Inferring Haplotypes from Genotypes on a Pedigree with Mutations, Genotyping Errors and Missing Alleles

Wei-Bung Wang* and Tao Jiang
Department of Computer Science, University of California - Riverside, Riverside, CA 92521, USA
Email: {weiw,jiang}@cs.ucr.edu

Inferring the haplotypes of the members of a pedigree from their genotypes has been extensively studied. However, most studies do not consider genotyping errors and *de novo* mutations. In this paper, we study how to infer haplotypes from genotype data which may contain genotyping errors, *de novo* mutations and missing alleles. We assume that there are no recombinants in the genotype data, which is usually true for tightly linked markers. We introduce a combinatorial optimization problem, called *haplotype configuration with mutations and errors* (HCME), which calls for haplotype configurations consistent with the given genotypes that incur no recombinants and require the minimum number of mutations and errors. HCME is NP-hard. To solve the problem, we propose a heuristic algorithm, the core of which is an *integer linear program* (ILP) using the system of linear equations over Galois field $GF(2)$. Our algorithm can detect and locate genotyping errors that cannot be detected by simply checking the Mendelian law of inheritance. The algorithm also offers error correction in genotypes/haplotypes rather than just detecting inconsistencies and deleting the involved loci. Our experimental results show that the algorithm can infer haplotypes with a very high accuracy, and recover 65%–94% of genotyping errors depending on the pedigree topology.

Keywords: algorithm, haplotype inference, pedigree, genotyping error, mutation, integer linear programming.

1. INTRODUCTION

In October 2002, the international HapMap project was launched[1]. One of the main objectives of HapMap project is to identify haplotype structures of humans and common haplotypes among different populations. A *haplotype* is a combination of alleles at multiple genetic marker (*e.g.*, SNP) loci on the same chromosome. In *diploid* organisms like human, chromosomes (other than the sex chromosomes) come in pairs. Each genetic marker on a pair of chromosomes occurs at the same location of both chromosomes. However, the marker may have different alleles on the two chromosomes. The set of its two alleles is called the *genotype* of the marker and the assignment of the two alleles to the paternal and maternal chromosomes is called the *phase* of the marker. Inferring haplotypes from genotypes over a set of marker loci is called *haplotype inference*, which is also referred to as *phasing*. The haplotype information of SNP markers is of tremendous value to gene mapping and other genetic analyses (such as linkage analysis) because it gives a more accurate description of the inheritance process than the genotype information. Because of cost considerations, genotype data instead of haplotype data are routinely collected in practice, especially in large-scale sequencing projects. Therefore, efficient and accurate computational methods for haplotype inference have been extensively studied in the literature. See Ref. 2 for a review of these methods as well as the basic concepts involved in haplotype inference.

Haplotype inference methods can be divided into three groups according to the type of given genotype data: methods for *population* data collected from unrelated individuals[3–5], methods for *pedigree* data collected from individuals (typically from an extended family) that are related by the parent-child relationship[6–14], and methods for *pooled samples*[15, 16]. Here, we are interested in only pedigree data.

Some real pedigree data may actually contain mutations. In particular, a *de novo mutation* is a mutation that is present for the first time in a family member as a result of a mutation in a germ cell (egg or sperm) of one of the parents or in the fertilized egg itself. It has been observed that the detection and analysis of mutations in a pedigree could pro-

*Corresponding author.

vide a good alternative for some genetic variation research[17–19].

Most genotype data contain genotyping errors. Genotyping errors have a severe impact on subsequent analyses, such as linkage analysis[20–22]. Even slight amounts of genotyping error may significantly decrease haplotype frequency and haplotype reconstruction accuracy[23]. Moskvina *et al.* showed that even with low genotyping error rates (< 0.01), systematic differences in the error rate between samples may result in type I error (*i.e.*, false positive) rates substantially above 0.05 in case-control association studies[24].

Genotyping errors and *de novo* mutations may cause violation of the Mendelian law of inheritance, and hence pedigree data with errors and mutations cannot be properly handled by the above existing haplotype inference methods. When these methods are faced with data containing errors and mutations, they typically delete the loci that appear to be inconsistent. Very few haplotype inference methods in the literature deal with pedigree data that contain mutations and errors (one such method is a genetic algorithm in Ref. 25). Moreover, detecting genotyping errors is challenging, since these errors do not necessarily violate the Mendelian law of inheritance within nuclear families.

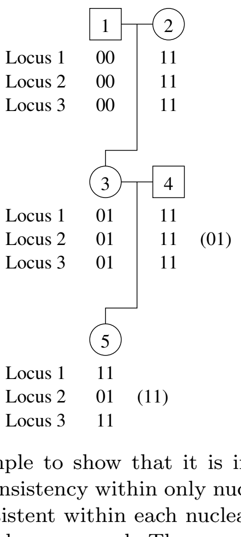


Fig. 1. An example to show that it is insufficient to consider Mendelian consistency within only nuclear families. The genotypes are consistent within each nuclear family although a genotyping error has occurred. The genotypes in parentheses indicate two possible ways of correcting the error. In a conventional pedigree, a square indicates a male, and a circle indicates a female.

In Fig. 1, all genotype data follow the Mendelian law of inheritance, but it requires two recombinants to explain locus 2 of individual 5. A better explanation is that there is a genotyping error on locus 2 of individual 4 or 5, and alternative genotypes are shown in the parentheses. There have been some work on detecting genotyping errors in the literature[21, 26–28]. Douglas *et al.* can detect 13%–77% of errors[26], and Zou *et al.* can detect $\leq 81\%$ of errors without assuming equal allele frequencies[28]. However, none of these work considers haplotype inference simultaneously.

In this paper, we study haplotype inference on pedigree data consisting of tightly linked markers that have no recombinants but may contain some genotyping errors, missing alleles and a very small number of *de novo* mutations (or simply, mutations). Since error and mutation events are rare, we formulate the problem as a combinatorial optimization problem, called the *haplotype configuration with mutations and errors* (HCME) problem, where we look for a haplotype solution consistent with the given genotype data that incurs no recombinants and requires the minimum number of mutations and errors. (Actually, we minimize a weighted summation of the numbers of mutations and errors.) Our hypothesis is that the configuration with the minimum number of mutations and errors is likely the true solution. This extends the well studied *zero-recombinant haplotype configuration* (ZRHC) problem where we try to find a consistent haplotype solution incurring no recombinants, errors or mutations. Although ZRHC is polynomial-time solvable[8], we can prove that HCME is NP-hard by a reduction from NAE-3SAT. We construct an *integer linear program* (ILP) for HCME using the system of linear equations over Galois field $GF(2)$ that has been developed in Ref. 8, 12 and 29 for solving ZRHC in almost linear time. We use the incremental approach introduced in a previous work[13] to reduce the number of constraints in the ILP instance.

