Efficient Inference of Haplotypes from Genotypes on a Pedigree

Jing Li *   Tao Jiang †

Abstract

We study *haplotype reconstruction* under the *Mendelian law* of inheritance and the *minimum recombination principle* on *pedigree data*. We prove that the problem of finding a *minimum-recombinant haplotype configuration (MRHC)* is in general NP-hard. This is the first complexity result concerning the problem to our knowledge. An iterative algorithm based on blocks of consecutive resolved marker loci (called *block-extension*) is proposed. It is very efficient and can be used for large pedigrees with a large number of markers, especially for those data sets requiring few recombinants (or recombination events). A polynomial-time exact algorithm for haplotype reconstruction without recombinants is also presented. This algorithm first identifies all the necessary constraints based on the Mendelian law and the zero recombinant assumption, and represents them using a system of linear equations over the cyclic group Z_2. By using a simple method based on Gaussian elimination, we could obtain all possible feasible haplotype configurations. A C++ implementation of the block-extension algorithm, called *PedPhase*, has been tested on both simulated data and real data. The results show that the program performs very well on both types of data and will be useful for large scale haplotype inference projects.

Keywords: Haplotyping, pedigree analysis, recombination, SNP, algorithm, computational complexity, Gaussian elimination over Z_2

1 Introduction

Genetic fine-mapping for complex diseases (such as cancers, diabetes, osteoporoses *etc.*) is currently a great challenge for geneticists and will continue to be so in the near future. With the availability of *single nucleotide polymorphisms (SNPs)* information, researchers see new potentials of genetic mapping. The ongoing genetic variation and haplotype map projects at the National Human Genome Research Institute (NHGRI) of the USA are focused on the discovery and typing of SNPs and development of high-resolution

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

† Department of Computer Science, University of California, Riverside, CA, and Shanghai Center for Bioinformatics Technology. jiang@cs.ucr.edu. Research supported by NSF Grants CCR-9988353, ITR-0085910, DBI-0133265, and National Key Project for Basic Research (973).

maps of genetic variation and haplotypes for human [10]. While a dense SNP haplotype map is being built, various new methods [11, 14, 23, 28] have been proposed to use haplotype information in linkage disequilibrium mapping. Some existing statistical methods for genetic linkage analysis have also shown increased power by incorporating SNP haplotype information [22, 30]. But, the use of haplotype maps has been limited due to the fact that the human genome is a *diploid* and, in practice, *genotype* data instead of *haplotype* data are collected directly, especially in large scale sequencing projects, because of cost considerations. Although recently developed experimental techniques [6] give the hope of deriving haplotype information directly with affordable costs, efficient and accurate computational methods for haplotype reconstruction from genotype data are still highly demanded.

The existing computational methods for haplotyping fit into two categories: statistical methods and rule-based methods. Both methodologies can be applied to *pedigree* data and *population* data (that has no pedigree information). Statistical approaches (*e.g.* [7, 12, 15, 25, 27]), such as the EM methods, estimate haplotype frequencies in addition to the haplotype configuration for each individual, but the algorithms 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 normally do not provide any numerical assessment of the reliability of their results. [1] By utilizing some reasonable biological assumptions, such as the *minimum recombination principle*, rule-based methods have proven to be powerful and practical [9, 13, 16, 21, 26, 29]. The minimum recombination principle basically says that genetic recombination is rare and thus haplotypes with fewer recombinants should be preferred in a haplotype reconstruction [9, 16, 21]. [2] The principle is well supported by practical data. For example, recently published experimental results [4, 8, 10] showed that, in the case of human, the number of distinct haplotypes is very limited. Moreover, the genomic DNA can be partitioned into long blocks such that recombination within each block is rare or even nonexistent.

We are interested in rule-based haplotype reconstruction methods on pedigrees. In a very recent paper [21], Qian and Beckmann proposed a rule-based algorithm to reconstruct haplotype configurations for pedigree data, based on the minimum recombination principle. (From now on, we call their algorithm MRH.) Given a pedigree and the genotype information about each member of the pedigree (with possibly missing data), the authors are interested in finding the haplotype configurations for each member such that the total number of recombinants (or recombination events) in the whole pedigree is minimized. We call the problem the *Minimum-Recombinant Haplotype Configuration (MRHC) problem*. Although the algorithm MRH in [21] performs very well for small pedigrees, its effectiveness scales very poorly because it runs extremely slowly on data of even moderate sizes, especially for data with biallelic markers. This is regrettable since large SNP data sets on pedigrees are becoming increasingly interesting and SNP markers are biallelic.

