Linear-Time Haplotype Inference on Pedigrees without Recombinations and Mating Loops

Mee Yee Chan[*] Wun-Tat Chan[†] Francis Y. L. Chin[‡*] Stanley P. Y. Fung[§]
Ming-Yang Kao[¶]

January 9, 2008

Abstract

In this paper, an optimal linear-time algorithm is presented to solve the haplotype inference problem for pedigree data when there are no recombinations and the pedigree has no mating loops. The approach is based on the use of graphs to capture SNP, Mendelian and parity constraints of the given pedigree. This representation allows us to capture the constraints as the edges in a graph, rather than as a system of linear equations as in previous approaches. Graph traversals are then used to resolve the parity of these edges, resulting in an optimal running time.

1 Introduction

The modeling of human genetic variation is critical to the understanding of the genetic basis for complex diseases. *Single nucleotide polymorphisms* (SNPs) [5] are the most frequent form of this variation, and it is useful to analyze *haplotypes*, which are sequences of linked SNP genetic markers (small segments of DNA) on a single chromosome. In diploid organisms, such as humans, chromosomes come in pairs, and experiments often yield *genotypes*, which blend haplotypes for the chromosome pair. This gives rise to the problem of inferring haplotypes from genotypes.

Before defining our problem, some preliminary definitions are needed. The physical position of a marker on a chromosome is called a *locus* and its state is called an *allele*. Without loss of generality, the allele of a *biallelic* SNP can be denoted by 0 and 1, and a haplotype with m loci is represented as a length-m string in $\{0,1\}^m$, and a genotype as a length-m string in $\{0,1,2\}^m$. Haplotype pair $\langle h_1,\ h_2\rangle$ is *SNP-consistent* with genotype g if where the two alleles of h_1 and h_2 are the same at the same locus, say 0 (respectively, 1), the corresponding locus of g is also 0 (respectively, 1), which denotes a *homozygous* locus; otherwise, where the two alleles of h_1 and h_2 are different, the corresponding locus of g is 2, which denotes a *heterozygous* locus (i.e., SNP). A genotype with s heterozygous loci can have 2^{s-1} SNP-consistent haplotype solutions. For example, genotype $g = 012212$ with $s = 3$ has four SNP-consistent haplotype pairs: $\{\langle 01\underline{11}1\underline{1},\ 01\underline{00}1\underline{0}\rangle$, $\langle 01\underline{11}1\underline{0},\ 01\underline{00}1\underline{1}\rangle$, $\langle 01\underline{10}1\underline{1},\ 01\underline{01}1\underline{0}\rangle$, $\langle 01\underline{10}1\underline{0},\ 01\underline{01}1\underline{1}\rangle\}$.

[*]Department of Computer Science, University of Hong Kong, Hong Kong. {mychan, chin}@cs.hku.hk
[†]Department of Computer Science, King's College, University of London, London, UK. joseph.chan@kcl.ac.uk
[‡]Supported by Hong Kong RGC Grant HKU-7119/05E and HKU Strategic Research Team Fund.
[§]Department of Computer Science, University of Leicester, Leicester, UK. pyfung@mcs.le.ac.uk
[¶]Department of Electrical Engineering and Computer Science, Northwestern University, USA. kao@northwestern.edu

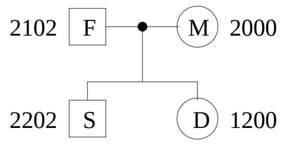

Figure 1: Example of a pedigree with 4 nodes.

A *pedigree* is a fundamental connected structure used in genetics. Figure 1 shows the pictorial representation of a pedigree with 4 nodes, with a square representing a male node and a circle representing a female node and children placed under their parents: in particular, a *father* (node F), a *mother* (node M) and two *children* (son node S and daughter node D). Each node in the pedigree is associated with a genotype. In Figure 1, for example, 2102 is the genotype for F and 2000 is the genotype for M. We assume that there are no *mating loops*, i.e., the pedigree does not contain loops. For example, marriage between descendants of a common ancestor forms a mating loop. However, polygamy or remarriage is allowed in the sense that step-children can exist. A precise definition of a mating loop will be given in Section 2. Note that mating loops are rare in real data sets, especially for humans [2].

A *consistent haplotype configuration* (with no recombinations) for a given pedigree is an assignment of a pair of haplotypes to each individual node such that (i) all the haplotype pairs are SNP-consistent with their corresponding genotypes and (ii) the haplotypes of each child are *Mendelian-consistent*, i.e., one of the child's haplotype is exactly the same as one of its father's and the other is the same as one of its mother's.

Haplotyping Pedigree Data (with No Recombinations) Problem (HPD-NR): Given a pedigree P where each individual node of P is associated with a genotype, find a consistent haplotype configuration (CHC) for P. □

Wijsman [7] proposed a 20-rule algorithm, and O'Connell [4] described a genotype elimination algorithm, both of which can be used for solving the HPD-NR problem. Li and Jiang [2] formulated the problem as a system of linear equations with $O(mn)$ equations and $O(mn)$ variables, where n is the number of individuals in the pedigree and m is the number of loci for each individual. The equations are then solved by Gaussian elimination. This gives a $O(m^3n^3)$ time algorithm. Xiao, Liu, Xia and Jiang [8] later improved the time complexity to $O(mn^2 + n^3 \log^2 n \log\log n)$. For the case without mating loops, their algorithm runs in $O(mn^2 + n^3)$ time.

It has long been conjectured that an $O(mn)$ time algorithm exists, but it should be appreciated that finding such an algorithm has been elusive and far less straightforward than many researchers have initially thought.

In this paper, we propose a new 4-stage algorithm that can either find a CHC solution or report "no solution" in optimal $O(mn)$ time when the pedigree has no mating loops. The main idea of our algorithm is to construct a tree to model the predigree. Each vertex in the tree represents a haplotype, i.e., each genotype corresponds to a pair of vertices. Different types of edges are added between the nodes to enforce the SNP- and Mendelian-consistencies. This is carried out gradually in Stages 1 and 2 in our algorithm, and Stage 3 is to add more edges to unify vertices of the same haplotype so that a connected tree is formed. The main difficulty in our approach is to resolve the correct alleles at the heterozygous loci in the CHC solution. We have developed a routine to resolve the heterozygous loci for tree vertices in a connected component, which is executed in Stages 2, 3 and 4. As a connected tree is formed after Stage 3, either a CHC solution is constructed or "no solution" is reported. This approach allows us to find a CHC solution without (directly) solving a system of linear equations, as in previous approaches [2, 8].

2 Preliminaries

Definition 1 (Pedigree graph and mating loop) *A* pedigree graph *is a graph derived naturally from the pedigree as follows. Each individual in the pedigree becomes a node in the graph. Whenever two individuals mate and produce children, there is an additional* mating node, *and there are undirected edges connecting the mating node to the two parents as well as to each of the children. A* mating loop *is a cycle in the pedigree graph, and a pedigree does not have a mating loop if the associated pedigree graph does not have cycles, i.e., the pedigree graph is a tree. A* trio *consists of a father, a mother and one of their children. A* nuclear family *consists of a father, a mother and all of their (shared) children.* □

We assume without loss of generality that the pedigree, and hence the pedigree graph, is connected.

Definition 2 (Family problem) *If there exists a family with father F, mother M and two children C_1 and C_2 in the pedigree and two loci i and j such that i and j are heterozygous in F, M and C_1 but are homozygous and heterozygous, respectively, in C_2, then we say that the pedigree has a* family problem*.* □

Figure 2 gives a simple example of a pedigree with a family problem. It can be easily checked that this, and any other pedigree with a family problem, has no CHC solution.

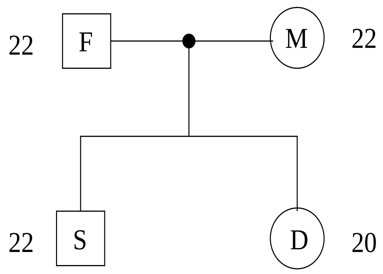

Figure 2: Pedigree with a family problem.

For each trio T, we define $het(T)$ as the set of all loci that are heterozygous for the father, the mother and the child in T, and $hom(T)$ as the set of all loci that are heterozygous for the father and the mother but homozygous for the child. These two sets for all trios can be computed easily in $O(mn)$ time.

