# An Exact Solution for Finding Minimum Recombinant Haplotype Configurations on Pedigrees with Missing Data by Integer Linear Programming *(Extended Abstract)*

Jing Li [*] Tao Jiang [†]

## ABSTRACT

We study the problem of reconstructing haplotype configurations from genotypes on pedigree data with missing alleles under the Mendelian law of inheritance and the minimum recombination principle, which is important for the construction of haplotype maps and genetic linkage/association analysis. Our previous results show that the problem of finding a *minimum-recombinant haplotype configuration (MRHC)* is in general NP-hard. The existing algorithms for MRHC either are heuristic in nature and cannot guarantee optimality, or only work under some restrictions (on *e.g.* the size and structure of the input pedigree, the number of marker loci, the number of recombinants in the pedigree, *etc.*). In addition, most of them cannot handle data with missing alleles and, for those that do consider missing data, they usually do not perform well in terms of minimizing the number of recombinants when a significant fraction of alleles are missing. In this paper, we develop an effective integer linear programming (ILP) formulation of the MRHC problem with missing data and a branch-and-bound strategy that utilizes a partial order relationship (and some other special relationships) among variables to decide the branching order. The partial order relationship is discovered in the preprocessing of constraints by considering unique properties in our ILP formulation. A directed graph is built based on the variables and their partial order relationship. By identifying and collapsing the strongly connected components in the graph, we may greatly reduce the size of an ILP instance. Non-trivial (lower and upper) bounds on the optimal number of recombinants are introduced at each branching node to effectively prune the search tree. When multiple solutions exist, a best haplotype configuration is selected based on a maximum likelihood approach. Our results on simulated data show that the algorithm could recover haplotypes with 50 loci from a pedigree of size 29 in seconds on a standard PC. Its accuracy is more than 99.8% for data with no missing alleles and 98.3% for data with 20% missing alleles in terms of correctly recovered phase information at each marker locus. As an application of our algorithm to real data, we present some test results on reconstructing haplotypes from a genome-scale SNP data set consisting of 12 pedigrees that have 0.8% to 14.5% missing alleles.

[*]Department of Computer Science, University of California - Riverside. jili@cs.ucr.edu. Research supported by NSF grant CCR-9988353.

[†]Department of Computer Science, University of California - Riverside and Shanghai Center for Bioinformatics Technology. jiang@cs.ucr.edu. Research supported by NSF Grants CCR-9988353, ITR-0085910 and CCR-0309902, and National Key Project for Basic Research (973) 2002CB512801.

## Categories and Subject Descriptors

J.3 [**Computer Applications**]: Life & Medical Sciences; G.1.6 [**Mathematics of Computing**]: NUMERICAL ANALYSIS—*Optimization*

## General Terms

Algorithms

## Keywords

Haplotyping, pedigree analysis, recombination, missing data imputation, integer linear programming, branch-and-bound algorithm

## 1. INTRODUCTION

With the completion of the Human Genome Project [12, 26], an (almost) complete human genomic DNA sequence has become available, which is essential to the understanding of the functions and characteristics of human genetic material. An important next step in human genomics is to determine genetic variations among humans and the correlation between genetic variations and phenotypic variations (such as disease status, quantitative traits, *etc.*). To achieve this goal, an international collaboration, namely, the international HapMap project, was launched in October, 2002. The main objective of the HapMap project is to identify the *haplotype* structure of humans and common haplotypes among populations. However, the human genome is a diploid and, in practice, haplotype data are not collected directly, especially in large scale sequencing projects (mainly) due to cost considerations. Instead, genotype data are collected routinely in large sequencing projects. Hence, efficient and accurate computational methods and computer programs for the inference of haplotypes from genotypes are highly demanded.

The input data for haplotype reconstruction can be divided into three categories: SNP segments from an individual [15, 17] or pooled samples [20], genotype data with *pedigree* information, and genotype data without pedigree information (also called *population* data sometimes) [7, 8, 10, 18, 24]. A recent comprehensive review of computational methods for haplotype inference can be found in [2]. We are only interested in genotype data with pedigrees. It is generally believed that haplotypes inferred from pedigrees are more accurate than those from population data. Moreover, some family based statistical association tests such as TDT (*i.e. Transmission Disequilibrium Test*) and its variants (*e.g.* [22, 29] among others) require access to haplotype information of each member in a pedigree. The existing computational methods for haplotyping pedigree data can be divided into into two categories: statistical methods and rule-based (*i.e.* combinatorial) methods. Statistical approaches (*e.g.* [16, 23]) estimate haplotype frequencies in addition to the haplotype configuration for each individual, but they are usually very time consuming and thus cannot handle large (in many cases, moderately large) data sets. On the other hand, rule-based approaches are usually very fast, although they do not normally provide numerical assessments of the reliability of their results. Nonetheless, by utilizing some reasonable biological assumptions, such as the *minimum recombination principle*, rule-based methods have proven to be powerful and practical [19, 21, 25, 27]. The minimum recombination principle basically says that genetic recombination is rare for closely linked markers and thus haplotypes with fewer recombinants should be preferred in a haplotype reconstruction [19, 21]. The principle is well supported by practical data. For example, recently published experimental results [5, 9, 11] demonstrate that, in the case of human, the number of distinct haplotypes is very limited. Moreover, the genomic DNAs can probably be partitioned into long blocks such that recombination within each block is rare or even nonexistent.

## 1.1 Previous Work on Rule-Based Haplotype Reconstruction on Pedigrees

Qian and Beckmann [21] proposed a rule-based algorithm to reconstruct haplotype configurations for pedigree data, based on the minimum recombination principle. (From now on, we refer to their algorithm as MRH.) Given a pedigree and the genotype information for each member of the pedigree (with possibly missing alleles), MRH attempts to find a haplotype configuration for each member such that the total number of recombinants (or recombination events) in the whole pedigree is minimized. We call the above problem the *minimum-recombinant haplotype configuration (MRHC)* problem. In recent papers [6, 13, 14], we showed that MRHC is in general NP-hard, even for pedigrees without mating loops, and developed an iterative heuristic algorithm, called *block-extension*, for MRHC that is much more efficient than MRH. Our preliminary experiments showed that the algorithm block-extension is often able to compute an optimal solution or nearly optimal solution when the minimum number of recombinants required is small [13, 14]. However, its performance deteriorates significantly when the input data requires more (*e.g.* 4 or more) recombinants. We have also devised an efficient exact algorithm based on Gaussian elimination for solving MRHC on pedigree data that requires no recombinants. In fact, the algorithm can find all haplotype configurations incurring no recombinants [13, 14]. More recently, two dynamic programming algorithms [6] are developed for (general) pedigrees of small sizes and loopless pedigrees with a small number of marker loci.