In this paper, we first show that the MRHC problem is in general NP-hard and then devise an efficient

[1] One can make an analogy between this relationship between statistical methods and rule-based methods and the relationship between the maximum likelihood methods and parsimony/distance methods in phylogenetic reconstruction.

[2] This is similar to the parsimony principle in phylogenetic reconstruction.

iterative (heuristic) algorithm (called *block-extension*) for MRHC. Like the existing rule-based haplotyping algorithms, our algorithm first attempts to resolve all unambiguous loci using the Mendelian law of inheritance. But instead of working on individual unresolved loci separately after the first step, as done in the algorithm MRH [21], we use some sensible greedy strategy (such as avoiding double recombinants within a small region of loci) to resolve loci that are adjacent to the previously resolved loci, resulting in *blocks* of consecutive resolved loci. Our algorithm then uses the longest block in the pedigree to resolve more unresolved loci under the minimum recombination principle. This may extend some blocks into longer blocks. The process is repeated until no blocks can be extended. The algorithm then fills the remaining gaps between blocks in each member by considering the haplotype information about the other members of the same nuclear family. The time complexity of the above algorithm is $O(dmn)$, where n is the size of the pedigree, m the number of loci, and d the largest number of children in a nuclear family. Observe that MRH runs in $O(2^d m^3 n^2)$ time [21]. Our preliminary experimental results demonstrate that the algorithm is much more efficient than the algorithm MRH because loci can be resolved much more quickly when they are considered together as blocks than when they are considered separately.

We also consider the special case of haplotype reconstruction where no recombinants are assumed. This special case is interesting not only because its solution may be useful for solving the general MRHC problem as a subroutine but also because the frequency of recombinants is expected to be close to 0 when a small (or moderately large) region of genomic DNA is considered [9, 16]. We present an algorithm to identify all 0-recombinant haplotype configurations consistent with the input data. The running time of the algorithm is polynomial in the input size and the number of consistent 0-recombinant haplotypes. Previously, only an exponential-time algorithm based on exhaustive enumeration was known [16]. Our algorithm first identifies all necessary (and sufficient) constraints on the haplotype configurations derived from the Mendelian law and the zero recombinant assumption, represented as a system of linear equations on binary variables over the cyclic group Z_2 (*i.e.* integer addition $\bmod 2$), and then solves the equations to obtain all consistent haplotype configurations satisfying the constraints, using a simple method based on Gaussian elimination. These consistent haplotype configurations are shown to be feasible 0-recombinant solutions. [3]

A C++ implementation of the block-extension algorithm, called PedPhase, has been tested on both simulated data and real data. The results on simulated data using three pedigree structures demonstrate that the block-extension algorithm runs very fast. For example, for 100 runs on a pedigree with 29 members and 50 marker loci, it uses less than 1 minute for both multi-allelic and biallelic data on a Pentium PC. In contrast, the program MRH (version 0.1) of Qian and Beckmann [21] requires 3 to 4 hours on the same data sets with multi-allelic genotype information. On the data sets with biallelic genotypes, we observed that MRH would need more than 20 hours for each run. In fact, the authors have recently told us that [20] MRH

[3] The zero recombinant assumption has been used in studies on both population data ([9]) and pedigree data (this paper). However, the assumption concerning a population data requires that there have been no recombination events ever since the single ancestral haplotype generating the population. Here, we only require that recombination events did not occur among the generations in a single pedigree. Since a pedigree typically spans a much smaller time period than the evolution of an entire population in the field of biology, the zero recombinant assumption concerning a pedigree is perhaps more realistic.

in general cannot handle data of such a (moderately large) size, especially when the genotypes are biallelic. In terms of performance (*i.e.* accuracy of the reconstructed haplotype configuration), MRH generally gives better results than PedPhase whenever it was able to handle the input data, because of its exhaustive search on individual locus. However, in most cases, the results of PedPhase were comparable. For multi-allelic genotypes, PedPhase was able to recover correct haplotype configurations in more than 90% of the cases. In particular, for data requiring zero recombinants, PedPhase could recover the correct solutions in almost all cases. PedPhase also performed very well on the moderately large biallelic data sets involving small numbers of recombinants that MRH could not handle. We further compared the performance of PedPhase, MRH and an EM algorithm on a real dataset that consists of 12 multi-generation pedigrees from [8]. We focused on a randomly selected human chromosome consisting of 10 blocks that span 618k base pairs (bps). The results show that both rule-based haplotyping methods (PedPhase and MRH) were able to discover almost all *common* haplotypes (frequency $> 5\%$) that were inferred by the EM algorithm in [8]. The haplotype frequencies estimated from the haplotype results of PedPhase and MRH (by simple counting) are very close to the estimations of the EM algorithm. For most of the blocks ($> 85\%$), both PedPhase and MRH were able to find haplotype configurations with 0 recombinants. The results show that rule-based haplotyping methods such as PedPhase and MRH could be very useful for large scale haploptying projects on pedigree data because they obtain results (*i.e.* common haplotypes) similar to those of EM algorithms and run much faster than EM algorithms.