Consider a nuclear family, which consists of a number of trios. The following observation is crucial: the nuclear family has no family problem if and only if for any two trios T_i, T_j in the family, $het(T_i)$ and $het(T_j)$ are either identical or disjoint. Note that $het(T_i) \cup hom(T_i) = het(T_j) \cup hom(T_j)$. Using this observation, we can check the pedigree for family problems in $O(mn)$ time as follows.

Lemma 1 *The family problems in the pedigree can be identified in $O(mn)$ time.*

Proof. Consider each nuclear family in the pedigree. We maintain sets $S_0, S_1, \ldots$ of trios where trios belonging to the same S_i have identical $het()$'s and trios belonging to different S_i's have disjoint $het()$'s. We extend the definition of $het()$ and $hom()$ to S_i naturally: $het(S_i)$ is the set of loci in $het(T)$ for $T \in S_i$, and $hom(S_i)$ is similarly defined. S_0 is a special set of trios with empty $het()$, and initially $S_1 = \{T_1\}$ where T_1 is a trio with non-empty $het()$. We then consider each trio one by one, and do the following:

For each new trio T,

(a) If $het(T)$ is empty, add T to S_0 and go to next T.

(b) Otherwise, for each S_i, $i \geq 1$,

   (i) If some loci in $het(S_i)$ are in $hom(T)$ but some are in $het(T)$, report *"family problem"* and halt.

   (ii) Else if all loci in $het(S_i)$ are in $hom(T)$, go to next i. If this is the last i, create a new S_j with T as a member, and go to next T.

   (iii) Else (all loci in $het(S_i)$ are in $het(T)$) check whether all other loci in T are homozygous. If this is false, then report *"family problem"* and halt. If this is true, add this T as a member of S_i, and skip to next T (no need to test the other S_j's).

The processing of each trio takes $O(m)$ time even though it may need to compare with all S_i (and there can be $O(n)$ of them), because the running time of steps (b)(i) and (b)(ii) is $O(|het(S_i)|)$, and the sum of all $|het(S_i)|$ is at most m. Thus the checking of family problems over the entire pedigree takes $O(mn)$ time. □

3 The Algorithm

3.1 Stage 1 – Setting Up the Local Graph G

Stage 1A – Checking for family problems: Our algorithm begins by checking for family problems. Only if there are no family problems will the algorithm continue; otherwise, "no solution" is reported. □

Stage 1B – Generating vector-pairs: For each trio in the given pedigree, let the respective genotypes of the father F, the mother M and the child C be: $x_1x_2 \ldots x_m$ and $y_1y_2 \ldots y_m$ and $z_1z_2 \ldots z_m$ where $x_i, y_i, z_i \in \{0, 1, 2\}$. We determine a pair of vectors (or vector-pair) each for the father, the mother and the child, namely: $\langle f_1, f_2\rangle$, $\langle m_1, m_2\rangle$ and $\langle c_1, c_2\rangle$, respectively, where $f_1 = x_{1,1}x_{1,2} \ldots x_{1,m}$ and $f_2 = x_{2,1}x_{2,2} \ldots x_{2,m}$; $m_1 = y_{1,1}y_{1,2} \ldots y_{1,m}$ and $m_2 = y_{2,1}y_{2,2} \ldots y_{2,m}$; $c_1 = z_{1,1}z_{1,2} \ldots z_{1,m}$ and $c_2 = z_{2,1}z_{2,2} \ldots z_{2,m}$. The vector-pairs are determined in the following manner.

1. For each locus i, for f_1 and f_2:

   (a) If $x_i = 0$ then $x_{1,i} = x_{2,i} = 0$.

   (b) If $x_i = 1$ then $x_{1,i} = x_{2,i} = 1$.

   (c) If $x_i = 2$ and $z_i = 0$ then $x_{1,i} = 0$ and $x_{2,i} = 1$.

   (d) If $x_i = 2$ and $z_i = 1$ then $x_{1,i} = 1$ and $x_{2,i} = 0$.

(e) If $x_i = 2$ and $z_i = 2$ and $y_i = 0$ then $x_{1,i} = 1$ and $x_{2,i} = 0$.

(f) If $x_i = 2$ and $z_i = 2$ and $y_i = 1$ then $x_{1,i} = 0$ and $x_{2,i} = 1$.

(g) If $x_i = 2$ and $z_i = 2$ and $y_i = 2$ then $x_{1,i} = ?$ and $x_{2,i} = ?$.

2. m_1 and m_2 are similarly determined.

3. Set $\langle c_1, c_2 \rangle = \langle f_1, m_1 \rangle$. Check that $\langle c_1, c_2 \rangle$ is consistent with C's genotype $z_1 z_2 \ldots z_m$; otherwise, report *"no solution."* □

The vector-pairs are the initial assignment of haplotypes assuming that C inherits f_1 from F and m_1 from M, i.e., $\langle c_1, c_2 \rangle = \langle f_1, m_1 \rangle$. For example, in Step 1(c), $z_i = 0$ and hence $z_{1,i} = z_{2,i} = 0$. Since c_1 is inherited from f_1, we can conclude $x_{1,i} = 0$ and therefore $x_{2,i} = 1$. When there is not enough information to deduce the value of a locus in the haplotype, a ? is used.

Observe that if a particular node N in the pedigree belongs to k different trios, then k vector-pairs, or $2k$ vectors, will be created for N in Stage 1B. These need to be unified eventually as a single vector-pair, because there is only one pair of haplotype for each node. This will be handled later when Endgame-consistency is defined, but first notice that we can define SNP-consistency and Mendelian-consistency in terms of vector-pairs. Let $\Phi(\mathrm{N})$ be the multiset comprised of these k vector-pairs of a node N. It is sometimes convenient to refer to the vectors rather than the vector-pairs. Thus, we let $\Gamma(\mathrm{N})$ be the multiset of $2k$ vectors, containing the two vectors of each vector-pair in $\Phi(\mathrm{N})$.

Definition 3 (SNP-Consistency Condition) SNP-consistency is said to be maintained *if and only if, for all nodes N in the pedigree, each vector-pair in Φ(N) is SNP-consistent with N's genotype. Vector-pair $\langle h_1, h_2 \rangle$ is said to be* SNP-consistent *with genotype g if h_1 and h_2 are both 0 (respectively, 1) at the same locus, then the corresponding locus of g is also 0 (respectively, 1); otherwise, if h_1 is 0 (respectively, 1) and h_2 is 1 (respectively, 0) at the same locus, then the corresponding locus of g is 2.* □

Definition 4 (Mendelian-Consistency Condition [1, 6]) Mendelian-consistency is said to be maintained *if and only if, for all nodes N in the pedigree, if N is a child in a trio comprised of F, M and N, then Φ(N) contains a vector-pair $\langle c_1, c_2 \rangle = \langle f_1, m_1 \rangle$ where $f_1 \in \Gamma$(F) and $m_1 \in \Gamma$(M).* □

Stage 1C – Constructing the local graph $G = (V, E)$: Let V be the multiset of all the vectors created in Stage 1B, and let E be the set of *red* and *brown* edges defined below:

1. A **red** edge is introduced to join the two vectors of each vector-pair generated in Stage 1A and indicates that a ? appearing at locus i of both vectors must be resolved differently in the later stages of the algorithm (the two vectors can be different or the same at the other non-? locus positions). The red edges enforce SNP-consistency.

2. For each F-M-C trio, let $\langle f_1, f_2 \rangle$, $\langle m_1, m_2 \rangle$ and $\langle c_1, c_2 \rangle$ be vector-pairs in Φ(F), Φ(M) and Φ(C), respectively, associated with this trio. Two **brown** edges are introduced, one connecting c_1 and f_1, and the other connecting c_2 and m_1. A brown edge between two vectors means that the two vectors must be the same at *all* locus positions. The brown edges enforce Mendelian-consistency. □

Example 1: Consider the pedigree with F (father), M (mother), S (son) and D (daughter) shown in Figure 1. Stage 1 produces the graph G in Figure 3 with 12 vertices and 10 edges (6 red and 4 brown), comprised of two connected components, one for each of the two trios, F-M-S and F-M-D, in the pedigree.