We have implemented the algorithm and tested it on both simulated data and real data. The experimental results demonstrate that our algorithm can infer haplotypes with a very high accuracy. It can also recover most of the errors and impute most of the missing alleles correctly. However, most mutations would be explained by genotyping errors because we give errors a smaller weight (since they are more frequent than mutations) in the objective function of the above ILP.

The rest of the paper is organized as follows. In Section 2, we introduce the linear system, and formally define HCME as an optimization problem. In Section 3, we explain each part of our algorithm in a subsection. We explain how to construct constraints for the given genotype data in Section 3.1. We briefly describe how to tackle the problem by using ILP as a black box, which is similar to our previous work, in Section 3.2. We explain how to recover the optimal haplotype configuration in Section 3.3. Some detailed implementation issues are discussed in Section 3.4. In Section 4, we show our experimental results on both simulated data and real data, and discuss how each parameter may affect the accuracy and efficiency of our algorithms. Section 5 concludes the paper with a few remarks.

A pedigree can be drawn in three different forms: conventional form, formal form and graph form (Fig. 2). A pedigree is a tree if its formal form is a tree. In this paper, we will follow Ref. 13 to consider only tree pedigrees, which are very common among human pedigrees. From now on, when we talk about paths and cycles in a pedigree, we will consider it as a graph form. Note that a tree pedigree may have local cycles within nuclear families (in its graph form).

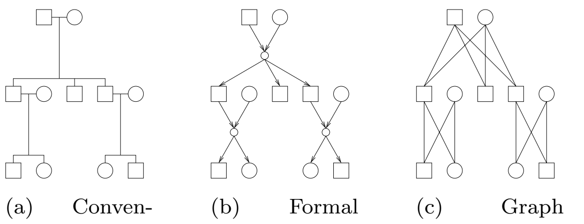


Fig. 2. Three equivalent pedigrees in different forms. The formal form contains mating nodes (smaller circles). In the graph form, each pair of parent-child is connected with an edge.

2. PRELIMINARIES

In this section, we first review the linear system developed in Ref. 12 and 29 for dealing with pedigree data without mutations and errors. We then explain how to modify the linear system for handling mutations and errors, and give the formal definition of HCME.

2.1. The Linear System

Let n denote the number of individuals in the input pedigree and m the number of marker loci of each individual. In this paper, we assume all alleles are bi-allelic (0 or 1). The genotypes of individual j is denoted as a ternary vector $\mathbf{g}_j$ and its kth entry $g_j[k]$ represents the genotype at locus k of individual j:

$$g_j[k] = \begin{cases} 0 & \text{if both alleles are 0's} \\ 1 & \text{if both alleles are 1's} \\ 2 & \text{if the locus is heterozygous} \end{cases}$$

For individual j, we define $\mathbf{p}_j \in GF(2)^m$ as the paternal haplotype of individual j. Each entry $p_j[k]$ of $\mathbf{p}_j$ is defined on $GF(2)$. Clearly, if $g_j[k] = 0$ or 1, then we can derive $p_j[k] = g_j[k]$ directly. To represent the maternal haplotype of individual j, we define $\mathbf{w}_j \in GF(2)^m$ to indicate if each locus of individual j is heterozygous. That is, $w_j[k] = 0$ if $g_j[k] = 0$ or 1, and $w_j[k] = 1$ if $g_j[k] = 2$. Clearly, the summation $\mathbf{p}_j + \mathbf{w}_j$ over $GF(2)$ represents the maternal haplotype of individual j.

Suppose that individual i is a parent of individual j. To unify the representation of the haplotype that j inherited from i, we define a binary vector $\mathbf{d}_{i,j}$ as follows: $\mathbf{d}_{i,j} = 0$ if i is j's father and $\mathbf{d}_{i,j} = \mathbf{w}_j$ if i is j's mother. Therefore, $\mathbf{p}_j + \mathbf{d}_{i,j}$ represents the haplotype that j got from i. We define $h_{i,j} \in GF(2)$ such that $h_{i,j} = 0$ if $\mathbf{p}_j + \mathbf{d}_{i,j}$ is i's paternal haplotype and $h_{i,j} = 1$ otherwise. Then $\mathbf{p}_i + h_{i,j} \cdot \mathbf{w}_i$ represents the haplotype that i passed to j. The binary variables $h_{i,j}$ thus fully describe the inheritance pattern in an ZRHC instance. Using these notations, we can derive an equation over $GF(2)$:

$$p_i[k] + h_{i,j} \cdot w_i[k] = p_j[k] + d_{i,j}[k] \qquad \forall \text{ parent-child pair } (i,j), \forall \text{ locus } k \qquad (1)$$

2.2. Impact of Mutations and Errors

We define a *mutation variable* $\mu_{i,j}[k] \in \mathbb{Z}$, $\mu_{i,j}[k] = 1$ if there is a mutation at locus k when i passes his haplotype to offspring j, and $\mu_{i,j}[k] = 0$ otherwise. For convenience, we make these three vectors symmetric by defining $\mathbf{d}_{j,i} = \mathbf{d}_{i,j}$, $h_{j,i} = h_{i,j}$, and $\mu_{j,i} = \mu_{i,j}$.

Using these notations, we modify Eq. (1) and obtain

$$p_i[k] + h_{i,j} \cdot w_i[k] = p_j[k] + d_{i,j}[k] + I(\mu_{i,j}[k] = 1) \quad (2)$$

where $I(\cdot)$ is the indicator function also defined on $GF(2)$.

An *error variable* $e_j[k] \in \mathbb{Z}$ of individual j at locus k is defined as $e_j[k] = 1$ if the observed genotype $g_j^o[k]$ is different from the actual genotype $g_j[k]$, and $e_j[k] = 0$ if there is no error, *i.e.*, when $g_j^o[k] = g_j[k]$. These errors hinder us from getting correct $p_i[k]$, $p_j[k]$, $w_i[k]$, and $d_{i,j}[k]$ in Eq. (2).

We define $w_j^o[k]$, $p_j^o[k]$, and $d_{i,j}^o[k]$ as $w_j[k]$, $p_j[k]$, and $d_{i,j}[k]$ derived from the observed genotype $g_j^o[k]$. If $g_j^o[k] = 0$ or 1, then $w_j^o[k] = 0$ and $p_j^o[k] = g_j^o[k]$. If $g_j^o[k] = 2$, then $w_j^o[k] = 1$ and $p_j^o[k] = p_j[k]$, which cannot be determined immediately. We define $d_{i,j}^o[k] = 0$ if i is j's father and $d_{i,j}^o[k] = w_j^o[k]$ otherwise.