The rest of this paper is organized as follows. We first introduce the biological significance of the MRHC problem and some directly related biological concepts and terminology. This is followed by a formal definition of the problem and the computational complexity result. The two (block-extension and 0-recombinant) algorithms are presented in sections 3 and 4. After showing the experimental results in section 5, we conclude the paper with some remarks about possible future work in section 6.

2 The MRHC problem and its NP-hardness

In this section, we first give a formal definition of the MRHC problem, including the necessary biological background, and then prove that the problem is NP-hard even if the input pedigree data contains only two loci.

Definition 2.1 *A pedigree graph is a connected directed acyclic graph (DAG) $G = \{V, E\}$, where $V = M \cup F \cup N$, M stands for the male nodes, F stands for the female nodes, N stands for the mating nodes, and $E = \{e = (u, v)$: $u \in M \cup F$ and $v \in N$ or $u \in N$ and $v \in M \cup F\}$. $M \cup F$ are called the individual nodes. The in-degree of each individual node is at most 1. The in-degree of a mating node must be 2, with one edge starting from a male node (called father) and the other edge from a female node (called mother), and the out-degree of a mating node must be larger than zero.*

In a pedigree, the individual nodes adjacent *to* a mating node (*i.e.* they have edges from the mating node) are called the *children* of the two individual nodes adjacent *from* the mating node (*i.e.* the father and mother nodes, which have edges to the mating node). The individual nodes that have no parents (in-degree is zero) are called *founders*. For each mating node, the induced subgraph containing the father, mother, mating, and child nodes is called a *nuclear family*. A *parents-offspring trio* consists of two parents and one of their children. A *mating loop* is a cycle in the graph if the directions of edges are ignored. Figure 1 shows an illustration of an example pedigree that highlights mating nodes on top of a conventional drawing of the pedigree (where the mating nodes are omitted). For convenience, we will use conventional drawings of pedigrees throughout the paper. Figure 9 shows a pedigree with a mating loop. [4]

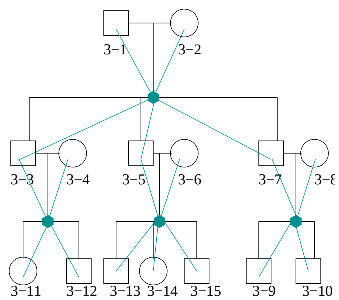

Figure 1: An illustration of a pedigree with 15 members. A square represents a male node, a circle represents a female node, and a solid (round) node represents a mating node. The children (*e.g.* 3-3, 3-5 and 3-7) are placed under their parents (*e.g.* 3-1 and 3-2).

The genome of an organism consists of *chromosomes* that are double strand DNA. Locations on a chromosome can be identified using *markers*, which are small segments of DNA with some specific features. A position of markers on the 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 markers. Different sets of markers have different properties, such as the total number of different allelic states at one locus, frequency of each allele, distance between two adjacent loci, *etc.* A microsatellite marker usually has several different alleles at a locus (called *multi-allele*) while an SNP marker can be treated as a *biallele*, 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 of a locus will be denoted using a set, *e.g.* $\{a, b\}$. If the two alleles 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 2 illustrates the above concepts.

[4]The pedigree diagrams in this paper were generated using WPEDRAW [3].

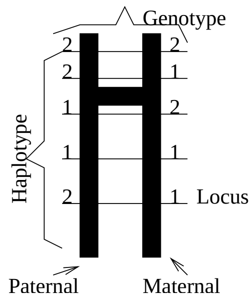

Figure 2: The structure of a pair of chromosomes from a mathematical point of view.

The Mendelian law of inheritance states that the genotype of a child must come from the genotypes of its parents at each marker locus. 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 3 illustrates an example where the paternal haplotype of member 3 is the result of a recombinant.