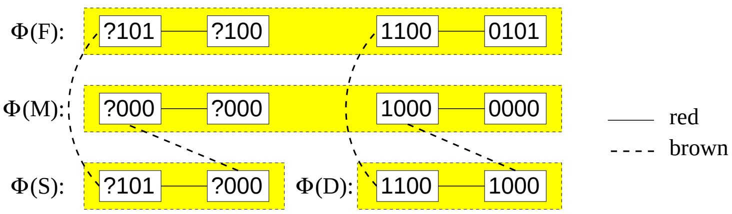

Figure 3: Graph G for Example 1.

Definition 5 *For any locus i in a connected component $\mathcal{G}$ of G, we say*

1. *Locus i is* resolved *in $\mathcal{G}$ if and only if all vectors in $\mathcal{G}$ have 0 or 1 at locus i.*
2. *Locus i is* unresolved *in $\mathcal{G}$ if and only if all vectors in $\mathcal{G}$ have ? at locus i.*
3. *Otherwise, locus i is* mixed *(it is a mix of ? and non-? at i).* □

In Example 1, the connected component for trio F-M-S has one unresolved locus (locus 1) and three resolved loci (loci 2, 3 and 4). Meanwhile, the component for trio F-M-D has no unresolved loci and four resolved loci (loci 1, 2, 3 and 4).

Lemma 2 *The time complexity of Stage 1 (Stages 1A, 1B and 1C) is $O(mn)$, where n is the number of nodes in the pedigree and m is the number of loci in each genotype. Furthermore, after Stage 1, all loci are either resolved or unresolved in each connected component of G (no mixed loci), and each connected component has six vertices. G has $O(n)$ vertices and edges and is acyclic.*

Proof. Checking for family problems takes $O(mn)$ time (Lemma 1). With $O(n)$ trios and six vectors generated per trio with each vector having m loci, Stage 1B takes $O(mn)$ time. With $O(n)$ trios and six vertices and five edges (three red and two brown) per trio introduced in G, Stage 1C takes $O(n)$ time altogether, and furthermore, G has $O(n)$ vertices and edges. G is acyclic because each connected component of G is a path consisting of 6 vectors, 3 red edges, and 2 brown edges of a trio. □

In later stages of our algorithm, no vector-pairs will be added to or deleted from each Φ(N), and all loci resolved in Stage 1 will remain unchanged. Components of G will later be merged with the addition of **green** (added in Stage 2) or **white** (added in Stage 3) edges until G becomes a single connected component. Before we explain why and how these edges are added, we first note that these new edges can always be added between two vectors in the same Γ(N). This structured way of adding edges to make G connected is possible given Lemma 3 below.

Lemma 3 *If G has more than one connected component, then there exists a node N such that there are two vector-pairs in Φ(N) which belong to two different connected components.*

Proof. Suppose to the contrary that, for all N, the vector-pairs in Φ(N) are all connected. We make use of the fact that the brown edges in G preserve the connectivity of any two nodes in the pedigree, which we have assumed to be connected. Therefore, if vector-pairs in Φ(N) are all connected for all N, then all vectors are connected together in a single connected component, which contradicts the assumption that G has more than one connected component. □

There are two reasons why we need to merge the connected components of G. First, each multiset Φ(N) may contain more than one vector-pairs; precisely, it contains k vector-pairs if N belongs to k different trios. However, by the time all loci are resolved, for all nodes N, each multiset Φ(N) must contain k copies of one unique vector-pair $\langle h_1, h_2\rangle$, which represents the haplotype-pair in a CHC for N. The green and white edges enforce this constraint by connecting vectors in Γ(N) that are supposed to be identical (because they have identical values at some resolved heterozygous loci). For example, consider two vector-pairs $\langle u_1, u_2\rangle$, $\langle v_1, v_2\rangle$ of a node N. If at a heterozygous locus i, u_1 is 0 and v_1 is also 0, then we know u_1 must be identical to v_1 (and u_2 identical to v_2). However, if there is another heterozygous locus j where u_1 is 0 and v_1 is 1, then it is impossible to give a unique vector-pair, and there is no CHC solution. We capture this observation by defining the following type of consistency:

Definition 6 (Endgame-Consistency Condition) *Vector-pairs $\langle u_1, u_2\rangle$, $\langle v_1, v_2\rangle \in \Phi(N)$ of a node N are said to be* Endgame-inconsistent *if the vector values at some heterozygous loci i and j ($i \neq j$) for u_1, u_2, v_1 and v_2 are a permutation of the four possibilities: 00, 01, 10 and 11; and* Endgame-consistent *otherwise. The node N is said to be* Endgame-inconsistent *if there exist vector-pairs in $\Phi(N)$ that are Endgame-inconsistent; and* Endgame-consistent *otherwise.* Endgame-consistency is said to be maintained *if and only if, for all nodes N in the pedigree, N is Endgame-consistent.* □

The second reason of adding green and white edges to connect the components of G is to further resolve the unresolved loci of each connected component of G with SNP-consistency and Mendelian-consistency maintained: vectors connected by green or white edges are supposed to be identical, and if a locus is resolved in one of the connected components but not in the other, the new connection allows us to resolve the locus in the other connected component as well. In the next subsection we will develop a procedure for resolving the values of loci in a connected component of G.

Our algorithm achieves a solution if, at the end of Stage 4, (a) graph G comprises a single connected component; (b) all loci are resolved in G; and (c) SNP-consistency, Mendelian-consistency and Endgame-consistency are maintained. However, our algorithm may report "no solution" if some N is Endgame-inconsistent before the end of Stage 4.

3.2 Stage 2 – Adding Green Edges

One of the most important aspects of our algorithm is that, at all stages, we maintain the property that each connected component of G has only resolved and unresolved loci (i.e., no mixed loci). In order to do this, we make extensive use of a subroutine called LOCUS_RESOLVE. LOCUS_RESOLVE($\mathcal{G}$) attempts to resolve ?'s in a connected component $\mathcal{G}$ of G. It looks at each locus in turn, identifies a resolved locus and uses this to resolve the locus at other vertices in the connected component by traversing in a manner consistent with the colours of the edges. We do not lose any feasible solution in this procedure because any feasible solution must satisfy SNP-consistency, Mendelian-consistency and Endgame-consistency, which are specified by the colours of the edges.

Define $v(i)$ to be the value of locus i ($= 0, 1$ or $?$) at vector v.

LOCUS_RESOLVE($\mathcal{G}$):

For each locus i:

1. Traverse the connected component $\mathcal{G}$ to find a vector v where $v(i)$ is resolved ($= 0$ or 1). If no such v exists, go to the next locus.

2. Traverse $\mathcal{G}$ using a linear-time graph traversal procedure (such as depth-first search), starting at v. For any edge $e = (v_1, v_2)$ traversed where $v_1(i) = x$ (0 or 1) and $v_2(i) = ?$,

   (i) If e is a red edge, set $v_2(i) = 1 - x$.

   (ii) Else set $v_2(i) = x$. □

Lemma 4 *LOCUS_RESOLVE($\mathcal{G}$) runs in $O(|\mathcal{G}|m)$ time, where $|\mathcal{G}|$ is the number of vectors in $\mathcal{G}$. All loci in $\mathcal{G}$ are either resolved or unresolved after running the procedure.*

Proof. LOCUS_RESOLVE performs m graph traversals, one for each locus, and each traversal takes $O(\mathcal{G})$ time. Hence the time complexity. At any locus i, if at least one vector is resolved at i, this will be identified in Step 1, and since $\mathcal{G}$ is connected, all other vectors will then be resolved at locus i in Step 2. □

Stage 2 will consider the nuclear families in the pedigree one by one and will try to connect the trios within the same nuclear family, in such a way as to respect Endgame-consistency. Specifically, green edges are added to connect two unconnected vectors in $\Gamma(\mathrm{N})$ that have the value 0 at heterozygous locus i of N, where N is the father or mother of the nuclear family. Green edges are like brown edges requiring that the ?s in the two vectors connected by the edge to be resolved the same.

There are two types of nuclear families: namely, the **Type A** families where there exists a locus that is heterozygous in either one of, but not both, the parents; and the **Type B** families where each locus is either heterozygous in both parents or homozygous in both parents. Stage 2 consists of Stages 2A and 2B, which process all Type A and Type B nuclear families, respectively.

