ as follows.
  - *Case 1:* $C^{\mathbb{C}}(e) \uplus C^{\mathbb{P}}(e)$ contains a path constraint (k_1, k_2, b, e). We set
  $$h_{i,j} = b + W(k_1) + W(k_2) + W(i) + W(j) \tag{5}$$
  An illustration of the vertices k_1, k_2, i and j is demonstrated in Figure 6(B).

– *Case 2:* $C^{\mathbb{C}}(e) \uplus C^{\mathbb{P}}(e)$ contains a cycle constraint (b, e). We set

$$h_{i,j} = b + W(i) + W(j) \tag{6}$$

An illustration of the vertices i and j is given in Figure 6(C).

Figure 4 displays a pseudo-code of the above procedure. The algorithm is called **Loop_Free_ZRHC_Phase**, which generates a general solution. Note that if we assign a particular binary value to each free variable in the algorithm, the algorithm obtains a particular solution. The correctness of the algorithm is given in the following lemma.

Lemma 3.6. *The algorithm* **Loop_Free_ZRHC_Phase** *correctly solves the ZRHC problem on a tree pedigree.*

Theorem 3.2. *The algorithm* **Loop_Free_ZRHC_Phase** *produces a particular solution to the ZRHC problem on a tree pedigree in $O(mn)$ time, and a general solution in $O(mn^2)$ time.*

4. Concluding Remarks

It still remains open if one could extend the algorithm **Loop_Free_ZRHC_Phase** to solve the ZRHC problem on a general pedigree in linear time.

Acknowledgement

We would like to thank Jing Xiao, Marek Chrobak and Qi Fu for many useful discussions. The research is supported in part by NSF grant CCR-0309902, NIH grant LM008991-01, NSFC grant 60528001, and the Changjiang Visiting Professorship at Tsinghua University.

References

[1] The full version of this paper is available at `http://www.cs.ucr.edu/~lliu/paper/tree_zrhc_full.ps`
[2] 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.
[3] J. R. O'Connell. Zero-recombinant haplotyping: applications to fine mapping using SNPs. *Genet. Epidemiol.*, 19 Suppl 1:S64–70, 2000.
[4] 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 (WABI'06)*, 56-67 ,2006.
[5] The international HapMap Consortium. International HapMap Project. *Nature*, 426:789-796, 2003.
[6] J. Li and T. Jiang. Efficient rule-Based haplotyping algorithm for pedigree data. *Proc. of 7th Annual Conference on Research in Computational Molecular Biology (RECOMB'03)*, 197-206, 2003.
[7] J. Li and T. Jiang. An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming. *Proc. of RECOMB'04*, 20-29, 2004.
[8] J. Li and T. Jiang. Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by Integer Linear Programming. *J. of Computational Biology* 12(6), 719-739, 2005.
[9] D. Qian and L. Beckmann. Minimum-recombinant haplotyping in pedigrees. *Am. J. of Hum. Genet.*, 70(6):1434–1445, 2002.
[10] J. Xiao, L. Liu, L. Xia, T. Jiang. Fast Elimination of Redundant Linear Equations and Reconstruction of Recombination-Free Mendelian Inheritance on a Pedigree. *Proc. of 18th Annual ACM-SIAM Symoposium on Discrete Algorithms (SODA'07)*, 655-664, 2007.