However, these existing algorithms for MRHC either are heuristics in nature (*e.g.* MRH and block-extension) and cannot guarantee optimality, or only work under some restrictions (on *e.g.* the size and structure of the input pedigree, the number of marker loci, the number of recombinants in a pedigree, *etc.*). Furthermore, most of them cannot handle missing data and, for those that consider missing data, their performance in terms of minimizing the number recombinants drops significantly in the presence of a moderately large amount of missing alleles. In practice, pedigree data often contains a significant amount of missing alleles. For example, as much as 14.5% of the alleles belonging to a block could be missing in the pedigree data studied in [9]. Some of the live-stock pedigree data that we have examined recently contained an even larger fraction of missing alleles. Unfortunately, consistent imputation of missing alleles is NP-hard even if we do not care about haplotypes and recombination [1].

## 1.2 Our Results

In this paper, we develop an effective integer linear programming (ILP) formulation of MRHC with missing alleles that integrates missing data imputation and haplotype inference, and a branch-and-bound strategy that utilizes a partial order relationship (and some other special relationships) among variables to decide the branching order. The partial order relationship is discovered in the preprocessing of constraints by taking advantage of some special properties in our ILP formulation. A directed graph is built based on the variables and their partial order relationship. By identifying and collapsing strongly connected components in the graph, we may greatly reduce the size of an ILP instance. Non-trivial (lower and upper) bounds on the optimal number of recombinants are estimated at each branching node to prune the branch-and-bound search tree. When multiple solutions exist, a best haplotype configuration is selected based on a maximum likelihood approach.

Our test results on simulated data using three pedigree structures demonstrate that the above algorithm (which will be referred to as simply as algorithm ILP from now on) is very efficient. For example, it could recover haplotypes with 50 loci on a pedigree of size 29 in seconds on a regular PC. It outruns MRH v0.2 on biallelic data while guaranteeing the minimum number of recombinants. With respect to performance (*i.e.* accuracy in terms of correctly recovered phase information at each marker locus), ILP was able to recover correct phase information at more than 99.8% of the marker loci for data with no missing alleles and 98.3% of the marker loci for data with as many as 20% missing alleles.

As an application to real data, we have applied the algorithm ILP to a genome-scale data set that consists of 12 multi-generation human pedigrees studied in a recent paper [9]. We focus on each of the blocks inferred in [9] and compare the haplotyping results of ILP with those of the EM algorithm used in [9]. The comparison shows that ILP outputs haplotype configurations that require a very few recombinants and result in roughly the same set of *common haplotypes* (*i.e.* haplotypes that occur with a frequency at least 5%) as the EM algorithm. We also compare ILP with the algorithm block-extension on chromosome 3 consisting of 10 blocks. The results show that ILP often finds solutions that require fewer recombinants than those returned by block-extension (or MRH, which has a similar performance). Out of the solutions for 120 (= $12 \cdot 10$) data sets found by ILP, only 2 requires recombinants while 18 solutions output by block-extension require recombinants. This difference is mainly due to different methods were used to impute missing alleles.

## 1.3 Organization of the Paper

The rest of this paper is organized as follows. We introduce briefly the biological background of the MRHC problem and some relevant terms in Section 2 and the integer linear program formulation in Section 3. In Section 4, we explore some special properties in the constraints of the ILP formulation and define several useful (partial order and other) relationships among the variables. Some implementation issues and statistical assessments on multiple optimal solutions are presented in Sections 5 and 6. Section 7 shows the experimental results on simulated data sets and on a real data set. We conclude the paper with a few remarks about possible future work in Section 8.

## 2. PRELIMINARIES

The genome of an organism consists of *chromosomes* that are double strand DNAs. Locations on a chromosome can be identified using *markers*, which are small segments of DNA with some specific features. A physical position of markers on a chromosome is called a *marker locus* and a marker state is called an *allele*. A set of markers and their positions define a *genetic map* of chromosomes. There are many types of markers. The two most commonly used markers are *microsatellite* markers and *SNP (single nucleotide polymorphism)* markers. Different sets of markers have different properties, such as the total number of distinct allelic states at a locus, frequency of each allele, distance between two adjacent loci, *etc.* A microsatellite marker usually has several different alleles at a locus (called *multi-allelic*) while an SNP marker can be treated as a *biallelic*, which has two alternative states. The average distance between two SNP marker loci is much smaller than the average distance between two microsatellite marker loci, thus making SNP markers superior to other markers in gene fine-mapping. In *diploid* organisms, chromosomes come in pairs. The status of two alleles at a particular marker locus of a pair of chromosomes is called a *marker genotype*. The genotype information at a locus will be denoted using a set, *e.g.* $\{a, b\}$. If the two alleles $a$ and $b$ are the same, the genotype is *homozygous*. Otherwise, it is *heterozygous*. A *haplotype* consists of all alleles, one from each locus, that are on the same chromosome. Figure 1(A) illustrates the above concepts, where alleles are represented by their (numerical) IDs.

A pedigree can be defined formally as follows.

Definition 2.1. *A pedigree graph is a weakly connected directed acyclic graph (DAG) $G = \{V, E\}$, where $V = M \cup F$, $M$ stands for the male nodes, $F$ stands for the female nodes. The in-degree of each node is 0 (founders) or 2 (non-founders). If the in-degree of a node is 2, one edge must start from a male node (called father) and the other edge from a female node (called mother) and the node itself is a child of its parents (father and mother).*

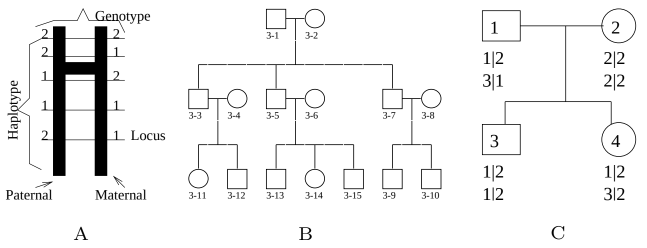


**Figure 1: A. The structure of a pair of chromosomes from a mathematical point of view. B. An illustration of a pedigree with 15 members. C. An example of recombination event.**

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

We use the term *haplotype configuration* to describes not only the paternal and maternal haplotypes of an individual, but also the (grandpaternal or grandmaternal) origin of each allele on the haplotypes. Observe that the number of recombinants required in a pedigree can be easily computed once the haplotype configuration of each member of the pedigree is given. The following problem, called MRHC in the above, is known to be NP-hard [13, 14], which trivially implies MRHC with missing alleles is also NP-hard.