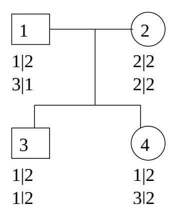

Figure 3: An example recombination event. The notation $i|j$ means that the haplotype information at the locus has been resolved, and we know that allele i is from the father and allele j is from the mother.

Alleles are denoted using (identification) numbers, as shown in Figure 2. We use *PS (parental source)* to indicate which allele comes from which parent at each locus. The PS value at a heterozygous locus can be $-1, 0$ or 1, where -1 means that the parental source is unknown, 0 means that the allele with the smaller identification number is from the father and the allele with the larger identification number is from the mother, and 1 means the opposite. The PS value will always be set as 0 for a homozygous locus. A locus is *PS-resolved* if its PS value is 0 or 1. For example, both loci of member 4 in Figure 3 are PS-resolved and their PS values are 0 and 1, respectively.

For convenience, we will use *GS (grand-parental source)* to indicate if an allele at a PS-resolved locus comes from a grand-paternal allele or a grand-maternal allele. Similar to a PS value, a GS value can also be -1, 0 or 1. The PS and GS information can be used to count the number of recombinants as follows.

For any two alleles that are at adjacent loci and from the same haplotype, they induce a *recombinant* (or *recombination event*) if their GS values are 0 and 1. An allele *GS-resolved* if its GS value is 0 or 1. A locus is *GS-resolved* if both of its alleles are GS-resolved.

Definition 2.2 *A haplotype configuration of a pedigree is an assignment of nonnegative values to the PS of each locus and the GS of each allele for each member of the pedigree that is consistent with the Mendelian law.*

Hence a haplotype configuration not only fully describes the haplotypes in the members of the pedigree, it also describes the origin of each allele on the haplotypes. The following problem, called MRHC in the above, has been studied in [21, 26]:

Definition 2.3 *Given a pedigree graph and genotype information for each member of the pedigree, find a haplotype configuration for the pedigree that requires the minimum number of recombinants.*

The following lemma shows that it suffices to compute only the required PS values in MRHC, because the corresponding GS values can be easily determined to minimize the number of recombinants once the PS values are given. However, we will need use both PS and GS values in one of our algorithms for convenience.

Lemma 2.4 *Given an instance of MRHC and the PS value for each locus of each individual, the GS value of each allele to achieve the minimum number of recombinants can be computed in $O(mn)$ time, where m is the number of loci and n is the number of individuals in the input pedigree.*

Proof: We can consider each parent and child relationship, and figure out an optimal assignment of GS values to the paternal (or maternal) alleles in the child by a simple dynamic programming algorithm. ■

Unfortunately, we can prove that MRHC is NP-hard and thus does not have a polynomial-time algorithm, unless P = NP. We note in passing that NP-hardness results were recently shown for several formulations of pedigree analysis in [1, 19].

Theorem 2.1 *MRHC is NP-hard.*

We prove the theorem by two lemmas. First, recall that the problem of *exact cover* by 3-sets where no element of the universe occurs in more than 3 input subsets (denoted as 3XC3, also called *3-dimensional matching*) is NP-hard [2]. Here we need consider a stronger version of 3XC3, denoted as 3XCX3, where each element of the universe occurs in exactly 3 input subsets. Lemma 2.5 shows that the 3XCX3 problem is NP-hard by a reduction from 3XC3. We then show in Lemma 2.6 that even with two loci, MRHC is NP-hard by a reduction from 3XCX3.

Lemma 2.5 *3XCX3 is NP-hard.*

Proof: The reduction is from 3XC3, which is known NP-hard [2]. Recall that for an instance of 3XC3, we have a set X of $3q$ elements and a collection $\mathcal{C}$ of n 3-element subsets of X where each element of X occurs in at most 3 subsets. We want to construct an instance of 3XCX3 such that each element occurs in exactly 3 subsets. Without loss of generality, let us assume that each element in X occurs in at least two subsets. Let s denote the number of elements occurring in exactly 3 subsets and t denote the number of elements only occurring in 2 subsets. We have $3s + 2t = 3n$. Thus, t must be a multiple of 3 and we can group the elements occurring in 2 subsets so that each group has 3 such elements. For a group of three elements a, b and c, construct four new 3-subsets $\{a, x, y\}, \{b, y, z\}, \{c, z, x\}$ and $\{x, y, z\}$, where x, y, z are new elements. This guarantees each element of X appears in exactly three subsets. It is easy to see the one-to-one correspondence between a solution of the 3XC3 instance and a solution of the constructed 3XCX3 instance. ■

The next lemma shows that MRHC on pedigrees with 2 loci (denoted as MRHC2) is NP-hard.