As we initialize variables that we can derive from the genotype data, we need to consider possible errors. We start with what we can observe, $\mathbf{p}_j^o$ and $\mathbf{w}_j^o$, and derive a *conditional equation* over $GF(2)$:

$$p_i^o[k] + h_{i,j} \cdot w_i^o[k] = p_j^o[k] + d_{i,j}^o[k] + I(\mu_{i,j}[k] = 1)$$
$$\text{if } e_i[k] = e_j[k] = 0 \quad (3)$$

The HCME problem can be formally defined as follows. Given an input pedigree and genotype data $\mathbf{g}_j^o$ for each individual j, find a solution to each $\mathbf{g}_j$, $\mathbf{p}_j$, $\mathbf{w}_j$, $h_{i,j}$, $\mu_{i,j}$ and e_j that satisfies Eq. (2) and (3), and minimizes the *mutation-error score*

$$c_1 \sum\nolimits_{i,j,k} \mu_{i,j}[k] + c_2 \sum\nolimits_{j,k} e_j[k]$$

where $c_1, c_2 > 0$ are weights for mutations and errors. These adjustable weights allow us to change preference between mutations and errors as the rates of mutations and errors may change in different applications. Our default is that $c_1 = 1.5$ and $c_2 = 1$. We assign a bigger weight to each mutation because mutation events typically have a much lower frequency ($\sim 10^{-9}$) than errors (0.1%–5%) do[30]. On the other hand, the larger weight for mutations makes it difficult to detect mutations, *i.e.*, they could easily be replaced by errors especially when the pedigree is shallow.

3. METHOD

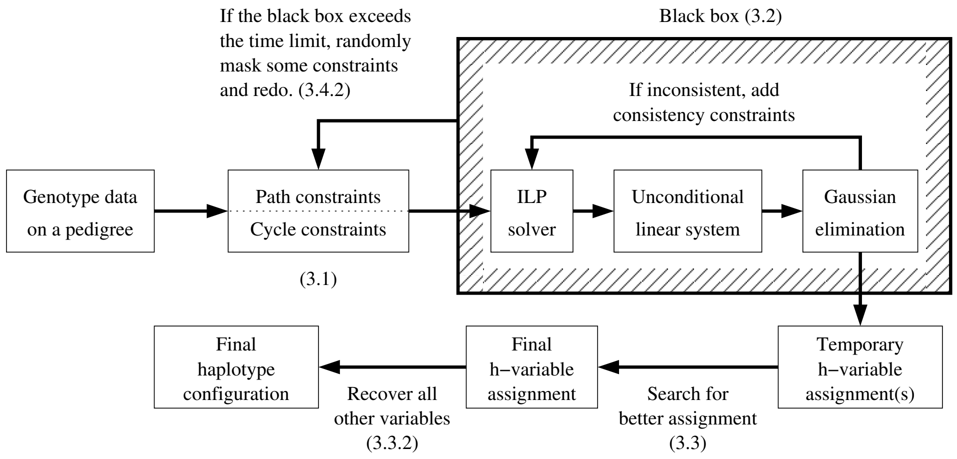
We first construct an equivalent conditional linear system with fewer variables, which can be converted to an ILP instance. The objective function of the ILP instance is still to minimize the mutation-error score. A standard ILP solver GLPK (the GNU Linear Programming Kit from `http://www.gnu.org/softward/glpk`) is invoked to solve the ILP. The ILP solution describes an unconditional linear system, which may or may not be consistent. The consistency of the unconditional linear system can be checked by Gaussian elimination. If it is consistent, we obtain a temporary assignment of the h-variables. If it is not consistent, we add more constraints and then invoke the ILP solver again. The ILP solver together with the Gaussian elimination subroutine will be referred to as the *black box*, which was introduced in our previous work[13]. The temporary h-variable assignment from the black box is often optimal and, in this case, we can compute an optimal haplotype configuration with the assignment. However, because of the loss of information (see Section 3.1.1) due to genotyping errors, the temporary h-variable assignment may sometimes be suboptimal, although it is usually very close to an optimal h-variable assignment. We start from the temporary h-variable assignment and search if there is any better h-variable assignment. We use the best h-variable assignment that we have found to compute the final haplotype configuration.

Occasionally, the black box may have difficulty assigning variables and exceed a predetermined time limit. If the black box exceeds the time limit, we apply some heuristic rules to reduce the size of the ILP instance and then call the black box again.

Fig. 3 shows the flowchart of our method. The construction of path constraints and cycle constraints will be explained in Section 3.1. The black box will be explained in Section 3.2. The search for better h-variables and recovery of haplotype configurations will be explained in Section 3.3. The time limit as well as some implementation issues that are not covered in the flowchart will be explained in Section 3.4.

3.1. Construction of Constraints

There are $O(mn)$ variables and equations in the linear system described in Section 2. As in Ref. 13 and 29, we can convert the system to an equivalent linear system involving only the h-variables, mutation



If the black box exceeds the time limit, randomly mask some constraints and redo. (3.4.2)
Black box (3.2)
If inconsistent, add consistency constraints
Genotype data on a pedigree
Path constraints
Cycle constraints
(3.1)
ILP solver
Unconditional linear system
Gaussian elimination
Final haplotype configuration
Recover all other variables (3.3.2)
Final h−variable assignment
Search for better assignment (3.3)
Temporary h−variable assignment(s)

Fig. 3. The outline of our algorithm. Numbers in parentheses indicate the sections with the corresponding details.

variables, and error variables.

3.1.1. *Equations with Fewer Variables*

The idea is to consider paths in the pedigree (of the graph form) connecting individuals with homozygous markers and derive equality constraints on the h-variables on such paths based on Eq. (3). Consider a locus k and a path $j_0, j_1, \dots, j_r$ in the input pedigree, where individuals j_i and j_{i+1} form a parent-child or child-parent pair. Suppose that $g^o_{j_0}[k]$ and $g^o_{j_r}[k]$ are homozygous, and $g^o_{j_1}[k] = \cdots = g^o_{j_{r-1}}[k] = 2$. We call the path $j_0, j_1, \dots, j_r$ an *all-heterozygous path* at locus k. Since $g^o_{j_0}[k]$ and $g^o_{j_r}[k]$ are homozygous, we have $p^o_{j_0}[k] = g^o_{j_0}[k]$ and $p^o_{j_r}[k] = g^o_{j_r}[k]$. We add up all conditional equations as given in Eq. (3) for all parent-child pairs on path $j_0, j_1, \dots, j_r$ to obtain a *path constraint* connecting j_0 and j_r:

$$\begin{aligned}
&\sum_{i=1}^{r-2} h_{j_i,j_{i+1}} + h_{j_0,j_1} \cdot I(j_1 \text{ is } j_0\text{'s parent}) \\
&\quad + h_{j_{r-1},r} \cdot I(j_{r-1} \text{ is } j_r\text{'s parent}) \\
&= g^o_{j_0}[k] + g^o_{j_r}[k] + \sum_{i=0}^{r-1} d^o_{j_i,j_{i+1}}[k] \qquad (4)\\
&\quad + I\left(\sum_{i=0}^{r-1} \mu_{j_i,j_{i+1}}[k] \text{ is odd}\right) \\
&\quad \text{if } e_{j_i}[k] = 0, i = 0, \dots, r.
\end{aligned}$$