For each Type A nuclear family, there exists a locus i that is heterozygous in either of, but not both of, F and M. Observe that locus i will be resolved in all vectors of the nuclear family comprised of father F, mother M and their children (that is how Stage 1B works). We can therefore connect all the vector-pairs into one single connected component by adding green edges between vectors in $\Gamma(\mathrm{N})$ with 0 at locus i, where N is the parent with heterozygous locus i. This we do in Stage 2A.

Stage 2A – Processing Type A families: For each nuclear family, which is comprised of, say, father F, mother M and their children, where there exists a locus i that is heterozygous in either (not both) of F and M, do the following:

1. Let $V = \{v \in \Gamma(\mathrm{N}) \mid v(i) = 0$ and v belongs to this nuclear family$\}$ where N is the parent (either M or F) such that i is heterozygous in N. Pick one vector from V and call it u.

2. For each vector $v \in V$, $v \neq u$, add a green edge to join u and v. After this, the vectors in the nuclear family are connected as a single connected component.

3. Run LOCUS_RESOLVE($\mathcal{G}$) where $\mathcal{G}$ is the connected component containing u.

4. Check for Endgame-consistency within the family, reporting *"no solution"* if it is not maintained. □

For each Type B nuclear family, we make use of the sets $het()$ and $hom()$ defined in Section 2. If there is a trio T where $hom(T)$ is empty, then $het(T)$ contains all loci (where F and M are also heterozygous). Since all distinct $het()$'s are disjoint, this implies any other trio T' in the nuclear family either has $het(T')$ empty, or $het(T') = het(T)$ (and $hom(T')$ is empty). In this case all trios with empty $hom()$ cannot be connected (in Stage 2B). So in Stage 2B we only consider trios where $hom(T)$ is non-empty.

We first consider each S_i as defined in the proof of Lemma 1, and connect all trios in the same S_i by adding green edges between them. All trios in the same S_i have identical $het()$ and hence identical (and non-empty) $hom()$, thus they share a resolved locus, which allows us to add a green edge correctly.

Then we will add edges connecting S_i and S_{i+1} for all i. If S_i and S_{i+1} share some common locus in their $hom()$'s, then this allows us to add a green edge correctly using the resolved loci. If they do not share any common locus in their $hom()$'s, then this implies that the union of their $het()$'s equals the set of all loci (where F and M are heterozygous). This means that they are the only two S_i's (call them S_1 and S_2) which have non-empty $het()$. If there are trios in S_0 (which has an empty $het()$), then S_0 shares some common resolved locus with both S_1 and S_2 and all S_i's can be connected by green edges. Otherwise if there are no trios in S_0, then S_1 and S_2 cannot be connected together.

Stage 2B – Processing Type B families: For each nuclear family, which is comprised of, say, father F, mother M and their children, where each locus is either homozygous in both F and M, or heterozygous in both F and M:

1. If there exists a locus i that is heterozygous in F (and also M), then do the following:
   (a) Let the sets of loci $het()$ and $hom()$ of each trio and the S_i's be as defined in the proof of Lemma 1.
   (b) For each S_i with corresponding trios $T_1, T_2, \ldots, T_k$:
      i. Pick a locus x in $hom(T_1)$. This is also in $hom(T_j)$ for all other j. If such x does not exist, go to the next S_i.
      ii. For each trio T_j other than T_1, add a green edge to join u and v, where $u, v \in \Gamma(\mathrm{F})$ are vectors for trios T_1 and T_j with value 0 at locus x.
   (c) For each pair of S_i and S_{i+1}:
      (i) Pick a T in S_i and a T' in S_{i+1}.
      (ii) If $hom(T)$ and $hom(T')$ share some common locus x, then add a green edge to join u and v, where $u, v \in \Gamma(\mathrm{F})$ are vectors for trios T and T' with value 0 at locus x.
      (iii) Otherwise (i.e., no such common locus exists), S_i and S_{i+1} are the only two such sets with a non-empty $het()$. Call them S_1 and S_2. If there are trios in S_0, then add a green edge between S_1 and S_0, and between S_2 and S_0, as in Step (c)(ii). If there are no trios in S_0, then no green edges are added.
   (d) Run LOCUS_RESOLVE() on each connected component of G of the nuclear family with the green edges added above.
   (e) Check for Endgame-consistency within the family, reporting *"no solution"* if it is not maintained.

2. Otherwise, no green edges are added for this family. □

Lemma 5 *The time complexity of Stage 2 (Stages 2A and 2B) is $O(mn)$. Furthermore, after Stage 2, all loci are either resolved or unresolved in each connected component of G, and G has $O(n)$ vertices and edges, and is acyclic.*

Proof. Stage 2A considers the nuclear families of the pedigree one by one. For each nuclear family, with, say, k children: locus i can be determined in $O(m)$ time; Step 1 takes $O(k)$ time with V containing one vector per trio of the family; Step 2 takes $O(k)$ time; Step 3 takes $O(km)$ time; and Step 4, which checks for Endgame-consistency, can be done in $O(km)$ time. Therefore, the total time complexity of Stage 2A is $O(mn)$.

Stage 2B processes the nuclear families similarly. Each addition of a green edge in Steps 1(b) and 1(c) takes $O(m)$ time, and thus for a nuclear family with k children, Steps 1(b) and 1(c) take $O(km)$ time. Steps 1(d) and 1(e) take $O(mn)$ time as in Stage 2A. Hence the time complexity of Stage 2B is also $O(mn)$.

The execution of LOCUS_RESOLVE after green edges are added ensures all loci are either resolved or unresolved in each connected component of G. No vertices are added to G and only up to $k-1$ green edges are added for each family with k children. Thus, G continues to have $O(n)$ vertices and edges. All green edges are added between vectors of the same individual node, and within a nuclear family, green edges are added in either the father or the mother but not both. Hence, in the absence of mating loops, G remains acyclic. □

Lemma 6 *If a connected component $\mathcal{G}$ of G has only resolved and unresolved loci (no mixed loci), then all possible ways of resolving ?'s in vectors in $\mathcal{G}$ such that SNP-consistency and Mendelian-consistency are maintained will either all make all vector-pairs Endgame-consistent or all make some vector-pairs Endgame-inconsistent.*

Proof. Consider a particular resolution of ?'s in the vectors in $\mathcal{G}$ such that SNP-consistency and Mendelian-consistency are maintained. Suppose Endgame-inconsistency occurs at node N, i.e., there exist two vector-pairs $\langle x_1, x_2\rangle$, $\langle y_1, y_2\rangle \in \Phi(\mathrm{N})$ such that the vector values at some heterozygous loci i and j ($i \neq j$) for u_1, u_2, v_1 and v_2 are a permutation of the four possibilities: 00, 01, 10 and 11. We can assume, without loss of generality, that the value at such i and j for x_1, x_2, y_1 and y_2 are 00, 11, 01 and 10, respectively. Consider the following three cases for the state of loci i and j prior to the resolution:

Case 1: Loci i and j were both unresolved in $\mathcal{G}$. Then, for all other possible resolutions, the values at loci i and j for x_1, x_2, y_1 and y_2 would either be 00, 11, 01 and 10 respectively, or 11, 00, 10 and 01 respectively, and Endgame-consistency would also be violated.

Case 2: Only one of locus i and j was unresolved, say i, in $\mathcal{G}$. Then, for all other possible resolutions, the values at loci i and j for x_1, x_2, y_1 and y_2 would either be 00, 11, 01 and 10 respectively, or 10, 01, 11 and 00 respectively, and Endgame-consistency would also be violated.

Case 3: Both locus i and j were resolved. Then, Endgame-inconsistency existed prior to any resolution of ?'s. □

The green edges are essential to the next stage of our algorithm by ensuring that certain trios are connected. This is established in Lemmas 7 and 8 below.