Lemma 2.6 *MRHC2 is NP-hard.*

Proof: We reduce 3XCX3 to MRHC2. Let $S_1, S_2, ..., S_n$ be the n 3-element subsets and X ($|X| = n = 3q$) the universe of a 3XCX3 instance. We construct a pedigree with genotype information at both loci. Both loci are biallelic and we denote the alleles as $\{1, 2\}$. For convenience, let us first ignore the gender of each individual node in the construction. For each subset S_i, we construct an (individual) node, still denoted by S_i. All such nodes have the same genotypes {1,2} at both loci. Suppose that an element a of X is contained in subsets S_1, S_2 and S_3. We include a small pedigree that consists of the nodes created from S_1, S_2 and S_3 (which we call the S-nodes) and some other nodes (which we will call A, B, C, D-nodes) as their relatives. Some of the A, B, C, D-nodes will be forced to have certain haplotype configurations by setting their and their relatives' (more precisely, mates' and children's) genotypes carefully (more details will be discussed below). Here, a forced haplotype configuration is the *unique* configuration that would minimize the number of recombinants required for the small gadget pedigree as well as for the whole pedigree. Let $\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}$ mean that alleles a_1 and a_2 form a haplotype and b_1 and b_2 form the other haplotype of some individual. Call such a configuration a *haplotype grouping*. [5]

The purpose of the gadget pedigree shown in Figure 4 is to force exactly one of the S-nodes to have haplotype grouping $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$ (via a recombination from its parents) and the other two to have haplotype groupings $\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$. The one with haplotype $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$ will correspond to the subset S_i that is included in the solution of 3XCX3 covering a. In the small pedigree, S_1 and S_2 mate and have four child nodes C_1, C_2, C_3 and C_4. We force C_1 to have haplotype grouping $\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$, and C_2 and C_3 to have haplotype groupings $\begin{bmatrix} 1 & 2 \\ 2 & 2 \end{bmatrix}$. To force C_1 to have the desired haplotype grouping, we could construct some new individuals as relatives (mate and children) of C_1 so that the desired haplotype grouping will benefit the whole pedigree. For example, we can

[5] Note that, this notation only shows the two haplotypes instead of the actual haplotype configuration. However, based on the minimum recombinant principle, we can easily compute an optimal haplotype configuration from such a grouping in the reduction. Thus, for convenience, we will treat $\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}$ as a haplotype configuration in this proof.

create a mate of C_1 (not shown in the figure) that has genotypes $\{1,1\}$ on both loci. We also create some constant number of children of C_1 (all of them are represented by the polygon connected with C_1 in Figure 4), all with genotypes $\{1,2\}$ on both loci. This will force C_1 to have the desired haplotype grouping in order not to incur any recombinants in its sub-pedigree. Each S_i has two parents A_i and B_i with genotypes $\{1,2\}$ on both loci. Their haplotype groupings are forced to be $\begin{bmatrix} 1\,2 \\ 1\,2 \end{bmatrix}$ (by including some constant number children with genotypes $\{1,1\}$ on both loci, not shown in the figure). The A- and B-nodes are introduced to minimize the number of S-nodes that are assigned the haplotype grouping $\begin{bmatrix} 1\,2 \\ 2\,1 \end{bmatrix}$.

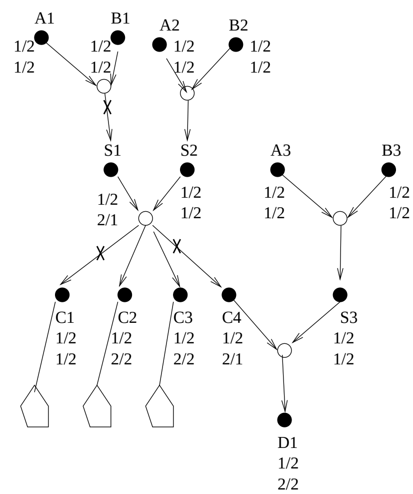

Figure 4: The gadget pedigree for an element of X and the three subsets containing the element. A $\bullet$ indicates an individual node. A $\circ$ indicates a mating node. A $\times$ indicates a recombination event. A / between two alleles indicates the haplotype grouping without specifying the PS value.