We denote $\mathcal{M} = \bigcup_{i=0}^{r-1}\{\mu_{j_i,j_{i+1}}[k]\}$, $\mathcal{E} = \bigcup_{i=0}^{r}\{e_{j_i}[k]\}$, and $c = g^o_{j_0}[k] + g^o_{j_r}[k] + \sum_{i=0}^{r-1} d^o_{j_i,j_{i+1}}[k]$, and let $\mathcal{H}$ be the collection of h-variables that have coefficient 1 in Eq. (4), *i.e.*, $h_{j_0,j_1} \in \mathcal{H}$ if and only if j_1 is j_0's parent, etc. The path constraint can be represented by the quadruple $(\mathcal{H}, \mathcal{M}, \mathcal{E}, c)$ which denotes that

$$\begin{aligned}
&\textstyle\sum_{h_{i,j}\in\mathcal{H}} h_{i,j} = c + I\left(\sum_{\mu\in\mathcal{M}} \mu \text{ is odd}\right) \\
&\quad \text{if } \textstyle\sum_{e\in\mathcal{E}} e = 0 \qquad (5)
\end{aligned}$$

Note that the observed genotypes along the path may actually contain more than one error, but our path constraint could be satisfied with at most one error. Thus, we may underestimate the number of errors by using such a path constraint. For convenience, we will refer to this inherent limitation of path constraints as the *loss of information*, which is mainly caused by the fact that we do not know the actual genotypes.

Consider a *local cycle* consisting of father i_1, mother i_2, and two adjacent children j_1, j_2. If both parents are heterozygous at locus k, we can obtain four conditional equations from Eq. (3) by replacing i with i_1, i_2, and j with j_1, j_2 (see Fig. 4). The summation of these conditional equations forms a *cycle constraint*:

$$\begin{aligned}
&h_{i_1,j_1} + h_{i_1,j_2} + h_{i_2,j_1} + h_{i_2,j_2} \\
&\quad = w^o_{j_1}[k] + w^o_{j_2}[k] + I\left(\textstyle\sum_{\mu\in\mathcal{M}} \mu \text{ is odd}\right) \\
&\quad \text{if } \textstyle\sum_{e\in\mathcal{E}} e = 0
\end{aligned}$$

where $\mathcal{M} = \{\mu_{i_1,j_1}[k], \mu_{i_1,j_2}[k], \mu_{i_2,j_1}[k], \mu_{i_2,j_2}[k]\}$, $\mathcal{E} = \{e_{i_1}[k], e_{i_2}[k], e_{j_1}[k], e_{j_2}[k]\}$. This constraint will also be denoted as $(\mathcal{H}, \mathcal{M}, \mathcal{E}, c)$ if we let $\mathcal{H} = \{h_{i_1,j_1}, h_{i_1,j_2}, h_{i_2,j_1}, h_{i_2,j_2}\}$ and $c = w_{j_1}[k] + w_{j_2}[k]$.

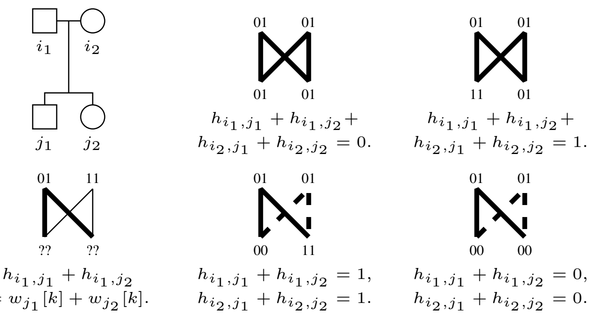

i_1 i_2
j_1 j_2
01 01
01 01
$h_{i_1,j_1} + h_{i_1,j_2} + h_{i_2,j_1} + h_{i_2,j_2} = 0.$
01 01
11 01
$h_{i_1,j_1} + h_{i_1,j_2} + h_{i_2,j_1} + h_{i_2,j_2} = 1.$
01 11
?? ??
$h_{i_1,j_1} + h_{i_1,j_2} = w_{j_1}[k] + w_{j_2}[k].$
01 01
00 11
$h_{i_1,j_1} + h_{i_1,j_2} = 1,$ $h_{i_2,j_1} + h_{i_2,j_2} = 1.$
01 01
00 00
$h_{i_1,j_1} + h_{i_1,j_2} = 0,$ $h_{i_2,j_1} + h_{i_2,j_2} = 0.$

Fig. 4. If both parents are heterozygous, and at least one child is heterozygous, then we will a cycle constraint. If both parents are homozygous, then there is no constraint. Otherwise, we will obtain either one or two path constraints, depending on the situation.

Let us look at the path constraints generated in the above nuclear family more closely. If both parents in the local cycle are homozygous at locus k, then the corresponding path constraint from one parent to the other will consist of no h-variables, and thus no path constraint will be derived. If one parent is heterozygous, the other is homozygous and both children are heterozygous at locus k, then we can derive a path constraint from the homozygous parent through one child, the other parent, the other child, and back to the homozygous parent. If there is exactly one homozygous parent and one homozygous child, the path constraint should be derived through the heterozygous child and the other parent. Otherwise, no path constraint will be derived for this local cycle.

For each locus and every pair of homozygous markers, we construct a path constraint as above. Since the pedigree is a tree, the number of such path constraints is at most $O(mn)$. Similarly, for each locus and local cycle, if both parents are heterozygous at the locus, we construct a cycle constraint as above. The number of such cycle constraints is also bounded by $O(mn)$. Let $\mathcal{S}$ denote the set of these constraints. The results in Ref. 29 show that the linear system formed by the above constraints (without the conditions) in terms of the h-variables is equivalent to the linear system defined by Eq. (3) (without the conditions) in terms of the h- and p-variables. In HCME with mutations and errors, a feasible assignment to the h-, μ-, and e-variables, can be extended to a feasible solution to all the h-, p-, μ- and e-variables.

Note that, loci with missing alleles could possibly be included in the linear system in Eq. (3) (as p-variables). However, they are excluded from the above path/cycle constraints on h-variables. Some of the missing alleles will be imputed after the h-variables are determined.

3.1.2. *The ILP Instance*

We construct an ILP instance for HCME based on the above linear system in h-variables. Recall that the objective function of the ILP is the mutation-error score

$$c_1 \sum\nolimits_{i,j,k} \mu_{i,j}[k] + c_2 \sum\nolimits_{j,k} e_j[k]$$

In our ILP instance, the path/cycle constraint $(\mathcal{H}, \mathcal{M}, \mathcal{E}, c)$ as given in Eq. (5) is actually modified as

$$\sum\nolimits_{h_{i,j}\in\mathcal{H}} h_{i,j} = c, \quad \text{if } \sum\nolimits_{\mu\in\mathcal{M}} \mu = \sum\nolimits_{e\in\mathcal{E}} e = 0 \tag{6}$$

with three technical reasons. First, there are rarely two or more mutations on a locus[31]. Second, Eq. (5) is not accurate because of the loss of information, and thus the ILP solver produces suboptimal intermediate results anyway. Third, Eq. (6) generates a smaller ILP instance and is more efficient.