Lemma 7 *Consider a nuclear family within the pedigree with father F, mother M and their children. If there exists a locus i that is heterozygous in either one (not both) of F and M, then after Stage 2A, the family will be represented by a single connected component in G.*

Proof. This follows straightforwardly from how Stage 2A works. □

Lemma 8 *Consider a nuclear family within the pedigree comprised of father F, mother M and their children and with no family problems. If there exists a locus that is heterozygous in both F and M but homozygous in both C_1 and C_2, C_1 and C_2 being children of F and M, then the components for trios F-M-C_1 and F-M-C_2 will become connected during Stage 2B.*

Proof. All trios with a non-empty *hom*() will be connected to other trios within the same S_i in Step 1(b) of Stage 2B, which in turn will be connected to all other S_j's in Step 1(c) of Stage 2B. All such S_i's will be connected to a single connected component since any two of them must share at least one common locus in their *hom*() (since otherwise, all trios will form into two S_1 and S_2 with disjoint *het*() and *hom*(); see the discussion just before Stage 2B is defined). Thus the two trios, which share a common locus in their *hom*()'s, will be connected in Stage 2B. □

The previous two lemmas lead up to Lemma 9 below, which defines the Mother-Father Property. The Mother-Father Property helps us from not having to check for Endgame-consistency for the mother if the father is Endgame-consistent, or vice versa. This will become important in Stage 3.

Lemma 9 (Mother-Father Property) *Suppose (a) M and F are the mother and father of two unconnected trios in G after Stage 2 and (b) the given pedigree has no family problems. Then, for all possible way(s) of resolving ?s in vectors in the two trios such that SNP-consistency and Mendelian-consistency are maintained, M and F are either both Endgame-consistent or both Endgame-inconsistent.*

Proof. Suppose F is Endgame-inconsistent. Without loss of generality, let the values at loci i and j for x_1, x_2, y_1 and y_2 be 00, 11, 01 and 10, respectively where $\langle x_1, x_2\rangle$, $\langle y_1, y_2\rangle \in \Phi(\mathrm{F})$. This means that loci i and j are heterozygous loci for F. Since the two trios are not connected by a green edge, loci i and j are also heterozygous for M (Lemma 7). Let C_1 and C_2 be the two respective children of F connected to $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$ by brown edges. In the absence of family problems and green edges connecting the two trios, there are only three cases to consider (there is no need to consider i or j being homozygous for both C_1 and C_2 according to Lemma 8): (i) when loci i and j are both heterozygous for both C_1 and C_2; (ii) when loci i and j are both heterozygous for C_1 and both homozygous for C_2; and (iii) when locus i is heterozygous for C_1 and homozygous for C_2 while locus j is homozygous for C_1 and heterozygous for C_2. It can be readily shown that in all three cases, M is also Endgame-inconsistent.

The argument is similar supposing M is Endgame-inconsistent. Thus, if F (or M) is Endgame-inconsistent, then M (respectively, F) is Endgame-inconsistent, and the contrapositive implies that, if M (or F) is Endgame-consistent, then F (respectively, M) is Endgame-consistent. The lemma follows. □

3.3 Stage 3 – Adding White Edges

After Stage 2, suppose G is left with more than one connected component. The idea of Stage 3 is to connect components of G together with white edges, so that a single connected component results and loci can be further resolved. Before we present the various sub-stages of Stage 3 formally, we first give an intuitive idea.

Suppose a pedigree has a CHC solution. Then for any node N with vector-pairs (u_1, u_2) and (v_1, v_2), if these vector-pairs are not already connected by green edges in Stage 2, then we need to add a white edge to connect either u_1 to v_1, or connect u_1 to v_2. These represent the two different ways of resolving the haplotypes in N so as to maintain Endgame-consistency (i.e., either u_1 should be identical to v_1, or it should be identical to v_2). Thus white edges are analogous to green edges and they are treated as "non-red" edges by LOCUS_RESOLVE.

While it may appear at first sight that each of these white edges can be added arbitrarily, it turns out that this is not true when multiple white edges are considered together, and we need a way to determine which of the two ways is the correct way of connecting the vectors. To do this, we first construct a **support graph *H*** in Stage 3A. The support graph contains unlabeled edges, each corresponding to a white edge in G, and which will be labeled with either 0 or 1 in Stage 3C. Suppose e is an unlabeled edge in H corresponding to the white edge between the vector-pairs (u_1, u_2) and (v_1, v_2) as defined in the previous paragraph. A label of 0 on e denotes that the white edge in G should connect u_1 and v_1, while a label of 1 denotes that the white edge should connect u_1 to v_2. This is how H is used. In order to find this labeling, we will construct another graph J in Stage 3B which captures the constraints on how the unlabeled edges can be labeled.

We start with the construction of H in Stage 3A.

Stage 3A – Constructing the support graph *H*:

1. For each nuclear family:

   If the vector-pairs in the family consist of $k > 1$ connected components[1], then do the following. Pick a vector from each of the connected components in either the father or the mother, but not both (all vectors must be from the same parent). Create a vertex in H for each such vector. Add $k - 1$ *unlabeled edges* to join these vertices in H.

2. For each pedigree node N:

   Suppose this individual N belongs to k' different nuclear families. Within each such nuclear family, any two vectors of N in G are either already connected in Stage 2, or they belong to connected components with corresponding vertices in H that are connected in Step 1 above. Vectors of N from different nuclear families are however not connected in either G or H. Pick one vector from N from each of these nuclear families. Create a vertex in H for each such vector. Add $k' - 1$ new *unlabeled edges* to connect them in H.

3. For each connected component $\mathcal{G}$ in G:

   (a) Let k'' be the number of vectors in $\mathcal{G}$ that are chosen as vertices in H in Steps 1 or 2 above.

   (b) Join these vertices in H with $k'' - 1$ *labeled edges*. The label is 0 if the path between the two vectors in G has an even number of red edges, and 1 otherwise. □

Figures 4(a) and 4(b) show a pedigree and the corresponding local graph G. Figure 4(c) shows the support graph H. In this example, since each nuclear family has only one connected component,

[1]After Stage 2, Type A families have $k = 1$, and vector-pairs of each Type B family are connected into at most two connected components except those trios whose $hom()$ is empty, which are still unconnected.

Step 1 is skipped. Unlabeled edges (A,B), (B',C) and (C',D) are added in Step 2, and labeled edges (B,B') and (C,C') with label 0 are added in Step 3. In this case H happens to be a path, but it can be more general. Lemma 10 shows that H is always acyclic.

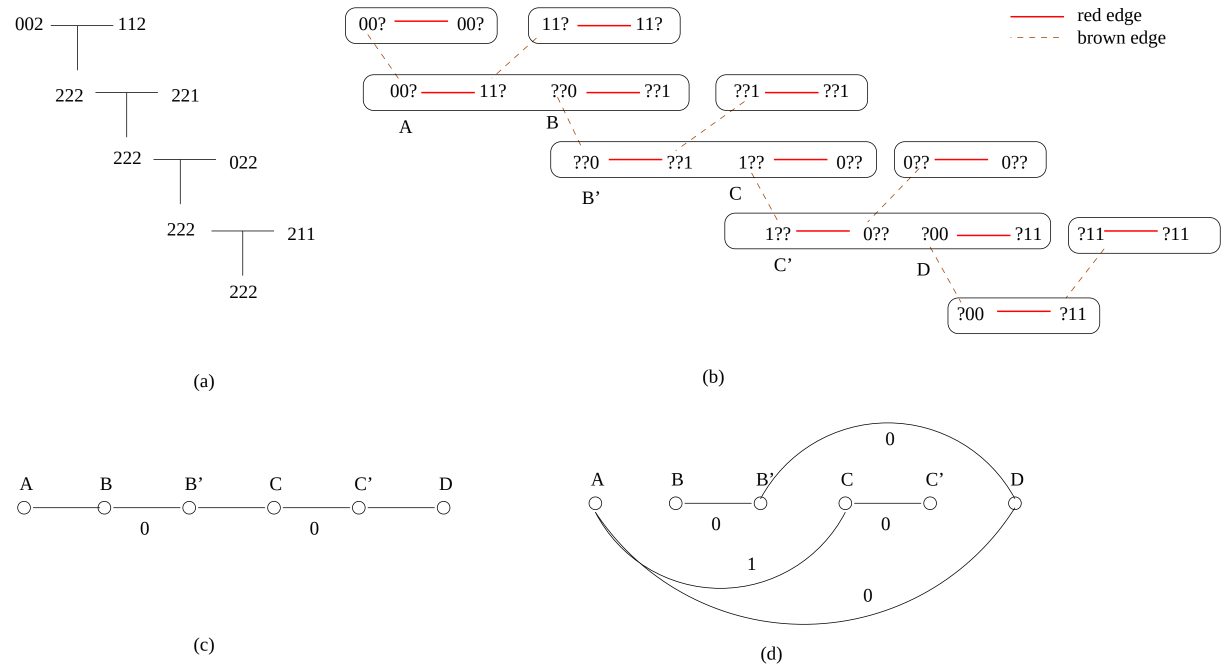

Figure 4: An example showing the steps in Stage 3. (a) The pedigree. (b) The local graph G. (c) The support graph H. Two edges are labeled with 0. (d) The parity constraint graph J. Three constraints are added.