Definition 2.2. *(MRHC) Given a pedigree and genotype information for each member of the pedigree, find a haplotype configuration for the pedigree that requires the minimum number of recombinants.*

## 3. AN ILP FORMULATION OF MRHC WITH MISSING ALLELES

We first introduce variables needed in the formulation. Consider an input pedigree with genotype information. Let $n$ denote the size of the pedigree, and $m$ the total number of marker loci. For a marker locus $j$, let $t_j$ denote the total number of distinct alleles that occur at locus $j$, and $M_j = \{m_1^j, m_2^j, ..., m_{t_j}^j\}$ the set of all possible alleles at locus $j$,

[1]The pedigree diagrams in this paper were generated using WPEDRAW [4].

where $m_k^j \geq 1$. For each member $i$ and locus $j$, we introduce $2t_j$ indicator (binary) variables $f_{i,k}^j$ and $m_{i,k}^j$ $1 \leq k \leq t_j$, denote the paternal allele and maternal allele of member $i$ at locus $j$, respectively. Namely, $f_{i,k}^j = 1$ ($m_{i,k}^j = 1$) if and only if the paternal (maternal) allele of member $i$ at locus $j$ is $m_k^j$. For each non-founder member $i$ and locus $j$, we introduce two indicator variables $g_{i,1}^j$ and $g_{i,2}^j$. (This information is unnecessary for the founders.) The variable $g_{i,1}^j$ indicates the grandparental origin of $i$'s paternal allele at locus $j$, *i.e.* $g_{i,1}^j = 0$ (or 1) if $i$'s paternal allele is copied from its father's paternal (or maternal, respectively) allele. The variable $g_{i,2}^j$ is defined for $i$'s maternal allele at locus $j$ in a similar way.

The haplotype (*i.e.* phase) information and the grandparental origin of each allele at non-founders are completely defined by the above $f$, $m$ and $g$ variables. Hence, we can easily formulate an integer program for MRHC with missing alleles using these variables, the genotype information, and the Mendelian law of inheritance. However, it is not obvious how we can represent the total number of recombinants in the pedigree (*i.e.* the objective function of MRHC) as a linear function of these variables. In order to make the objective function linear, we introduce a variable $r_{i,l}^j$ for each pair of "adjacent" variables $g_{i,l}^j$ and $g_{i,l}^{j+1}$ ($1 \leq j \leq m-1$ and $l = 1, 2$) to count the number of recombinants. Here, $r_{i,l}^j = 1$ if and only if $g_{i,l}^j \neq g_{i,l}^{j+1}$. The total number of recombinants can thus be described as:

$$\sum_{\text{non-founder } i} \sum_{j=1}^{m-1} (r_{i,1}^j + r_{i,2}^j) \tag{3.1}$$

## 3.1 The Constraints

For each member $i$ and locus $j$, the $f$ and $m$ variables have to satisfy the following constraints:

$$\sum_{k=1}^{t_j} f_{i,k}^j = 1, \sum_{k=1}^{t_j} m_{i,k}^j = 1 \tag{3.2}$$

Given the genotype information (denoted as $\{a, b\}$) of member $i$ at locus $j$, we have the following constraints:

$$\begin{aligned} \{m_r^j, m_s^j\} &\Rightarrow \{f_{i,r}^j + f_{i,s}^j = m_{i,r}^j + m_{i,s}^j = 1, \\ &\qquad f_{i,r}^j + m_{i,r}^j = f_{i,s}^j + m_{i,s}^j = 1\} \quad (3.3) \\ \{m_r^j, m_r^j\} &\Rightarrow \{f_{i,r}^j = 1, m_{i,r}^j = 1\} \quad (3.4) \\ \{m_r^j, 0\} &\Rightarrow \{f_{i,r}^j + m_{i,r}^j \geq 1\} \quad (3.5) \end{aligned}$$

where $m_r^j, m_s^j \in M_j, m_r^j \neq m_s^j$, and 0 stands for a missing allele. If both alleles are missing at the locus, no further constraints (other than constraint 3.2) are provided.

Following the Mendelian law of inheritance, the $f$, $m$ and $g$ variables must satisfy constraints:

$$f_{i,k}^j - f_{f,k}^j - g_{i,1}^j \leq 0 \tag{3.6}$$

$$f_{i,k}^j - m_{f,k}^j + g_{i,1}^j \leq 1 \tag{3.7}$$

where $1 \leq k \leq t_j$ and $f$ (in the subscript) denotes $i$'s father. Constraint 3.6 ensures that if $i$'s paternal allele is supposed to originate from its father's paternal allele (*i.e.* when $g_{i,1}^j = 0$), then the two alleles must be the same. In other words, constraint 3.6 implies that if $f_{i,k}^j = 1$, $f_{f,k}^j$ must be 1 when $g_{i,1}^j = 0$. Constraint 3.7 deals with the case $g_{i,1}^j = 1$ in a similar way. The constraints relating $i$ to its mother can be defined in the same way. Recall that variable $r_{i,k}^j$ is the exclusive-or of variables $g_{i,k}^j$ and $g_{i,k}^{j+1}$. The following four constraints will ensure this relationship.

$$r_{i,k}^j - g_{i,k}^j - g_{i,k}^{j+1} \leq 0 \tag{3.8}$$

$$r_{i,k}^j + g_{i,k}^j + g_{i,k}^{j+1} \leq 2 \tag{3.9}$$

$$-r_{i,k}^j + g_{i,k}^j - g_{i,k}^{j+1} \leq 0 \tag{3.10}$$

$$-r_{i,k}^j - g_{i,k}^j + g_{i,k}^{j+1} \leq 0 \tag{3.11}$$

Finally, since all the variables are binary integers, we have constraint

$$\begin{gathered} f_{i,k}^j, m_{i,k}^j, g_{i',l}^j, r_{i',l}^{j'} \in \{0,1\}, \\ 1 \leq j \leq m, 1 \leq i \leq n, 1 \leq k \leq t_j, \\ 1 \leq j' \leq m-1, \text{non}-\text{founder } i', 1 \leq l \leq 2 \end{gathered} \tag{3.12}$$

The above (binary) integer linear program defines exactly the MRHC problem with missing alleles. Observe that it implicitly contains the problem of checking Mendelian consistency of genotype data with missing alleles on a pedigree, which is known an NP-hard problem [1]. The total numbers of variables and constraints are linear in the input size. Since it uses binary representation, the actual number of variables could be quite large for multiallelic data. In the next subsection, we will try to simplify the formulation a bit.

## 3.2 A Simplified Formulation

The main idea of our simplification is to explicitly explore the dependency relationship between variables in the system and try to remove as much redundancy as possible without complicating the constraints too much. Observe that constraints 3.3 and 3.4 supersede constraint 3.2. So, we only need to keep constraint 3.2 when some alleles are missing. We can thus replace constraints 3.2 and 3.5 by:

$$\begin{aligned} \{0,0\} &\Rightarrow \{\sum_{k=1}^{t_j} f_{i,k}^j = 1, \sum_{k=1}^{t_j} m_{i,k}^j = 1\} \quad (3.13) \\ \{m_r^j, 0\} &\Rightarrow \{-f_{i,r}^j - m_{i,r}^j \leq -1, \\ &\qquad \sum_{k=1}^{t_j} f_{i,k}^j = 1, \sum_{k=1}^{t_j} m_{i,k}^j = 1\} \quad (3.14) \end{aligned}$$

Furthermore, since there are only two variables in each equality of constraint 3.3, fixing the value of any variable would determine the other three variables. We arbitrarily select one of the four variables as the *representative*, and substitute appropriately the representative for the other three variables in the system. Constraint 3.3 can then be removed. Constraint 3.4 can be removed if we replace its variables by constant values in the system. Constraints 3.6 and 3.7 will be kept only if they remain non-trivial after constant variables are replaced. Constraints 3.13 and 3.14 may also contain equality constraints with only two variables after constant variables are replaced. Those equality constraints with two variables can be treated similarly as constraint 3.3.