For each path/cycle constraint $(\mathcal{H}, \mathcal{M}, \mathcal{E}, c)$ in $\mathcal{S}$, we introduce a binary *equation variable* as in Ref. 13

$$E_{\mathcal{H}} = \sum\nolimits_{h_{i,j}\in\mathcal{H}} h_{i,j} \tag{7}$$

and require that the corresponding quadruple $(E_{\mathcal{H}}, \sum_{\mu\in\mathcal{M}} \mu, \sum_{e\in\mathcal{E}} e, c)$ must not be $(0,0,0,1)$ or $(1,0,0,0)$.

$$\begin{cases} E_{\mathcal{H}} + \sum_{\mu\in\mathcal{M}} \mu + \sum_{e\in\mathcal{E}} e + (1-c) \geq 1 \\ (1-E_{\mathcal{H}}) + \sum_{\mu\in\mathcal{M}} \mu + \sum_{e\in\mathcal{E}} e + c \geq 1 \end{cases} \tag{8}$$

In other words, if there are no mutations and errors, the equation in Eq. (6) must hold. The final ILP instance includes all the constraints in Eq. (8). The number of constraints is clearly bounded by $O(mn)$.

3.2. The Black Box

The black box consists of two elements: the ILP solver and the Gaussian elimination subroutine. After we set up the ILP instance as above, we invoke the ILP solver. The ILP solver will return an assignment of the mutation variables, error variables and equation variables. The assignment of the mutation and error variables is not accurate because of the loss of information and will be ignored. The assignment of the equation variables implies an unconditional linear system of the h-variables. The linear system can be solved by Gaussian elimination and we then obtain a solution of the h-variables. The linear system is usually consistent. If the linear system is not consistent, our Gaussian elimination subroutine will detect inconsistent equation variables. We then add some *consistency constraints* to the ILP instance and invoke the ILP solver again. Consistency constraints make sure that the assignments of the involved equation variables in the constraints will be consistent with each other. Inconsistency may happen repeatedly, and we keep adding consistency constraints until the assignment of the equation variables is consistent. The detail of consistency constraints can be found in our previous work[13]. Note that by not including consistency constraints at the beginning, we enhance the efficiency of our program without losing any accuracy.

3.3. Recovery of Haplotype Configurations

3.3.1. *Search for a Better h-variable Assignment*

The temporary h-variable assignment retrieved from Gaussian elimination is usually an optimal assignment. If it is not, it usually differs from an optimal assignment at very few h-variables. We evaluate an h-variable assignment by computing the minimum possible mutation-error score which is consistent with the h-variable assignment. The computation of mutation-error score is explained in Section 3.3.2.

Let t be the number of h-variables and $\mathbf{H}^{(0)} = [h_{i_1,j_1}, h_{i_2,j_2}, \ldots, h_{i_t,j_t}]$ be the temporary assignment of h-variables obtained from Gaussian elimination. Let u_r be a unit vector on $GF(2)^t$ with a 1 on the rth entry. Assume the $\mathbf{H}^{(k)}$ is the h-variable assignment after k rounds of searching. In the $(k+1)$st round, we consider t possible h-variable assignment $\mathbf{H}_r^{(k+1)} = \mathbf{H}^{(k)} + u_r$, $r = 1, \ldots, t$. We compute the minimum mutation-error score for each assignment, and $\mathbf{H}^{(k+1)}$ will be the assignment with the minimum score among $\mathbf{H}^{(k)}$ and $\mathbf{H}_r^{(k+1)}$, $r = 1, \ldots, t$. This search continues until $\mathbf{H}^{(k)} = \mathbf{H}^{(k+1)}$ for some k, and $\mathbf{H}^{(k)}$ will be the final h-variable assignment.

3.3.2. *Computing Mutation-Error Scores and Haplotype Configurations*

In this section, we assume that the h-variable assignment is fixed. Once the h-variables are assigned, we need to assign errors and mutations that minimize the mutation-error score as given in Eq. (7). We use the dynamic programming method to find the minimum mutation-error score as follows. Here, each locus is considered separately.

We pick an arbitrary node (individual) as the root of the input (tree) pedigree. For each node, we consider all four possible genotypes and compute the best mutation-error score of the subtree under the node for each genotype. The score could be computed with the score of children with respect to the rooted tree of all possible genotypes. We compute the score iteratively from the leaves to the root. This dynamic programming procedure determines an optimal assignment of all mutations and errors by a simple backtracking subroutine, where ties are broken arbitrarily.

After assigning errors, we mark genotypes where errors are assigned as missing alleles. For each pair of alleles of a non-missing marker, if there is no error and the genotype is homozygous, we mark the alleles as *confirmed*. We start from confirmed alleles, update parent's/children's alleles with the given h-variables and the mutation assignment, and confirm these newly updated alleles. We keep updating and confirming alleles until there are no more alleles that we can update. Alleles that are not confirmed remain unknown. After updating all the alleles, we obtain the whole haplotype configuration with the locations of mutations and correction of errors. We output the configuration if the h-variable assignment is the final assignment.

3.4. A Few Implementation Issues

3.4.1. *Free Variables*

Occasionally, the solution of the unconditional linear system obtained from ILP solver is not unique. If there are two or more solutions, then we have multiple h-variable assignments as starting points of the search. We simply do the search described in Section 3.3 for all starting points and select the best one. We record h-variable assignments that have been searched, and avoid redundant search.

3.4.2. *Time Limit*

Sometimes, the ILP solver may have difficulty in determining errors, mutations, and equation variables. It is usually because there are some "troublesome" loci with errors. If the ILP solver fails to assign errors to these loci and attempts to assign errors to other loci that do not actually have errors, it may take a very long time for the ILP solver to find a feasible assignment, or the ILP solver may keep returning inconsistent assignments of equation variables. To avoid these situations, we set a time limit. The default time limit is chosen as 3 minutes based on empirical observations. When the black box fails to return an assignment of h-variables within the time limit, we terminate the black box. We modify the ILP instance as follows and then run the black box again. We repeat this process if the black box keeps exceeding the time limit.