Lemma 10 *If there are no mating loops in the pedigree, then H is acyclic.*

Proof. If there is a cycle in H, it cannot involve only vectors in one nuclear family, by our construction (Step 1 of Stage 3A). Any other cycle is impossible without mating loops. □

Lemma 11 *H is connected, has $O(n)$ vertices and edges, and can be constructed in $O(n)$ time.*

Proof. Vertices in H which correspond to vectors in the same connected component in G are connected by labeled edges. The different connected components of G can always be connected by adding edges joining vectors in the same $\Phi(\mathrm{N})$ for some node N (Lemma 3); these correspond to the unlabeled edges in H. Hence H is connected.

H clearly has $O(n)$ vertices since the set of vertices is a subset of those of G. Since H has no cycles by Lemma 10, it has $O(n)$ edges.

Steps 1 and 2 of Stage 3A take $O(n)$ time since the time to process each nuclear family or individual is proportional to the number of vectors in them. We can check for connectivity easily by preprocessing (e.g., traversing G to identify connected components). Step 3 also takes $O(n)$ time, where the only tricky part is computing the labels on the labeled edges. This can be done in constant time per label, once the following preprocessing step is done. By traversing each connected component $\mathcal{G}$ of G, we can compute, for each vertex v in $\mathcal{G}$, whether the number of red edges in the path from a fixed vertex t in $\mathcal{G}$ is odd or even, i.e., the parity. Since G has $O(n)$ edges, this can be done in $O(n)$ time and is only done once as a preprocessing step. Then, the parity of the path in G between any pair of vertices u and v in the same connected component can be computed in constant time from the parity of the path between u and t and that between t and v. □

In Stage 3B, we construct a **parity constraint graph J** to represent the constraints on the labeling of H. One of the essential differences between H and J is that H shows connections between "neighboring" components while J captures all parity constraints between far-apart components.

Figure 4(d) shows the graph J which is derived from the graph H in Figure 4(c). The vertices in J are the same as the vertices in H. Since the vertices in H correspond to vertices (vectors) in G, we can extend the terminology of vectors to the vertices in H: for example, we say that a vertex u in H is heterozygous at locus i when i is heterozygous in a pedigree node N where $u \in \Gamma(\text{N})$. Simlarly we can speak of a vertex as homozygous, resolved, unresolved, etc. at a locus i.

Assume white edges have been added to G. G remains acyclic by a reasoning similarly to showing that G is acyclic after adding green edges (Lemma 5). Hence, a path between any two vectors u and v in G is unique. If u and v are heterozygous and resolved (have 0 or 1) at locus i but all other vectors (if any) in the path between u and v are unresolved at locus i, then there is a constraint on how the unresolved loci can be resolved (equivalently, how the white edges should be added): namely, the number of red edges in the path in G must be even (or odd) if u and v have the same (respectively, different) parity of resolved loci at a locus i. This is because the unresolved loci at the two ends of each red edge must have different parity. To represent this constraint, we add an edge (u, v) labeled L between u and v in J, where L is 1 if u and v are resolved differently at locus i, and 0 otherwise.

A straightforward implementation of the above idea will lead to too many edges. Stage 3B below only adds $O(mn)$ edges to J, and Lemma 12 shows that this is sufficient to represent all parity constraints.

Stage 3B – Constructing the parity constraint graph J:

1. The vertices in J are the same as the vertices in H.

2. Add an edge between two vectors u and v in J if (u, v) is labeled in H. Furthermore, the label of this edge in J is the same as its label in H.

3. For each locus i, consider the tree H as if it is separated into subtrees at all vertices where i is homozygous. That is, each subtree does not contain any vertex homozgyous at i. For each subtree, we root the subtree at an arbitrary vertex that is heterozygous and resolved at i. (If there is no such vertex, go to the next subtree.) Traverse the subtree and find, for each vertex v that is heterozygous and resolved at locus i, its lowest ancestor u that is also heterozygous and resolved at locus i. Add an edge between u and v in J. Label this edge with 0 (or 1) if u and v have same (respectively, different) parity of locus value. Note that there may already be such an edge in J, with the same or different labels, due to other loci. If it is the same label, do not add the edge (which is redundant). If the label is different, report *"no solution."*

4. Check whether all cycles in J have an even number of edges labeled 1. Report *"no solution"* and stop if there is a cycle in J with an odd number of edges labeled 1.

5. Note that J may not be connected. To make J connected, add edge (u,v) to J if u and v are in different connected components in J and (u,v) is an edge in H. This is always possible

because H is a connected graph, and J and H have the same set of vectors as the vertices. Arbitrarily label this edge with 0. We call the corresponding edge in H a **free edge** because we have the freedom to label (u, v) with 1 instead. We continue adding edges until J is connected. □

In effect, the graph J represents a set of linear equations modulo 2; in the example in Figure 4(d) the equations are $x_{AB} + x_{B'C} = 1, x_{B'C} + x_{C'D} = 0$, and $x_{AB} + x_{B'C} + x_{C'D} = 0$. The free edges in Step 5 correspond to the edges in H that are still unlabeled after Step 4, and are the result of the degrees of freedom in the system of equations. These unlabeled edges are 'free' by themselves, in the sense that they can be assigned either 0 or 1, but once an assignment is made on one of the free edges, the other free edges may become non-free. For example, consider Figure 4(d). There are no free edges since H is one connected component (and the system of linear equations has no degree of freedom). If we assume the edge connecting A and D does not exist, then there are two connected components, and AB, $B'C$ and $C'D$ are all potential free edges. But there is only one degree of freedom since if we assign AB to, say, 0 then all other 'free' edges have their values fixed.

Lemma 12 below shows that Step 3 in Stage 3B adds sufficient edges in J to represent the parity constraints.

Lemma 12 *Suppose there is a path between two vertices u and v in H and there is a locus i such that both u and v are heterozygous and resolved at i while all other vertices in this path are not resolved at i. Let $L = 0$ if u and v have the same resolved value at i, and 1 otherwise. Then, there is a path in J connecting u and v so that the XOR's of all labels in this path equals L.*

Proof. Since all other vertices in this path are not resolved, u and v must be in the same subtree in Step 3 of Stage 3B. If one of u or v is an ancestor of the other in the rooted subtree (for locus i), not necessarily the lowest ancestor, then by the construction in Step 3 of Stage 3B, there is a sequence of edges $(u, v_0), \dots, (v_k, v)$ in J connecting u and v, such that all these vertices are resolved at locus i. If this path is a single edge (in the case of the lowest ancestor), then we are done. Otherwise, we can assume by induction that the XOR of the labels on the path between u and v_k is equal to the parity difference of u and v_k. Adding the edge from v_k to v, with the label equal to the parity difference of v_k and v, the claim follows.

Otherwise, if neither u nor v is an ancestor of the other in the subtree, let w be the lowest common ancestor of u and v which is resolved at i. We then have two paths, one from w to u and the other from w to v, which by a similar argument to the above, have the correct labels. Hence, there is a path from u to v (through w) in J, and the XOR of the labels on this path is equal to the parity difference of u and w XORed with the parity difference of w and v, the result of which is the parity difference of u and v. □

Lemma 13 *If Steps 3 and 4 of Stage 3B report "no solution," then there is no CHC solution. Otherwise, J has no odd cycle (an odd cycle is a cycle where the number of 1-labeled edges is odd).*

Proof. In Step 3, if "no solution" is reported, there are two conflicting constraints. In Step 4, if "no solution" is reported, then there is an odd cycle. Each edge with label 1 denotes that the resolved loci at two ends of the edge must be of different parity (and label 0 implies same parity). It follows that there is no way of resolving the loci consistently along an odd cycle. □

The free edges introduced in Step 5 are to make J connected so that we can determine the parity difference between any two nodes in J and completely label all edges in H. The following lemma shows that these free edges can be added freely and labeled arbitrarily without affecting the existence of the CHC solution.