Let us assume that C_4 has genotypes $\{1,2\}$ on both loci, and produces a child D_1 with S_3. We force D_1 to have the haplotype grouping $\begin{bmatrix} 1\,2 \\ 2\,2 \end{bmatrix}$ (D_1's mate and children are not shown in Figure 4). Without considering the order, there are three haplotype groupings for S_1 and S_2, namely, $\begin{bmatrix} 1\,2 \\ 1\,2 \end{bmatrix} \times \begin{bmatrix} 1\,2 \\ 1\,2 \end{bmatrix}$, $\begin{bmatrix} 1\,2 \\ 1\,2 \end{bmatrix} \times \begin{bmatrix} 1\,2 \\ 2\,1 \end{bmatrix}$ and $\begin{bmatrix} 1\,2 \\ 2\,1 \end{bmatrix} \times \begin{bmatrix} 1\,2 \\ 2\,1 \end{bmatrix}$. There are two possible haplotype groupings for S_3, *i.e.* $\begin{bmatrix} 1\,2 \\ 2\,1 \end{bmatrix}$ or $\begin{bmatrix} 1\,2 \\ 1\,2 \end{bmatrix}$. By examining the six combinations of the above assignments, we know that if only one of the three S-nodes has haplotype grouping $\begin{bmatrix} 1\,2 \\ 2\,1 \end{bmatrix}$ and the other two have haplotype groupings $\begin{bmatrix} 1\,2 \\ 1\,2 \end{bmatrix}$, the above little gadget pedigree can be resolved with just two recombinants on the C-nodes and one recombinant on the S-nodes. The latter recombinant may also be "shared" by two other gadget pedigrees corresponding to two elements of X. Figure 4 illustrates such a possible haplotype (grouping) assignment with two recombinants on the C-nodes. (The actual PS and GS values are not shown in the figure, although they can be easily determined.) Otherwise, the gadget pedigree would require at least three recombinants on the C-nodes. The above construction does not take into account the gender of each individual, especially that of an S-node. To solve the gender problem, we duplicate every gadget pedigree constructed above. (Actually, we need only duplicate the S-nodes, although there is no harm to duplicate the C-nodes and D-nodes.) So, there are two nodes for each S_i. We make one male and the other one female. Thus we could arrange any two S-nodes to be of opposite genders whenever they have a mating relation. The only thing left is that we need make sure that the two nodes for each S_i must always have the same haplotype assignments. For this, we can construct another small gadget pedigree for each such pairs of S-nodes. Let these two nodes mate and have two children E_1 and E_2. Both E_1 and E_2 have the same genotypes $\{1,2\}$ at both loci. We can see that only if the two parent nodes have the same pair of haplotypes (not necessarily with the same PS values), the small pedigree can be realized with zero recombinants. Otherwise, at least two recombinants would be needed. This constraint is also illustrated in Figure 6 (picture on the right).

The above construction can be completed in polynomial time. It is easy to prove that an exact cover of X corresponds to a haplotype configuration of the above pedigree with $(3q \cdot 2 + q) \cdot 2 = 14q$ recombinants, and vice versa. Hence, 3XCX3 reduces to MRHC2, and MRHC2 is NP-hard. ■

The above reduction uses pedigrees with mating loops (which are formed when the gadget pedigrees in Figure 4 are "glued" together through the S-nodes). Very recently, K. Doi proved that MRHC is NP-hard even if the input pedigree structure has no mating loops, by a reduction from MAX CUT [5]. However, the reduction requires pedigrees with unbounded degrees and numbers of loci.

3 An efficient iterative algorithm for MRHC

Because of the NP-hardness of MRHC, it is unlikely to find an efficient algorithm to solve MRHC exactly. We thus propose an iterative heuristic algorithm for MRHC, called the *block-extension* algorithm. Here, a block means a consecutive sequence of resolved loci of some individual. The basic idea of the algorithm is to partition the loci in each member of the pedigree into blocks after applying the Mendelian law and some simple greedy strategy (such as avoiding double recombinants within a small region of loci). It then repeatedly uses the longest haplotype block in the pedigree to extend other blocks (in other members) by resolving unresolved loci based on the minimum recombination principle. This approach may potentially be more efficient than the algorithm MRH from [21], especially when the number of required recombinants is small, because multiple loci are resolved simultaneously instead of separately. More precisely, the block-extension algorithm is based on the following observations:

Observation 1: Long haplotype blocks are common in human genomes [4, 8, 10]. Few or zero recombinants are expected within these blocks. In our simulation study, we have also observed a lot of long blocks in each member of the pedigree after applying the Mendelian law.

Observation 2: Shared haplotype blocks among siblings are strong evidence that no recombination should occur on those siblings based on the assumption that recombinants are rare in a pedigree. Thus, these haplotype blocks shared by siblings should also exist in their parents. It is therefore reasonable to use blocks in children to resolve the corresponding loci in their parents.