## 4. EXPLORING THE CONSTRAINTS

Because of the NP-hardness of MRHC, it is unlikely to find an efficient polynomial-time algorithm to solve the above ILP formulation. We adopt a widely used strategy, *branch-and-bound*, to search for an optimal solution. A comprehensive treatment of integer linear programming techniques based on branch-and-bound can be found in [28]. Basically, the branch-and-bound method solves an ILP instance by dropping the integer constraint (*i.e.* linear relaxation) to obtain a lower bound of the instance. The procedure terminates when the optimal solution (of the relaxed instance) is integral or larger than some estimated upper bound, or no feasible solutions exist. Otherwise, it branches on some selected variables and creates some sub-instances. The process iterates until all sub-instances have been considered or pruned. Clearly, different branching orders may have a large impact on the size of the search tree. In this section, we consider some special properties of the above ILP formulation and use them to guide the branching process. We will also use additional lower bounds derived from nuclear families and upper bounds derived by the block-extension heuristic to prune the search tree.

### 4.1 A Partial Order Relationship

The replacement of constant variables in inequality constraints may result in many inequality constraints with two variables, which define a partial order relationship among the involved variables as given below. For convenience, let us drop the subscript of a variable in the following and denote

$$y^\gamma = \begin{cases} y & \text{if } \gamma = 1 \\ 1-y & \text{if } \gamma = 0 \end{cases}$$

Each inequality constraint involving two variables can be expressed in the form

$$y_i^\alpha \le y_j^\beta \tag{4.1}$$

We define a directed graph $G$ on variables involved in the above inequality constraints as follows. For each variable $y^\gamma$, $G$ contains a vertex $v(y^\gamma)$. There is an edge from $v(y_i^\alpha)$ to $v(y_j^\beta)$ if inequality 4.1 holds. It is easy to see that for all the vertices in a *strongly connected component* (SCC) of $G$, their corresponding variables must have the same value in a feasible solution. By identifying and collapsing the SCCs of $G$, we can remove many variables and simplify their associated inequality constraints as we did for the two-variable equality constraints before. The SCCs can be constructed by using a standard depth-first search (DFS) [3].

Furthermore, the following rules can be used to detect inconsistency and variables with "forced" constant values. Rule 1 states that a variable and its complement cannot occur in the same SCC $S$. Rule 2 states that if a variable is smaller (or larger) than another variable and the complement of this variable, it must be 0 (or 1, respectively). Rule 3 states if a variable is smaller than its complement, it must be 0.

*Rule 1:*

$$v(y^0), v(y^1) \in S \Rightarrow \text{Inconsistency}$$

*Rule 2:*

$$y_i^\alpha \le y_j^\beta \bigwedge y_i^\alpha \le y_j^{1-\beta} \Rightarrow y_i^\alpha = 0$$
$$y_i^\alpha \le y_j^\beta \bigwedge y_i^{1-\alpha} \le y_j^\beta \Rightarrow y_j^\beta = 1$$

*Rule 3:*

$$y_i^\alpha \le y_i^{1-\alpha} \Rightarrow y_i^\alpha = 0$$

The above simplifications may result in new equality constraints with two variables and constant variables. We can repeat the steps in Section 3.2 to further reduce these variables and constraints.

Not only does the directed graph provide a way to reduce variables and constraints, the partial order obtained on the variables after shrinking each SCC can also guide the selection of branching variables. Observe that, if $y_i^\alpha \le y_j^\beta$, then $y_i^\alpha = 1$ implies $y_j^\beta = 1$. So, we perform a topological sort on the shrunk graph (which is a directed acyclic graph, or DAG). For each vertex $v$, we define the weight of $v$ as the number of successors of $v$ in the topological sort. When branching, we consider the variables in the topological sorted order and always take the 1-branch first for each variable.

### 4.2 Equality Constraints with Three or More Variables

The procedure in Section 3.2 removes all equality constraints from constraints 3.3, 3.13 and 3.14 with two variables. This leaves some equality constraints with more than two variables as given in constraints 3.13 and 3.14. These equality constraints could be modified in the above simplification process (*i.e.* some variables can be replaced and constants substituted in), but we may assume without loss of generality that each of them still has the form of $\sum_{k=q}^{p} f_{i,k}^j = 1$ (or $\sum_{k=q}^{p} m_{i,k}^j = 1$) because the variables are binary integers. [2] Observe that, in such an equality constraint, fixing a variable to one would make all other variables zero. So, we say that the variables in each equality constraint form an *exclusion* set (for lack of better terms). We can take advantage of this information when selecting branching variables by considering the variables from an exclusion set consecutively and taking the 1-branch first for each variable. We define the weight of a variable in such an exclusion set as the size of the set minus one.

### 4.3 Lower Bounds from Nuclear Families

When dealing with data sets that require a large number of recombinants (*e.g.* data in linkage analysis involving markers separated by large genetic distances), the linear relaxation usually does not give a tight lower bound on the number of minimum of recombinants. Observe that an effective lower bound must involve variables from more than 1 locus and MRHC is already NP-hard for data with 2 loci. Hence, we have to work with substructures of the input pedigree. A natural substructure in a pedigree is a nuclear family. A nuclear family constitutes a small instance of MRHC and can usually be solved in a much shorter time. The solution for a nuclear family gives a valid lower bound (*i.e.* or cut) concerning the $r$ variables from the family and the sum of these lower bounds forms a lower bound for the whole pedigree. The lower bound/cut can be computed in advance for the root node of the branch-and-bound search tree and

[2]If it is necessary, we could introduce some complementary variables to replace expressions of the form $1-x$ in an equality constraint. If a variable and its complement both appear in (different) equality constraints, we could derive new equalities without such a variable or its complement by summing up appropriate equalities.

updated at each branching node, but the latter might incur big time overhead. A sensible strategy is to keep track of the difference between the current upper bound and lower bound, and update the lower bound only when the difference is larger than a predefined threshold.

### 4.4 An Upper Bound

When the input data require a small number of recombinants (*e.g.* in the case of SNP data), a tight upper bound could be more effective than the above lower bound because many nuclear families could be realized with 0 recombinants. In this case, the block-extension algorithm [13, 14] can be used to estimate an upper bound because it is very efficient and accurate when the number of recombinants is small.