Lemma 14 *If there is a CHC solution of the pedigree, then any label (0 or 1) on the free edges introduced in Step 5 of Stage 3B will make all vector-pairs SNP-consistent, Mendelian-consistent and Endgame-consistent. On the other hand, if there is no CHC solution, none of the labels on the free edges can achieve Endgame-consistency for all vector-pairs.*

Proof. It is obvious that SNP-consistency and Mendelian consistency will always be maintained because of the red and brown edges. Since a free edge (u, v) is only added when J is not connected, u and v must be two vectors in different connected components of J and there must not exist two vectors, one in each connected component, where locus i is resolved and the vectors in the path in H between these two vectors are unresolved at locus i.

As edge (u,v) is in H, u and v must be in the same $\Phi(\mathrm{N})$ for some node N whose loci are either homozygous or heterozygous. If N is heterozygous at locus i, then locus i will be unresolved in all vectors in either u's connected component, or v's connected component, or both connected components (otherwise J cannot be disconnected). In all these cases, it can be shown that, from Lemma 6, Endgame-consistency will be maintained or not maintained no matter whether the free edge is labeled with 0 or 1. Thus the lemma is proved. □

Lemma 15 *J has $O(mn)$ edges and can be constructed in $O(mn)$ time.*

Proof. There are $O(n)$ vertices and edges in H. Thus, Steps 1 and 2 of Stage 3B can be done in $O(n)$ time.

Step 3 can be done in $O(mn)$ time using a recursive traversal of each subtree of H for each locus as follows. We start at the root noting itself as the lowest resolved ancestor, and recursively traverse each child, passing down the ancestor information in the recursive calls. At each child, its lowest resolved ancestor is the lowest resolved ancestor of the parent. If the child itself is resolved heterozygous, then the child notes itself as the lowest resolved ancestor in subsequent traversal of its own children. Thus it takes $O(n)$ time to perform such traversal for each locus.

Each traversal of a locus adds at most $O(n)$ edges to J, so J has at most $O(mn)$ edges.

In Step 4 we need to identify cycles with an odd number of edges labeled 1. If we imagine contracting every edge in J with label 0, then the problem reduces to checking whether the contracted graph has an odd cycle, which amounts to checking bipartiteness. Thus Step 4 can be done in $O(mn)$ time.

Step 5 can also be done in $O(mn)$ time as follows. First, for each vertex x in J, keep a list $\mathrm{LIST}(x)$ of vertices adjacent to x in H. Next, we perform a two-pass traversal as follows. Start with an arbitrary vertex X, traverse the connected component and label the vertices traversed as "marked." Then we go back and traverse the connected component again starting at X. When traversing vertex x, we check whether all the vertices in $\mathrm{LIST}(x)$ are marked. If there is a vertex y that is unmarked, we add (x, y) to J and perform the same two-pass traversal for the connected component of y; in effect we are doing a traversal within traversal. In this way, we grow from one connected component of J and add free edges to connect to other connected components. The two-pass approach is employed to prevent adding edges which are not actually free after other free edges are added. Since the graph has $O(mn)$ edges, Step 5 takes $O(mn)$ time.

Thus, the total time complexity of Stage 3B is $O(mn)$. □

Next, we use J to complete the labeling of H.

Stage 3C – Completing the labeling of H:

1. Traverse J, computing, for each vertex v in J, the parity (odd or even) of the number of 1-labeled edges in the path from a fixed vertex t in J.

2. For each unlabeled edge (u, v) in H, if u and v have same parity in J, then label edge (u, v) in H with 0, otherwise with 1. □

Lemma 16 *All edges in H can be labeled with 0 or 1 in $O(mn)$ time in Stage 3C, and the labels in H are consistent with the parity constraints specified in J in the sense that the parity between any two vectors u and v specified in J is the same as the parity of the number of 1-labeled edges in the path between u and v in H.*

Proof. Note that J specifies a unique parity between any two vertices because J has no odd cycle. Consider a path $P = (v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ in H. Each unlabeled edge (v_i, v_{i+1}) in P receives a label equal to the parity between v_i and v_{i+1} in J in Step 1 of Stage 3C, which is equal to the parity of the number of 1-labeled edges between v_i and v_{i+1} in J. Each labeled edge (v_i, v_{i+1}) in P has the same edge (with the same label) in J, and hence the label of this edge in H is also equal to the parity of the number of 1-labeled edges between v_i and v_{i+1} in J. Hence, the parity of the number of 1-labeled edges in P is equal to the XOR of the labels on all edges in P, which in turn is equal to the XOR of the parity between v_i and v_{i+1} in J over all i, which is the parity between v_0 and v_k in J.

As far as the time complexity is concerned, since H has $O(n)$ edges and J has $O(mn)$ edges, the total time complexity of Stage 3C is $O(mn)$. □

Stage 3D – Adding white edges to G:

1. For each edge (u, v) in H that became labeled during Stage 3C:
   (a) If the edge is labeled 1, then let x be the vector adjacent to v by a red edge; otherwise, let x be v.
   (b) Add a white edge between u and x.
2. G now becomes a single connected component. Run LOCUS_RESOLVE(G). □

Lemma 17 *Stage 3D can be done in $O(mn)$ time, and after Stage 3D, G will be a single connected component with only unresolved and resolved loci.*

Proof. Step 1 of Stage 3D considers each of the $O(n)$ edges of H one by one, each taking constant time; thus this step takes $O(n)$ time. Step 2 takes $O(mn)$ time. The time complexity thus follows.

H is a connected graph that contains at least one vertex from each connected component of G, and the newly labeled edges in H are between vertices that were not connected in G (while the edges that were already labeled are between vertices that were already connected in G). Therefore, each white edge added results in one fewer connected component in G, and G will become a single connected component after Step 1 finishes.

LOCUS_RESOLVE ensures that G, as a single connected component, has only unresolved and resolved loci. □

Figure 5 shows the graph H of the same example in Figure 4 after the edges are labeled. The resulting white edges are added to G and the loci resolved.

Lemma 18 *If the pedigree has a CHC solution, Stage 3D maintains Endgame-consistency.*

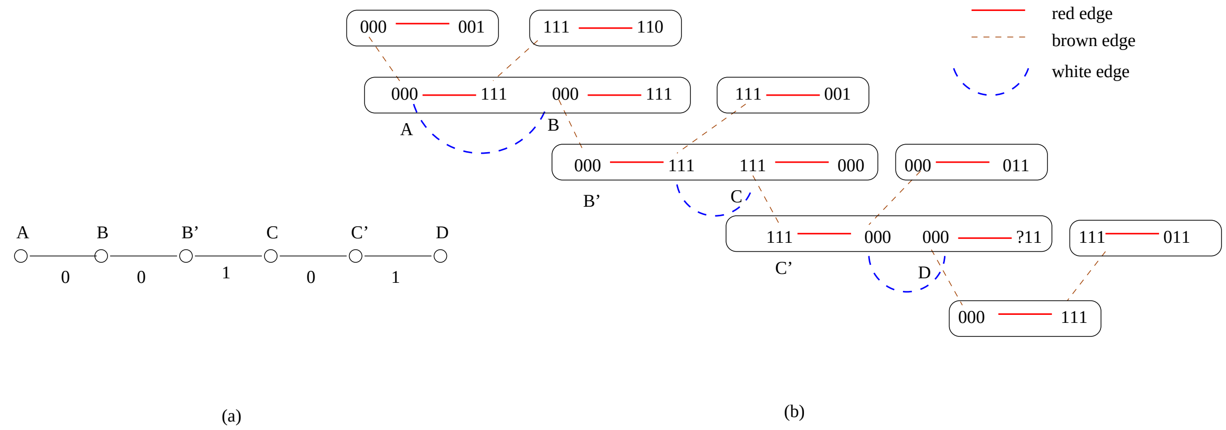

Figure 5: The same example from Figure 4 showing the result of Stages 3C and 3D. (a) The graph H with the correct labels. In this case there is a unique solution. (b) The local graph G after addition of white (dashed) edges. Loci values are also resolved.