The first time the black box exceeds the time limit, we check if we can fix the values of some equation variables to reduce the ILP solver's work. For example, suppose that there are 9 or more constraints extracted from different loci with the same $\mathcal{H}$ and c, *i.e.*, $E_{\mathcal{H}} = c$ is suggested 9 times with different conditions, and there are none or only 1 constraint supporting $E_{\mathcal{H}} = 1 - c$. The odds strongly favor $E_{\mathcal{H}} = c$ and thus we can simply assign c to $E_{\mathcal{H}}$. Let us call such a constraint that suggests $E_{\mathcal{H}} = 1 - c$ *conflicting.* If there are two or more conflicting constraints (over different paths or cycles) extracted from the same locus, then the locus is considered *troublesome.* We *mask* all troublesome loci by dropping all constraints extracted from them.

If the black box exceeds the time limit the second time, we randomly select a locus and mask it. If the black box keeps exceeding the time limit, we will increase the number of masked loci by 1 each time. Occasionally, the black box may keep running and failing for a long time. We thus have a global time limit. If our program exceeds the global time limit, we terminate the computation and concede. The default global time limit is currently set as 30 minutes.

4. EXPERIMENTAL RESULTS

We have implemented our algorithms in C, denoted as MePhase. In this section, we test MePhase on both simulated data and real data to perform an empirical evaluation of its accuracy and efficiency. For simulated data, we generate both tree pedigrees and genotype data randomly. We design experiments to test how the pedigree topology and data quality (*i.e.*, the missing and genotyping error rates) may affect the performance of MePhase. We do not consider a very large number of loci since the zero-recombinant assumption holds only for tightly linked markers. For real data, we use the SNP microarray data published by Wirtenberger *et al.*[32] They genotyped 16 members of a three-generation family using the GeneChip Human Mapping 10K Array (Affymetrix). Since Wirtenberger *et al.* did not delete Mendelian inheritance errors in their published data, the data along with their report of recombinant regions are ideal for us to test MePhase. According to Ref. 32, there are 6.24% missing alleles and 0.29% Mendelian inheritance errors detected in all family trios in the data. However, Hao *et al.* have reported that the genotyping error rate of GeneChip Mapping 10K Array is about 0.1%[30], which is much smaller than the error rate given in Ref. 32. We think that the number of Hao *et al.* is more accurate because they focused on the estimation the of error rate while Wirtenberger *et al.* did not. Our results also support the error rate given by Hao *et al.*

4.1. Simulated Data

Thomas *et al.* have proposed algorithms to generate tree pedigrees uniformly randomly with specific numbers of individuals and mating notes[33]. Their algorithms have been implemented in Java, which will be used to generate pedigrees in this paper. Zou *et al.* showed that the size of a nuclear family may affect the accuracy of error detection[28]. Thus, we consider the average nuclear family size as a parameter along with pedigree size and analyze how nuclear family and pedigree sizes may affect the accuracy and efficiency of our algorithms. Let f be the average nuclear family size and n again denote the number of individuals. Because tree pedigrees are considered, f is determined by n and the number of mating nodes:

$$f = \frac{n + \text{\# of mating nodes} - 1}{\text{\# of mating nodes}}$$

Therefore, we can alter the number of mating nodes to obtain pedigrees of different average nuclear family sizes. We will generate pedigrees with $n = 29, 52, 67, 82$ and $f = 3, 4, 5, 6$. For each of the 16 combinations of n and f, we generate 5 random pedigrees. This results in 80 different pedigrees in total. Note that for $n = 29$, we could only achieve $f = 5.67$ instead of 6 because the number of mating nodes should be an integer.

To generate genotype data, we use haplotype data downloaded from HapMap (`http://hapmap.ncbi.nlm.nih.gov/downloads/phasing/2009-02_phaseIII/HapMap3_r2/`). For each run of the experiment, we randomly select an interval of SNPs. Then for each founder in the pedigree, we randomly pick two haplotypes to form the genotype of the founder. Each individual randomly passes one of its two haplotypes to each of its offsprings, where mutations are incorporated randomly according to the mutation rate. This results in a genotype for each individual. Finally, missing alleles and genotyping errors are randomly added to the genotypes according to their rates.

We run MePhase on each simulated data and compare the haplotype configurations given by MePhase with the true haplotype configurations of the simulated data. We will consider the ability of MePhase in imputing missing alleles and detecting mutations and genotyping errors. When the location of a detected error is slightly different from that of the true error in the pedigree, we say that the error is *shifted*. Table 1 shows the impact of n and f on both efficiency and accuracy of MePhase in phasing as well as error detection and correction. MePhase infers haplotypes with a very high accuracy in all settings. In general, a bigger f leads to denser constraints, less freedom of haplotype assignments, and a higher accuracy. On the other hand, a smaller f leads to more founders. When an error event happens to the genotype of a founder, there is a good chance that the erroneous genotype does not violate the Mendelian law of inheritance. This is likely to keep the error undetectable. Therefore, a smaller f will lead to more founders and more undetected errors. Table 1 also clearly illustrates that a bigger n leads to a longer running time. Moreover, a bigger f also leads to a longer running time, since a bigger f leads to more path and cycle constraints.

Most mutations (85%–95%) are explained by errors because we give errors a smaller weight. A mutation on a founder will often be explained by an error, or remain undetectable if the Mendelian law of inheritance is not violated. A mutated allele will for sure be explained by an error if it does not get passed to any offspring. Most mutations can be explained by errors with some small shifts within the pedigree, especially when the mutations are near founders or children with no offsprings. Our combinatorial optimization model is incapable of catching such mutations precisely.

As shown in Table 1, MePhase may produce a suboptimal solution, especially when most genotypes are homozygous, since it may return an h-variable assignment that is locally optimal during the search for a better h-variable assignment. The suboptimal h-variable assignment usually differs from the true assignment at many h-variables, but requires only one more error than the true haplotype solution.

In our experiment, we observed that if the black box in MePhase fails to return an assignment within the time limit once, it will tend to fail many times. The standard ILP solver GLPK uses a branch-and-bound algorithm to find integral solutions. When GLPK falls into a bad branch, it may take very long time for GLPK to get out of the branch. This creates a big variance in running time. For bigger n and f, we had to give MePhase a longer global time limit.

The quality (*i.e.*, the missing and genotyping error rates) of data may also have an impact on both accuracy and efficiency of MePhase. We run MePhase on 5 pedigrees of $n = 29$ and $f = 4$ with various missing and error rates as shown in Table 2. A large variance in running time is observed which might explain why there is no clear impact of data quality on running time. The missing rate has some impact on the running time, while the error rate does not affect the running time much. However, the impact of data quality on accuracy is small but noticeable. When the missing rate goes higher, the accuracies in phasing, error correction and missing

Table 1. The impact of pedigree size n and average nuclear family size f on the efficiency and accuracy of MePhase in phasing and error detection and correction. Here, the number of loci $m = 50$, the missing rate $= 0$, and the mutation rate $= 1$ per pedigree on average. For testing the accuracy of MePhase and the optimality of its solution, we ran 200 replicates for each pedigree and set the global time as 10 minutes. For testing the time efficiency and failure rate, we ran 30 replicates for each pedigree and set the global time limit as 30 minutes (but 180 minutes for $(n, f) = (52, 6), (62, 5), (62, 6), (82, 5), (82, 6)$). Clearly, MePhase rarely fails to find a solution under this setting.