## 5. IMPLEMENTATION ISSUES

In our implementation of the ILP algorithm, we solve the linear relaxations of the ILP instances by using the IBM Optimization and Solution Library (OSL). A complete pseudocode of the branch-and-bound algorithm that summarizes the discussions in sections 3 and 4 is given in Figure 2.

```
Algorithm ILP
Input: Genotype data of a pedigree with possible missing alleles
Output: Haplotypes for all the members in the pedigree
Data structure: Constant variable set $C$, the set of representatives ($R$) of some variables, constraints $S$, global upper bound $gub$ and lower bound $glb$, exclusion sets $I$, partial order relationship $O$, partial order graph $G$ and topological sorted order $L$, instance list $P$, current instance $p$, and local lower bound $lb$ by linear relaxation.

1. //Init:
2. Collect $C$, $R$ and $S$, and calculate $gub$ by BE and $glb$ from nuclear families.

3. //Preprocessing
4. Iterate until no new updates exist:
      - Update $R$ and $S$ by removing constant variables;
      - Update $O$ and build $G$;
      - Find the SCCs of $G$ and update $R$;
      - Detect variables forced into constants and put them in $C$;
5. Prepare $I$ and $L$, set the current instance as $p$, run steps 6–10 until $P$ is empty.

6. //Branch-and-bound
7. Solve $p$ by linear relaxation to obtain a local lower bound $lb$.
      - If $p$ is infeasible or $lb \geq gub$, mark it as processed;
      - If the solution is integral, update $gub$;
      - Otherwise, continue to branch.

8. //Branching and selection
9. If $I \neq \phi$ or $L \neq \phi$, select a variable with the largest weight to
   branch, select the instance that results from the 1-branch as $p$,
   and put the outcome of the 0-branch into $P$;
10. Otherwise select a variable from the objective function, select
   the outcome of the 0-branch as $p$, and put the outcome of the
   1-branch into $P$.

11. Output the optimal solutions or report that the instance is
   infeasible.
```

**Figure 2: A pseudo-code of the ILP algorithm.**

## 6. STATISTICAL ASSESSMENT OF MULTIPLE SOLUTIONS

Our algorithm in fact finds all solutions with the minimum number of recombinants. We can further choose the "best" one from these optimal solutions using a maximum likelihood approach if the genetic distances between markers are given. An alternative treatment is to output all solutions together with their associated probabilities. Both approaches require the calculation of the likelihood of a haplotype solution given the genotype information. Because the number of optimal haplotype solutions is usually very small, this calculation is much easier than calculating the maximum likelihoods for all feasible solutions [16]. Let $H$ denote a haplotype solution and $G$ the input genotypes. Let $f(i)$ and $m(i)$ denote the father and mother of an individual $i$, and $h_i$ and $g_i$ the haplotypes and genotype of $i$. The likelihood of the haplotype configuration $H$ given the genotypes $G$ is

$$P(H|G) = \prod_{\text{founder } i} P(h_i|g_i) \prod_{\text{non-founder } i} P(h_i|h_{f(i)}, h_{m(i)}), \tag{6.1}$$

where the term $P(h_i|g_i)$ can be obtained under Hardy-Weinberg equilibrium assumption, if prior knowledge about haplotype frequencies is known. The term $P(h_i|h_{f(i)}, h_{m(i)})$ (transmission probability) can be calculated under the assumption that recombination events are independent (no interference) and uniformly distributed.

If our input is actually a population of pedigrees (as in the case of [9]), we can further estimate the population haplotype frequencies and the probability of observing the genotypes in each pedigree given the estimated haplotype frequencies by an expectation-maximization (EM) algorithm, under the assumption that all the founders are independent. The EM algorithm works by summing over all possible optimal solutions for each pedigree, weighted according to their relative likelihoods. More specifically, an arbitrary optimal solution is selected from each pedigree and the haplotype frequencies are estimated according to Equation 6.2. Based on the estimated frequencies, a haplotype solution with the maximum probability (Equation 6.3) is then chosen for each pedigree, which will serve the input of the next iteration. Let $h_i^1$ and $h_i^2$ denote the two haplotypes of a founder $i$ for any given optimal solution, and $N$ the total number of founders. Let $\pi(h_i^1)$ denote the number of haplotypes $h_i^1$ occurring in all founders. Then the expected frequency of haplotype $h_i^1$ is simply

$$\hat{f}(h_i^1) = \frac{\pi(h_i^1)}{2N}. \tag{6.2}$$

The probability of observing genotypes $G$ and a haplotype solution $H$ in a pedigree is

$$P(G, H|\hat{f}) = \prod_{\text{founder } i} \hat{f}(h_i^1)\hat{f}(h_i^2) \prod_{\text{non-founder } i} P(h_i|h_{f(i)}, h_{m(i)}). \tag{6.3}$$

## 7. PRELIMINARY EXPERIMENTAL RESULTS

We have implemented the above algorithm ILP as a module of our *PedPhase* program, which is available at website `http://www.cs.ucr.edu/~jili/haplotyping.html`. To evaluate the efficiency of ILP, we first compared ILP, block-extension (BE, also in PedPhase) and MRH v0.2 [21] on simulated genotype data in terms of efficiency on three different pedigree structures. The results show that, as an exact solution, our ILP is in fact faster than MRH v0.2 on SNP data. We also evaluated how the number of marker loci, the size of a pedigree, the number of recombinants, and the amount of missing alleles affect the efficiency. An advantage of using simulated data here is that we know the true haplotype configurations and the number of recombinants. Hence, we can evaluate the accuracy of ILP in terms of the percentage of markers with haplotypes correctly inferred. As an application to real data, we tested ILP on a genome-scale data set that consists of 12 multi-generation human pedigrees studied in a recent paper [9]. We focused on each of the inferred blocks and compare the results of ILP with those of the EM algorithm used in [9]. The comparison shows that ILP outputs haplotype configurations that require a very few recombinants and result in roughly the same set of common haplotypes as the EM algorithm. We also compared ILP with block-extension on chromosome 3 consisting of 10 blocks. The results show that ILP often finds solutions that require fewer recombinants than those returned by block-extension. The details of the test results are given in the following subsections.

## 7.1 Efficiency and Accuracy of ILP on Simulated Data

We used a method similar to those in [6, 13, 14] to generate simulated data sets. Three different pedigree structures were considered. One is a small pedigree with 15 members as shown in Figure 1. The second is a medium-sized pedigree with 29 members as shown in Figure 3 (left) and the third is a pedigree of 17 members with a mating loop as shown in Figure 3 (right). Both multi-allelic (with 6 alleles per locus) and biallelic data were considered. The alleles were generated following a uniform frequency distribution in order to maximize the chance of heterozygosity (to test the worst-case performance of the algorithms). Three different numbers of loci, namely 10, 25 and 50 were considered. The number of recombinants used in generating each pedigree ranged from 0 to 4. In addition, we considered the rate of missing alleles as 5%, 10%, 15%, and 20%. For each data set, 100 copies of random genotype data were generated. The total number of data sets used is 45000 ($= 3 \cdot 2 \cdot 3 \cdot 5 \cdot 5 \cdot 100$).