Proof. Suppose, to the contrary, that some node N becomes Endgame-inconsistent after Stage 3D. Without loss of generality, let the values at loci i and j for x_1, x_2, y_1 and y_2 be 00, 11, 01 and 10, respectively, where $\langle x_1, x_2\rangle, \langle y_1, y_2\rangle \in \Phi(\mathrm{N})$.

Consider the situation prior to Stage 3D. Since the pedigree has a CHC solution, given Lemma 6, the vector-pairs in each connected component are Endgame-consistent. Thus $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$ must belong to different connected components; call them G_1 and G_2, respectively. Now suppose $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$ become connected during Stage 3D after the addition of a white edge e, which connects G_1 and G_2. There are four cases to consider:

Case 1: e connects $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$. White edge e corresponds to an edge in H, and since H is acyclic, it is the unique edge between the vector-pairs and is labeled with a unique parity. Without loss of generality, suppose e connects x_1 and y_1 and is labeled 0. This white edge will make x_1 and y_1 equal and therefore the value of loci i and j cannot possibly become 00 for x_1 and 01 for y_1.

Case 2: e connects $\langle x_3, x_4\rangle$ in G_1 and $\langle y_3, y_4\rangle$ in G_2 where $\langle x_3,\ x_4\rangle$ and $\langle y_3,\ y_4\rangle \in \Phi(\mathrm{N})$. Since the pedigree has a CHC solution, and G_1 has only resolved and unresolved loci, according to Lemma 6, vector-pairs of N that are in G_1 must be Endgame-consistent. This implies that $\langle x_1,\ x_2\rangle$ and $\langle x_3, x_4\rangle$, which are in G_1, are Endgame-consistent. Likewise, $\langle y_1, y_2\rangle$ and $\langle y_3, y_4\rangle$ must also be Endgame-consistent. By the argument in Case 1, $\langle x_3, x_4\rangle$ and $\langle y_3, y_4\rangle$ must also be Endgame-consistent. This makes it impossible for $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$ to be Endgame-inconsistent.

Case 3: e connects $\langle x_3, x_4\rangle$ in G_1 and $\langle y_3, y_4\rangle$ in G_2 where $\langle x_3, x_4\rangle$ and $\langle y_3, y_4\rangle \in \Phi(\mathrm{M})$ and M is N's spouse. Suppose $\langle u_1, u_2\rangle \in \Phi(\mathrm{M})$ belongs to the same trio as $\langle x_1, x_2\rangle$ and suppose $\langle v_1, v_2\rangle \in \Phi(\mathrm{M})$ belongs to the same trio as $\langle y_1, y_2\rangle$. According to Lemma 9, $\langle u_1, u_2\rangle$ and $\langle v_1, v_2\rangle$ are also Endgame-inconsistent. Thus, we can consider $\langle u_1, u_2\rangle$ and $\langle v_1, v_2\rangle$ instead of $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$, and accordingly, apply the arguments of Case 2.

Case 4: e connects $\langle x_3, x_4\rangle$ in G_1 and $\langle y_3, y_4\rangle$ in G_2 where $\langle x_3, x_4\rangle$ and $\langle y_3, y_4\rangle \in \Phi(\mathrm{M})$ and M is neither N nor N's spouse. Assuming no mating loops, this case does not exist. □

3.4 Stage 4 – Finishing Up

At this point, our graph G has only one connected component, and it only has resolved or unresolved (no mixed) loci, since this is the property we maintain by our locus resolve procedures. If all loci are resolved then of course we are done. For those loci that are still unresolved, Lemma 6 tells us that any way of resolving makes no difference: we can arbitrarily resolve them in an SNP-consistent and Mendelian-consistent manner, and it will not affect Endgame-consistency in the sense that either all vector-pairs will be Endgame-consistent for all resolutions, or there will be Endgame-inconsistent vector-pairs for all resolutions. This means the algorithm does not need to try all possibilities; any one will do. This is crucial for avoiding an exponential blow-up.

Hence, we do the following as the final stage of our algorithm.

Stage 4 – Dealing with a single connected component:

1. Arbitrarily pick a vector u of G. For all unresolved loci i, assign a value of 0 to each of them. Then run LOCUS_RESOLVE(G).

2. For all N, check Φ(N) for Endgame-consistency and report *"no solution"* if it is not maintained.

Lemma 19 *Stage 4 runs in $O(mn)$ time.*

Proof. After Stage 3D, G is a single connected component and has no mixed loci. By Lemma 6, we do not have to try all possible resolutions; any one will do. Let s be the number of unresolved loci in G. The time complexity of resolving all of these unresolved loci is $O(sn)$ since LOCUS_RESOLVE runs in $O(n)$ time. Checking all N for Endgame-consistency can be done in $O(mn)$ time. □

Note that assigning a value of 1 instead of 0 to any locus before running LOCUS_RESOLVE would work equally well (Lemma 6); the effect is that all 1's become 0 and all 0's become 1 at each such locus, giving another solution. In general, if there are s unresolved loci after Stage 3D and the pedigree admits a consistent solution, then there are 2^s different CHC solutions. However if every node in the pedigree has exactly s heterozygous loci then there are only 2^{s-1} different CHC solutions due to symmetry.

Theorem 1 *For a given pedigree, we can either achieve a solution that represents a CHC for the given pedigree, or report "no solution" when there is no solution, in $O(mn)$ time where n is the number of nodes in the pedigree and m is the number of loci.* □

Proof. Each of the stages of our algorithm runs in $O(mn)$ time. The algorithm only reports "no solution" when there can be no CHC solution. If the algorithm does not report "no solution" then after Stage 4, all loci are resolved and SNP-consistency, Mendelian-consistency and Endgame-consistency are all maintained. Thus the resolved loci values represent a CHC solution. □

4 Open Problems

In this paper, an optimal linear-time algorithm is presented to solve the haplotype problem for pedigree data when there are no recombinations and the pedigree has no mating loops. It remains an open problem to extend the algorithm to handle mating loops. For the haplotyping problem with recombinations, the problem becomes intractable even when at most one recombination is allowed at each haplotype of a child, or when the problem is to find a feasible haplotype with the minimum number of recombinations (even without mating loops) [3]. However, there is still much scope for further study. For example, in practice, pedigree data often contains a significant amount of missing alleles (up to 14-15% of the alleles belonging to a block could be missing in the pedigree data studied). In some cases, the deduction of the missing information on alleles is possible. The goal is then to devise an efficient algorithm to determine as many missing alleles as possible.

Acknowledgements

We thank Tao Jiang and the referees for their valuable comments.

References

[1] R. Cox, N. Bouzekri, S. Martin, L. Southam, A. Hugill, M. Golamaully, R. Cooper, A. Adeyemo, F. Soubrier, R. Ward, G. M. Lathrop, F. Matsuda and M. Farrall. Angiotensin-1-converting enzyme (ACE) plasma concentration is influenced by multiple *ACE*-linked quantitative trait nucleotides. *Human Molecular Genetics*, 11:2969-2977, 2002.

[2] J. Li and T. Jiang. Efficient inference of haplotypes from genotypes on a pedigree. *Journal of Bioinformatics and Computational Biology*, 1(1):41-69, 2003.

[3] J. Li and T. Jiang. An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming. *Proceedings of 8th International Conference on Computational Molecular Biology*, pages 20-29, 2004.

[4] J. R. O'Connell. Zero-recombinant haplotyping: applications to fine mapping using SNPs. *Genetic Epidemiology*, 19 Suppl 1:S64-70, 2000.

[5] E. Russo and P. Smaglik. Single nucleotide polymorphism: Big pharmacy hedges its bets. *The Scientist*, 13, 1999.

[6] N. Wang, J. M. Akey, K. Zhang, K. Chakraborty, and L. Jin. Distribution of recombination crossovers and the origin of haplotype blocks: The interplay of population history, recombination, and mutation. *The American Journal of Human Genetics*, 11:1227-1234, 2002.

[7] E. M. Wijsman. A deductive method of haplotype analysis in pedigrees. *The American Journal of Human Genetics*, 41(3):356-373, 1987.

[8] J. Xiao, L. Liu, L. Xia and T. Jiang. Fast elimination of redundant linear equations and reconstruction of recombination-free Mendelian inheritance on a pedigree. *Proceedings of 18th ACM-SIAM Symposium on Discrete Algorithms*, pages 655-664, 2007.