Average nuclear family size	Pedigree size (n)	Recovered errors	Shifted errors	Undetected errors	Correctly phased markers	Suboptimal solution	Average time (sec)	Failure rate
$f = 3$	29	67.1%	9.9%	23.0%	99.2%	0.0%	0.03	0.0%
	52	67.6%	9.2%	23.2%	99.5%	0.3%	1.37	0.0%
	67	66.2%	8.9%	25.0%	99.7%	0.9%	1.88	0.0%
	82	65.9%	8.6%	25.4%	99.7%	0.6%	4.99	0.0%
$f = 4$	29	83.9%	5.8%	10.3%	99.3%	0.2%	4.71	0.0%
	52	82.7%	5.6%	11.7%	99.6%	0.2%	18.3	2.0%
	67	83.4%	5.0%	11.6%	99.7%	0.8%	30.3	0.7%
	82	83.4%	5.4%	11.3%	99.8%	0.8%	49.1	0.0%
$f = 5$	29	91.7%	3.7%	4.5%	99.3%	0.0%	4.92	0.7%
	52	90.7%	2.9%	6.4%	99.5%	0.5%	46.5	1.3%
	67	91.1%	3.0%	5.9%	99.7%	0.7%	402	0.7%
	82	90.8%	3.4%	5.8%	99.9%	0.9%	498	2.7%
$f = 6$	29	93.1%	2.8%	4.1%	99.3%	0.1%	22.1	1.3%
	52	94.5%	2.1%	3.4%	99.6%	0.4%	661	0.0%
	67	94.4%	1.8%	3.8%	99.7%	0.6%	619	0.0%
	82	94.4%	2.0%	3.6%	99.8%	1.0%	740	2.7%

Table 2. The impact of data quality on accuracy and efficiency. Here, $n = 29$, $f = 4$, $m = 50$, the mutation rate $= 1$ per pedigree on average, the global time limit is set as 30 minutes, and we ran 30 replicates for each setting.

Error rate	Missing rate	Correctly recovered errors	Correctly imputed missing alleles	Correctly phased markers	Suboptimal solution	Average time	Failure rate
0.0%	0%	–	–	99.7%	0.0%	2.30	0.0%
	5%	–	79.3%	98.6%	0.0%	58.2	3.3%
	10%	–	77.4%	97.2%	0.0%	82.3	6.0%
	20%	–	72.2%	93.5%	0.0%	97.3	11.3%
0.5%	0%	85.0%	–	99.6%	0.0%	13.0	0.0%
	5%	83.3%	78.5%	98.4%	0.7%	24.2	5.3%
	10%	83.3%	76.6%	97.1%	0.7%	65.0	6.7%
	20%	79.4%	72.1%	93.4%	0.7%	70.9	8.0%
1.0%	0%	85.3%	–	99.3%	0.0%	2.37	0.7%
	5%	83.2%	78.8%	98.2%	0.0%	44.5	1.3%
	10%	79.5%	76.6%	96.7%	0.0%	64.1	6.7%
	20%	75.1%	72.0%	93.2%	0.7%	39.9	7.3%

allele imputation slightly decrease. The genotyping error rate has a smaller impact on the accuracy of MePhase, but the trend is clear.

4.2. Real Data

Fig. 5(a) shows the pedigree used in Ref. 32. Individuals 17 and 18 (*i.e.*, the founders) are missing, and they have only two children (*i.e.*, individuals 1 and 13). Therefore, these two children are under no constraints. We partition the pedigree into pedigrees A and B, and run MePhase on both pedigrees separately.

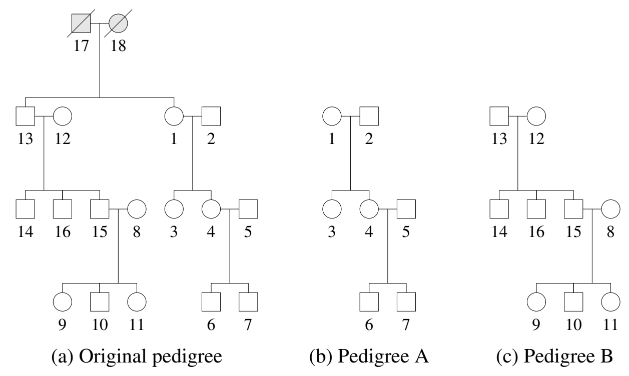


Fig. 5. The diagram in (a) shows the pedigree of the SNP microarray data reported by Wirtenberger *et al.*[32] Since the two founders are completely missing, we divide the pedigree into two disjoint pedigrees A and B.

Wirtenberger *et al.* reported a set of recombinant regions (*i.e.*, intervals where a recombinant might be located) in the SNP microarray data. We preprocess the SNP data by eliminating all SNP loci in the recombinant regions. A set of loci between two neighboring recombinant regions is considered as a *block*. We then run MePhase on each block separately.

Table 3 shows the number of errors that MePhase found in both pedigrees A and B. The table also shows the number of *hidden errors*, which are errors that cannot be found by checking the Mendelian law of inheritance within nuclear families, found by MePhase. MePhase also identified a mutation on chromosome 6. The overall genotyping error rate in non-recombinant regions found by MePhase is 0.175%. If we exclude the hidden errors, the error rate would be reduced to 0.146%, which is close to the error rate reported in Ref. 30.

To estimate the accuracy of our reported numbers, we also run experiments with simulated data on pedigrees A and B. The result (data not included here) shows that MePhase is able to detect 85% of the errors on pedigree A and 87% of the errors on pedigree B. By extrapolation, we estimate that the genotyping errors in the non-recombinant regions of this SNP microarray data is close to 0.2% taking into account the errors that MePhase might have failed to detect.

5. CONCLUSION

We have introduced a combinatorial optimization model for haplotype inference on pedigrees in the presence of mutations and genotyping errors that generalizes the previous models in the literature. We have designed and implemented an heuristic algorithm for the model based on the previously developed system of linear equations and ILP. Our experimental results on simulated data demonstrate that our program can infer haplotypes with a very high accuracy, impute most missing alleles, detect and correct most genotyping errors, and identify some mutations (although many mutations could be confused with errors).

Acknowledgements

We would like to thank Dr. Minzhu Xie for several discussions and comments on the manuscript. The research is supported in part by NIH grant 2R01LM008991 and NSF grant IIS-0711129.