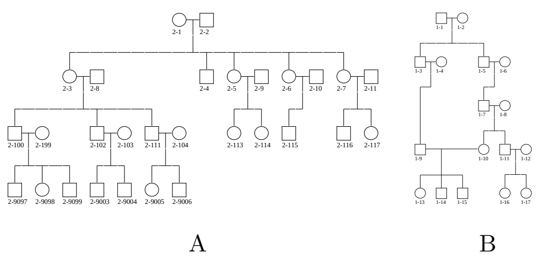


**Figure 3: A. A pedigree with 29 members. B. A pedigree with 17 members and a mating loop.**

The test results demonstrate that ILP is slower than block-extension, but faster than MRH v0.2 on biallelic data (although it is a little bit slower than MRH v0.2 on multi-allelic data), as described in Table 1. In the table, the first column indicates the combination of parameters: the size of the pedigree, the number of loci in each member, the number of distinct alleles allowed at each locus, and the number of recombinants used to generate the genotype data, respec-

**Table 1: Speeds of BE, MRH and ILP on multi-allelic and biallelic markers.**

| Parameters | BE | MRH | ILP |
|---|---|---|---|
| (17,10,6,0) | 2.1s | 7s | 34s |
| (17,10,6,4) | 2.1s | 11s | 37s |
| (15,25,6,0) | 2.7s | 18s | 2m34s |
| (15,25,6,4) | 2.9s | 33s | 3m9s |
| (29,10,6,0) | 3.2s | 10s | 1m49s |
| (29,10,6,4) | 3.1s | 15s | 1m57s |
| (29,25,6,0) | 15s | 4m | 15m2s |
| (29,25,6,4) | 10s | 2m6s | 15m10s |
| (17,10,2,0) | 1.9s | 15s | 20s |
| (17,10,2,4) | 2.3s | 1m11s | 23s |
| (15,25,2,0) | 4.7s | 10m50s | 1m6s |
| (15,25,2,4) | 4.8s | 13m49s | 1m18s |
| (29,10,2,0) | 2.8s | 6m26s | 44s |
| (29,10,2,4) | 2.7s | 3m46s | 50s |
| (29,25,2,0) | 2.3s | 2h7m | 3m41s |
| (29,50,2,0) | 16s | 45h | 15m21s |

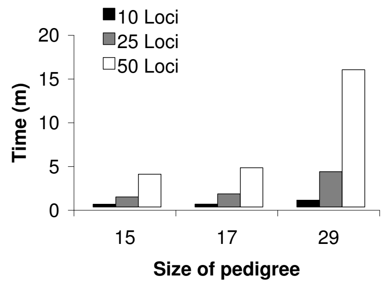
tively. The time used by each program in this section is the total time for 100 random runs for each parameter combination on a Pentium IV with 1.7GHz CPU and 516MB RAM.

Figure 4(A) uses a bar diagram to show how the speed of ILP is affected by the input size (*i.e.* number of marker loci and size of a pedigree) on biallelic data, when the number of recombinant is fixed as 1. For example, ILP requires a total of 20 minutes for 100 runs on a pedigree with 29 members and 50 loci. This suggests that ILP is efficient for MRHC instances of practical sizes. To consider the effect of the number of recombinants and the rate of missing data, Figure 4(B) indicates that the running time will increase with these two parameters, mainly because the number of free variables in the system will increase. But unlike parameters such as the pedigree size and the number of loci, for which the time increases exponentially as expected, the growth rates in Figure 4(B) are more like linear functions. In this figure, the number of marker loci is 50 and the size of the pedigree is 29.

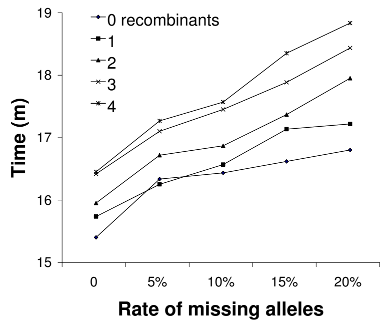
Not only does ILP solve MRHC with missing data optimally, the results in Figure 4(C) demonstrate that the algorithm is very good at recovering true haplotypes. Its overall accuracy is better than 98% in terms of the number of missing alleles correctly recovered and the number of loci with haplotypes correctly inferred, for missing rate as many as 20%. Its accuracy on data with no missing alleles is even better; More than 99.8% of the loci were correctly phased. The few errors were mainly due to existence of multiple optimal solutions. These results also show that, when the number of recombinants are few and the recombination events are randomly distributed, the true haplotype configuration is often a minimum recombinant haplotype configuration. In contrast, similar simulations in [13, 14] show that the performance of block-extension could fall below 80% on pedigrees with mating loops or data that require a moderately large number (*i.e.* $\geq 4$) of recombinants.

## 7.2 A Genome-Scale Haplotype Reconstruction

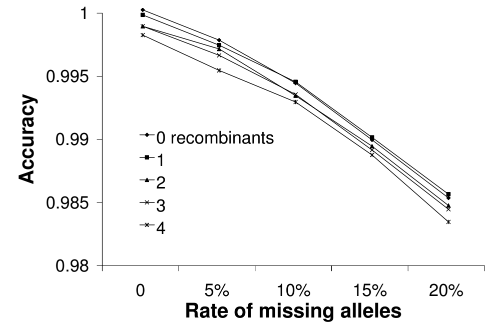
We have also tested ILP on a real data set from Whitehead/MIT Center for Genome Research. Gabriel *et al.* [9] recently reported results on a genome-scale SNP haplotype block partition and haplotype frequency estimation project. Their original data set consists of 4 populations and 54 autosomal regions, each with an average size of 250K bps,



A



B



C

**Figure 4: Some simulation results on ILP. A. Effect of problem size on speed. B. Effect of number of recombinants and rate of missing alleles on speed. C. Effect of number of recombinants and rate of missing alleles on accuracy.**

spanning 13.4M bps (about 0.4%) of the human genome. Haplotype blocks were defined using the normalized linkage disequilibrium parameter $D'$. Blocks with fewer than four markers are omitted from further consideration. Within each block, haplotypes and their frequencies were calculated via an EM algorithm from [8]. One of the populations (European) has pedigree information and was used in our study. There are totally 93 members in the European population, separated into 12 multi-generation pedigrees (each with 7-8 members). The genotyped regions are distributed among all the 22 autosomes and each autosome contains 1 to 10 regions. The overall allele missing rate in a block is between 0.8% and 14.5%. We downloaded the SNP genotype data and pedigree structures from Whitehead/MIT Center for Genome Research website (`http://www-genome.wi.mit.edu/mpg/hapmap/hapstruc.html`), and obtained the results of the EM algorithm concerning common haplotypes and their frequencies in the population from the authors of [9]. [3] ILP was able to reconstruct the haplotypes for all regions and pedigrees accurately, by taking advantage of the available haplotype block structure. A comparison of the EM algorithm and ILP in terms of the common haplotypes that they output is given in Table 2. The first column of the table is the chromosome number and the second column is the number of blocks with more than four markers (no blocks with length larger than four in chromosome 19). Columns 3 and 4 are the average numbers of common haplotypes per block found by EM and ILP, respectively. Column 5 is the average number of different common haplotypes output by the two methods, which is usually about 10% of the common haplotypes output by each method. Column 6 is the average number of recombinants in each pedigree and each block as found by ILP, which is close to zero. A detailed description of the haplotypes found by ILP for the data set will be available at website `http://www.cs.ucr.edu/~jili/haplotyping.html`.