Observation 3: Double recombinants are very rare within a small region. Once we know that two nonadjacent alleles on a haplotype block have the same GS value, the alleles of the block sandwiched by the two alleles should have the same GS value. Otherwise, double recombinants would occur.

Observation 4: It is sometimes possible to determine that some individuals must involve recombinants. We may be better off to leave these individuals until the last.

We now describe the block-extension algorithm in more detail. The main steps of the algorithm are also illustrated by an example. The algorithm has 5 steps. Assume that the input genotype data is consistent with Mendelian law. (One can use the genotype-elimination algorithm of O'Connell's [17, 18] to check Mendelian consistency, although Aceto *et al.* [1] recently showed that consistency checking in general is NP-complete.) Note that, although it suffices to focus on the PS values by Lemma 2.4, we will use both PS and GS in the algorithm for the convenience of presentation.

Step 1: Missing genotype imputing by the Mendelian law. We scan the whole pedigree bottom-up and consider the nuclear families one by one. For each parent that has missing data, we check if there is an allele in any of its children that does not appear in its mate whose genotype is known. We may also impute some missing alleles in a child/parent when the alleles of a parent/child at some homozygous locus do not appear in the child/parent's genotype. But, only part of the missing data may be imputed by this method. More missing data in founders can be imputed by first counting the allele frequency and then randomly sampling according to the allele distribution. Missing data in children are then determined by randomly sampling from their parents.

Step 2: PS and GS assignments by the Mendelian law. By a top-down scan, we try to resolve all loci that have unambiguous PS and/or GS values under the Mendelian law. For the founders of the pedigree, we can resolve all their homozygous loci and one arbitrary heterozygous locus by setting the PS and GS values of all these loci to be 0. To resolve a locus in a non-founder, let us consider each parents-offspring trio. There are two rules we can use to resolve loci in the child: 1) if there is at least one parent of the trio that is homozygous at this locus, we can resolve PS value of the locus in the child, and 2) if there is an allele in the child that differs from both alleles in one parent, it must come from the other parent. Here, we may also be able to determine the GS values of some alleles in the child using the PS information at the parents.

By running these two steps, we should get similar results as running the first two steps of MRH. But our method is much simpler than MRH. There are more than 40 (detailed) rules [21] used in MRH for missing data imputing and PS and GS assignments.

Step 3: Greedy assignment of GS values. The greedy step to assign GS works in a bottom-up fashion. We begin with a lowest nuclear family in the pedigree. For each child of the family, we check if the child has one or more loci whose alleles have known GS and if all the known GS values of alleles on the same haplotype are the same (which means that at least by now, there is no evidence that recombinants exist in this child). Otherwise, just mark this child as processed and continue on with other children until we find such a child. Based on the known GS information, the GS values of nearby alleles in the child will be assigned according to the following rules. Assume that we know the GS of one (*e.g.* paternal) allele at the

i^{th} locus of the child, and we want to set the GS of its paternal allele at the $(i+1)^{th}$ locus. We check if any of the child's or the father's $(i+1)^{th}$ loci have been PS-resolved. If neither of them are PS-resolved, we cannot assign the GS. If the child is PS-resolved and the father is homozygous, we set the GS of the paternal allele of the child at the $(i+1)^{th}$ locus equal to the GS of the paternal allele at the i^{th} locus. If the child is PS-resolved but the father is PS-unresolved (and thus heterozygous), we will try the two alternative GS assignments for the paternal allele of the child at locus $i+1$. Each assignment would force the PS value of the father at locus $i+1$ thus resolve its PS. Then we count the numbers of recombinants between loci i and $i+1$ within the nuclear family for these two choices, and select the one with fewer recombinants. If the child is not PS-resolved at locus $i+1$, but the parent is PS-resolved, we consider the two alternative PS (and thus GS) assignments for the child at locus $i+1$, and select the one with fewer recombinants. If the numbers of recombinants for the two alternative assignments are equal, we do nothing. After each known GS value has been processed, we mark the child as processed and continue on with other children in this nuclear family. During the process, if we see that two non-adjacent alleles on the paternal (or maternal) haplotype have the same GS, we will assign alleles from the same haplotype sandwiched by the two alleles the same GS value as mentioned in observation 3. Once this nuclear family is finished, we continue the GS assignment in another nuclear family by a bottom-up scan.