References

1. The Internaltional HapMap Consortium. The international HapMap project. *Nature* 2003; **426**: 789–796.
2. Li J and Jiang T. A survey on haplotype algorithms for tightly linked markers. *J Bioinform Comput Biol* 2008; **6**(1):241–259.
3. Gusfield D. Inference of haplotypes from samples of diploid populations: complexity and algorithms. *J Comput Biol* 2001; **8**(3): 305–323.
4. Niu T, Qin ZS, Xu X, and Liu JS. Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms. *Am J Hum Genet* 2002; **70**: 157–169.
5. Stephens M, Smith NJ, and Donnelly P. A new statistical method for haplotype reconstruction from population data. *Am J Hum Genet* 2001; **68**: 978–989.
6. Abecasis GR, Cherny SS, Cookson WO, and Cardon LR. Merlin — rapid analysis of dense genetic maps using sparse gene flow trees. *Nat Genet* 2002; **30**(1): 97–101.
7. Lander ES and Green P. Construction of multilocus genetic linkage maps in humans. *Proc Natl Acad Sci USA* 1987; **84**(8): 2363–2367.
8. Li J and Jiang T. Efficient inference of haplotypes from genotypes on a pedigree. *J Comput Biol* 2003; **1**(1): 41–69.
9. Li J and Jiang T. Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by integer linear programming. *J Comput Biol* 2005; **12**(6): 719–739.
10. Qian D and Beckmann L. Minimum-recombinant haplotyping in pedigrees. *Am J Hum Genet* 2002; **70**(6): 1434–1445.
11. Sobel E, Lange K, O'Connell JR, and Weeks DE. Haplotyping algorithms. *Genetic Mapping and DNA Sequencing* 1996; **81**(3): 89–110.
12. Xiao J, Liu L, Xia L, and Jiang T. Fast elimination of redundant linear equations and reconstruction of recombination-free mendelian inheritance on a pedigree. *18th Annual ACM-SIAM Symposium on Descrete Algorithms* 2007; 655–664.
13. Wang WB and Jiang T. Efficient inference of haplotypes from genotypes on a pedigree with mutations and missing alleles. *CPM* 2009; 353–367.
14. Zhang K, Sun F, and Zhao H. HAPLORE: A program for haplotype reconstruction in general pedigrees without recombination. *Bioinformatics* 2005; **21**(1): 90–103.
15. Wang S, Kidd KK, and Zhao H. On the use of DNA pooling to estimate haplotype frequencies. *Genetic*

Table 3. The number of blocks, loci, and detected errors on each chromosome in each pedigree. The asterisk indicates that a mutation is reported on chromosome 6.

			Pedigree A		Pedigree B	
Chromosome number	Number of blocks	Number of loci	Errors found	Hidden errors	Errors found	Hidden errors
1	26	372	2	1	8	3
2	23	484	2	0	9	1
3	19	385	0	0	10	1
4	22	428	7	0	4	0
5	19	406	2	1	7	0
6*	16	508	7	0	4	1
7	21	353	7	1	8	1
8	13	307	1	0	8	1
9	15	307	1	0	3	0
10	22	245	2	1	5	2
11	17	406	7	2	13	1
12	17	312	2	1	6	1
13	18	294	2	0	9	1
14	13	182	0	0	3	0
15	12	197	3	2	4	0
16	6	135	2	1	2	0
17	6	93	1	0	4	3
18	9	231	2	0	5	2
19	4	29	0	0	0	0
20	7	105	0	0	2	0
21	5	141	2	0	0	0
22	2	6	0	0	0	0

Epidemiology 2003; **24**(1): 74–82.

16. Yang Y, Zhang J, Hoh J, Matsuda F, Xu P, Lathrop M, and Ott J. Efficiency of single-nucleotide polymorphism haplotype estimation from pooled DNA. *Proc Natl Acad Sci USA* 2002; **100**(12): 7225–7230.
17. Badaeva TN, Malysheva DN, Korchagin VI, and Ryskov AP. Genetic variation and *de novo* mutations in the parthenogenetic caucasian rock lizard *darevskia unisexualis*. *PLoS ONE* 2008; **3**(7): e2730.
18. Ellegren H. Microsatellite mutations in the germline: implications for evolutionary inference. *Trends in Genetics* 2000; **16**(12): 551–558.
19. Olson TM, Doan TP, Kishimoto NY, Whitby FG, Ackerman MJ, and Fananapazir L. Inherited and *de novo* mutations in the cardiac actin gene cause hypertrophic cardiomyopathy. *J Mol Cell Cardiol* 2000; **32**(9): 1687–1694.
20. Abecasis GR, Cherny SS, and Cardon LR. The impact of genotyping error on family-based analysis of quantitative traits. *Eur J Hum Genet* 2001; **9**: 130–134.
21. Douglas JA, Boehnke M, and Lange K. A multipoint method for detecting genotyping errors and mutations in sibling-pair linkage data. *Am J Hum Genet* 2000; **66**: 1287–1297.
22. Lebrec JJ, Putter H, Houwing-Duistermaat JJ, and van Houwelingen HC. Influence of genotyping error in linkage mapping for complex traits — an analytic study. *BMC Genetics* 2008; **9**(57).
23. Kirk KM and Cardon LR. The impact of genotyping error on haplotype reconstruction and frequency estimation. *Eur J Hum Genet* 2002; **10**: 616–622.
24. Moskvina V, Craddock N, Holmans P, Owen MJ, and O'Donovan MC. Effects of differential genotyping error rate on the type I error probability of case-control studies. *Human Heredity* 2006; **61**: 55–64.
25. Tapadar P, Ghosh S, and Majumder PP. Haplotyping in pedigrees via a genetic algorithm. *Human Heredity* 2000; **50**(1): 43–56.
26. Douglas JA, Skol AD, and Boehnke M. Probability of detection of genotyping errors and mutations as inheritance inconsistencies in nuclear-family data. *Am J Hum Genet* 2002; **70**(2): 487–495.
27. O'Connell JR and Weeks DE. PedCheck: A program for identification of genotype incompatibilities in linkage analysis. *Am J Hum Genet* 1998; **63**: 259–266.
28. Zou G, Pan D, and Zhao H. Genotyping error detection through tightly linked markers. *Genetics* 2003; **164**: 1161–1173.
29. Liu L and Jiang T. A linear-time algorithm for reconstructing zero-recombinant haplotype configuration on pedigrees without mating loops. *J Comb Optim* 2008; **19**: 217–240.
30. Hao K, Li C, Rosenow C, and Wong WH. Estimation of genotype error rate using samples with pedigree information — an application on the GeneChip Mapping 10K array. *Genomics* 2004; **84**: 623–630.
31. Kimura M and Crow JF. The number of alleles that can be maintained in a finite population. *Genetics* 1964; **49**: 725–738.
32. Wirtenberger M, Hemminki K, Chen B, and Burwinkel B. SNP microarray analysis for genome-wide detection of crossover regions. *Human Genetics* 2005; **117**: 389–397.
33. Thomas A and Canning C. Simulating realistic zero loop pedigrees using a bipartite prüfer code and graphical modelling. *Math Med Biol* 2004; **21**(4): 335–345.