**Table 2: Comparison of the EM and ILP algorithms on a human genome SNP data.**

| Chr | # of blocks | EM | ILP | Mismatch | Recombs(ILP) |
|---|---|---|---|---|---|
| 1 | 22 | 3.82 | 4.00 | 0.45 | 0.034 |
| 2 | 6 | 3.33 | 4.00 | 0.67 | 0.000 |
| 3 | 10 | 3.9 | 4.00 | 0.50 | 0.033 |
| 4 | 7 | 3.57 | 3.29 | 0.14 | 0.048 |
| 5 | 7 | 3.86 | 4.12 | 0.43 | 0.024 |
| 6 | 11 | 3.55 | 3.54 | 0.67 | 0.008 |
| 7 | 9 | 2.67 | 3.33 | 0.22 | 0.037 |
| 8 | 8 | 3.63 | 3.38 | 0.25 | 0.000 |
| 9 | 3 | 3.67 | 4.33 | 1.33 | 0.333 |
| 10 | 7 | 4.14 | 3.57 | 0.71 | 0.095 |
| 11 | 5 | 3.40 | 3.60 | 0.40 | 0.083 |
| 12 | 6 | 3.00 | 2.83 | 0.17 | 0.00 |
| 13 | 6 | 3.67 | 3.83 | 0.50 | 0.042 |
| 14 | 4 | 3.50 | 3.50 | 0.00 | 0.000 |
| 15 | 3 | 3.33 | 4.33 | 1.00 | 0.028 |
| 16 | 4 | 3.50 | 3.75 | 0.25 | 0.125 |
| 17 | 2 | 2.5 | 2.00 | 0.50 | 0.000 |
| 18 | 4 | 3.25 | 3.25 | 0.25 | 0.125 |
| 20 | 2 | 4.00 | 4.00 | 0.00 | 0.000 |
| 21 | 1 | 2.00 | 3.00 | 1.00 | 0.167 |
| 22 | 8 | 4.12 | 3.88 | 0.50 | 0.021 |

We further compared the results of ILP with our previous experiment on the block-extension and EM algorithms

[3]Here, the pedigree information was used first to resolve the phases of some heterozygous loci using the Mendelian law of inheritance before the EM algorithm was run on founders.

**Table 3: The regions and blocks on chromosome 3.**

| Name | Length | SNPs | Blocks | SNPs/block | Missing rate |
|---|---|---|---|---|---|
| 16a | 40 | 14 | 1 | 5 | 7.96% |
| 16b | 106 | 53 | 1 | 6 | 3.76% |
| | | | 2 | 4 | 2.69% |
| 17a | 186 | 70 | 1 | 6 | 4.70% |
| | | | 2 | 5 | 1.50% |
| | | | 3 | 4 | 7.80% |
| | | | 4 | 6 | 6.27% |
| 18a | 286 | 74 | 1 | 16 | 3.70% |
| | | | 2 | 6 | 5.73% |
| | | | 3 | 4 | 2.15% |

**Table 4: Common haplotypes and their frequencies obtained by block-extension, ILP and the EM method. In haplotypes, the alleles are encoded as 1=A, 2=C, 3=G, and 4=T.**

| Block | Common haplotypes | EM | BE | ILP |
|---|---|---|---|---|
| 16a-1 | 4 2 2 2 2 | 0.4232 | 0.3817 | 0.3750 |
| | 3 4 3 4 4 | 0.2187 | 0.1720 | 0.2187 |
| | 4 2 2 2 4 | 0.2018 | 0.1935 | 0.1979 |
| | 3 4 2 2 4 | 0.1432 | 0.1613 | 0.1458 |
| sum | | 0.9869 | 0.9085 | 0.9374 |
| 16b-1 | 3 2 4 1 1 2 | 0.8014 | 0.7634 | 0.7813 |
| | 1 3 2 3 3 4 | 0.0833 | 0.0753 | 0.0833 |
| sum | | 0.8847 | 0.8387 | 0.8646 |
| 16b-2 | 4 1 2 2 | 0.5410 | 0.4892 | 0.5104 |
| | 2 3 3 4 | 0.2812 | 0.2581 | 0.2500 |
| | 2 3 3 2 | 0.1562 | 0.1344 | 0.1562 |
| sum | | 0.9784 | 0.8788 | 0.9166 |
| 17a-1 | 3 1 3 4 4 4 | 0.3403 | 0.3172 | 0.2917 |
| | 1 3 3 2 4 2 | 0.3021 | 0.2419 | 0.2500 |
| | 3 3 2 4 2 4 | 0.1354 | 0.0914 | 0.0938 |
| | 3 3 3 4 4 4 | 0.1021 | 0.1183 | 0.1354 |
| | 3 3 2 4 4 4 | 0.0681 | 0.0806 | 0.0729 |
| | 1 3 3 2 4 4 | 0.0521 | | |
| sum | | 1.0000 | 0.8494 | 0.8438 |
| 17a-2 | 2 3 2 4 2 | 0.3542 | 0.2903 | 0.3229 |
| | 3 3 4 2 4 | 0.3333 | 0.2957 | 0.3125 |
| | 3 3 4 4 2 | 0.1458 | 0.1344 | 0.1563 |
| | 3 4 4 4 4 | 0.1250 | 0.1452 | 0.1250 |
| sum | | 0.9583 | 0.8656 | 0.9167 |
| 17a-3 | 4 4 3 1 | 0.4129 | 0.4355 | 0.4167 |
| | 3 1 1 2 | 0.2813 | 0.2258 | 0.2292 |
| | 4 1 3 1 | 0.2363 | 0.1935 | 0.2188 |
| | 4 1 3 2 | 0.0696 | 0.0753 | 0.0729 |
| sum | | 1.0000 | 0.9301 | 0.9376 |
| 17a-4 | 3 4 4 1 2 4 | 0.3854 | 0.3710 | 0.3436 |
| | 2 3 2 4 3 2 | 0.3333 | 0.2903 | 0.3021 |
| | 3 4 2 4 2 4 | 0.2500 | 0.1881 | 0.2188 |
| sum | | 0.9687 | 0.8494 | 0.8645 |
| 18a-1 | 1444231214144132 | 0.2697 | 0.2473 | 0.2396 |
| | 1444111214144132 | 0.2396 | 0.2151 | 0.2083 |
| | 1444131214144132 | 0.1887 | 0.2204 | 0.1979 |
| | 4222133313412211 | 0.1250 | | |
| | 1444231234144132 | 0.0833 | 0.0699 | 0.0729 |
| | 4444133214144132 | | | 0.0521 |
| sum | | 0.9063 | 0.7527 | 0.7708 |
| 18a-2 | 3 1 2 4 4 2 | 0.4967 | 0.4892 | 0.4271 |
| | 1 3 2 4 3 4 | 0.2604 | 0.1935 | 0.1667 |
| | 3 1 2 2 4 2 | 0.1271 | 0.0753 | 0.0938 |
| | 1 3 4 4 4 4 | 0.0938 | 0.0806 | 0.0729 |
| | 1 3 2 4 3 2 | | 0.0538 | 0.0625 |
| | 3 1 2 4 4 4 | | | 0.0521 |
| sum | | 0.9780 | 0.8924 | 0.8751 |
| 18a-3 | 2 2 1 1 | 0.4186 | 0.4032 | 0.3854 |
| | 4 3 3 3 | 0.2188 | 0.1935 | 0.2188 |
| | 2 3 1 1 | 0.2064 | 0.2204 | 0.2396 |
| | 4 3 1 3 | 0.1250 | 0.1559 | 0.1146 |
| sum | | 0.9688 | 0.9730 | 0.9584 |

on chromosome 3 reported in [13]. (The results of the EM algorithm was obtained from the authors of [9].) There are 4 regions in the chromosome 3 data and each region is partitioned into 1 to 4 blocks. The region name, physical length (kbps), number of blocks, SNPs in each block and percentage of missing alleles of each block are summarized in Table 3. In the experiment, the algorithm block-extension imputed missing alleles by sampling the alleles according to their estimated frequencies. On the other hand, ILP inferred missing alleles so that the number of required recombinants is minimized in the final haplotype solution. In other words, both missing data imputation and the reconstruction of haplotype configurations were combined in one framework. After haplotypes were inferred for the members of all pedigrees, population haplotype frequencies were estimated by counting the founders' haplotypes. Such frequency information can be used to estimate the likelihood of the haplotypes in a pedigree as described in Section 6. The common haplotypes, their frequencies and their total frequency in each block of chromosome 3 estimated by block-extension, ILP and the EM algorithm are summarized in Table 4. The majority of the common haplotypes identified by the three algorithms are the same. The small number of differences mainly concern haplotypes with frequencies close to 5%. Similar patterns were also observed in the tests on other chromosomes. Furthermore, for common haplotypes shared by the three algorithms, all three algorithms gave frequencies close to each other, although the frequencies given by ILP and block-extension are in general smaller than those found by the EM algorithm. This is perhaps mainly due to different strategies used for imputing missing alleles and the fact that the EM algorithm only used the founders of the pedigrees in its computation.

We have also looked at the number of recombinations required in the solutions found by ILP. Out of the 120 data sets derived from the 10 blocks in chromosome 3 and the 12 pedigrees, only 2 data sets had solutions that require recombinants. In contrast, 18 data sets had recombinants in the solutions found by block-extension [13]. The difference could be due to different methods for missing data imputation and the fact that block-extension is a heuristic algorithm.

## 8. CONCLUDING REMARKS

Our simulation results show that the minimum recombination principle is valid when the number of recombinants is small and the recombination events are randomly distributed, because most of the minimum recombinant solutions correctly recovered the missing alleles and the phase information. The results on the real data set show that in terms of estimating common haplotypes and their frequencies, the solutions from ILP and the EM algorithm in general agree with each other. In order to assess the accuracies of rule-based algorithms and statistical algorithms and their efficiency systematically on data with and without pedigrees, more sophisticated models to generate data are needed.

It is possible to extend the ILP formulation by incorporating coefficients that represent the likelihoods of recombination (such as genetic distances between markers) into the objective function.

## 9. ACKNOWLEDGEMENT

We thank the two anonymous referees for many helpful suggestions, especially on Section 6. We are grateful to Drs. David Altshuler, Mark Daly, Stacey Gabriel, Stephen Schaffner, and their entire group at Whitehead/MIT Center for Genome Research for sharing their haplotype block and frequency results analyzed in [9] with us and for answering many of our inquiries.

## 10. REFERENCES

[1] L. Aceto, J. A. Hansen, A. Ingólfsdóttir, J. Johnsen, and J. Knudsen. The complexity of checking consistency of pedigree information and related problems. *Manuscript*, 2003.

[2] P. Bonizzoni, G. Della Vedova, R. Dondi, and J. Li. The haplotyping problem: an overview of computational models and solutions. *J Comp Sci Tech*, 18(6):675-688, 2003.

[3] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, Massachusetts, 2001.

[4] D. Curtis. A program to draw pedigrees using LINKAGE or LINKSYS data files. *Ann Hum Genet*, 54:365–367, 1990.

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

[6] K. Doi, J. Li, and T. Jiang. Minimum recombinant haplotype configuration on tree pedigrees. *In Proc. WABI'03*, pages 339-353, 2003.

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

[8] L. Excoffier and M. Slatkin. Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. *Mol Biol Evol*, 12:921–927, 1995.

[9] S. B. Gabriel *et al.* The structure of haplotype blocks in the human genome. *Science*, 296(5576):2225–9, 2002.

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

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

[12] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[13] J. Li and T. Jiang. Efficient inference of haplotypes from genotypes on a pedigree. *J Bioinfo Comp Biol*, 1(1):41–69, 2003.

[14] J. Li and T. Jiang. Efficient rule-based haplotyping algorithms for pedigree data. *In Proc. RECOMB'03*, pages 197–206, 2003.

[15] L. Li, J. H. Kim, and M. S. Waterman. Haplotype reconstruction from SNP alignment. *In Proc. RECOMB'03*, pages 207–216, 2003.

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

[17] R. Lippert, R. Schwartz, G. Lancia, and S. Istrail. Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Briefings in Bioinformatics*, 3(1):23–31, 2002.

[18] T. Niu, Z. S. Qin, X. Xu, and J. S. Liu. Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms. *Am J Hum Genet*, 70(1):157–169, 2002.

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

[20] I. Pe'er and J. S. Beckmann. Resolution of haplotypes and haplotype frequencies from SNP genotypes of pooled samples. *In Proc. RECOMB'03*, pages 237–246, 2003.

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

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

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

[24] M. Stephens, N. J. Smith, and P. Donnelly. A new statistical method for haplotype reconstruction from population data. *Am J Hum Genet*, 68(4):978–89, 2001.

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

[26] J. C. Venter *et al.* The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001.

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

[28] L. Wolsey. *Integer programming.* John Wiley & Sons Inc, 1998.

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