Step 4: Block-extension. Find a longest haplotype block in some member of the pedigree consisting of alleles with the same GS value. We use this haplotype block to resolve the corresponding loci in the first degree relatives (children and parents) of the member. Each decision on a locus will be made by counting the numbers of recombinants in the nuclear family resulted by the two alternative PS assignments of the locus. The assignment with the smaller number of recombinants will be selected. Recently resolved members will be used to resolve their first degree relatives again, and this is continued so that every member in the pedigree will be processed with respect to this block. During this process, the haplotype block in some members may overlap with other blocks thus the longest block may get extended to result in longer blocks. We repeat the above process with the current longest block that has not be considered yet, until no blocks can be extended.

Step 5: Finishing the remaining gaps. After step 4, it is still possible that there are some gaps of PS-unresolved loci between blocks. Such gaps may exist only when the input data contain special patterns. For example, at some locus, all the members of the pedigree are heterozygous or contain missing data. Another possible scenario is that for some three adjacent loci, no members of the pedigree have two adjacent PS-resolved loci. If we assume that alleles occur with the same frequency at a locus, it is not difficult to see that the probability that we have the above scenarios is very small. (These situations never happened in our simulation study.) When a gap does occur, the algorithm would pick an unresolved locus, compare two alternative PS-assignments within a nuclear family at the locus and select the one that gives the smaller number of recombinants. Hopefully this assignment will extend some blocks, and we will repeat steps 3-5 again until all gaps are filled.

Figure 5 illustrates the main steps of the above algorithm using a simple example. Since the input pedigree and genotype information, as given in (A), have no missing data, we skip step 1. (B) shows the result of step 2. In step 3, since we know the GS values of the alleles at the first locus of member 3-3, we assign the adjacent alleles (at loci 2 to 4) the same GS value. This forces locus 2 at both parents of member 3-3 to be PS-resolved. The result of step 3 is shown in (C). In step 4, we use the longest block (loci 1 to 4 in member 3-1) to help resolve the same region in member 3-4 (by consulting the corresponding region in member 3-2). This results in a longer block in member 3-4 as shown in (D). We then use the longest block in member 3-4 to help resolve loci 5 and 6 in members 3-1 and 3-2 and the blocks in members 3-1 and 3-2 to help resolve locus 5 in member 3-3, as shown in (E). In step 5, we consider the two alternative PS assignments for locus 7 in member 3-1. For each such assignment, we calculate the best PS assignments for locus 7 in the other members of the same nuclear family (*i.e.* members 3-2, 3-3, and 3-4) in order to minimize the number of recombinants required. The assignment shown in (F) turns out to be the best choice, and the final haplotype configuration is shown in (G), which happens to require one recombinant.

Theorem 3.1 *Let n denote the size of the pedigree, m the number of loci, and d the largest number of children in a nuclear family. The block-extension algorithm runs in $O(dmn)$ time.*

Proof: The worst-case time complexity of the algorithm can be analyzed as follows. The first step only involves a bottom-up scan and thus runs in $O(dmn)$ time, because each nuclear family is visited exactly once and for each nuclear family, we may spend at most $O(dm)$ time to impute missing data in the nuclear family. Similarly, steps 2 and 3 run in $O(dmn)$ time each. In step 4, each locus is involved in at most one block extension operation. Since there are totally mn loci and the time for setting the PS value at a locus is at most $O(d)$, this step also takes $O(dmn)$ time. In step 5, every time we fix the PS value of an unresolved locus, we spend $O(d)$ time and then call steps 3 and 4 to see if more loci can be resolved. Hence, it takes at most $O(d)$ time to resolve a locus in this step, and step 5 takes at most $O(dmn)$ time totally. In summary, the block-extension algorithm runs in $O(dmn)$ time. ■

4 A constraint-based algorithm for 0-recombinant data

As a heuristic, the above block-extension algorithm does not always compute an optimal solution for MRHC, especially when the number of required recombinants increases. One of our ultimate objectives is to design an algorithm that runs fast enough on real data and always gives an optimal or almost optimal haplotype configuration, even if its running time is exponential in the worst case. Our first attempt is an efficient (polynomial-time) algorithm to compute all possible haplotype assignments involving no recombinants. Not only will the algorithm be useful for solving 0-recombinant data (*i.e.* data that can be interpreted with zero recombinants, which are common for organisms like human as mentioned in the introduction), it may also serve as a subroutine in a general algorithm for MRHC.

We consider only data that has no missing genotypes. [6] Observe that finding a haplotype configuration

[6] The Mendelian consistency can be easily checked in this case and thus we present the algorithm assuming the input data are Mendelian consistent.

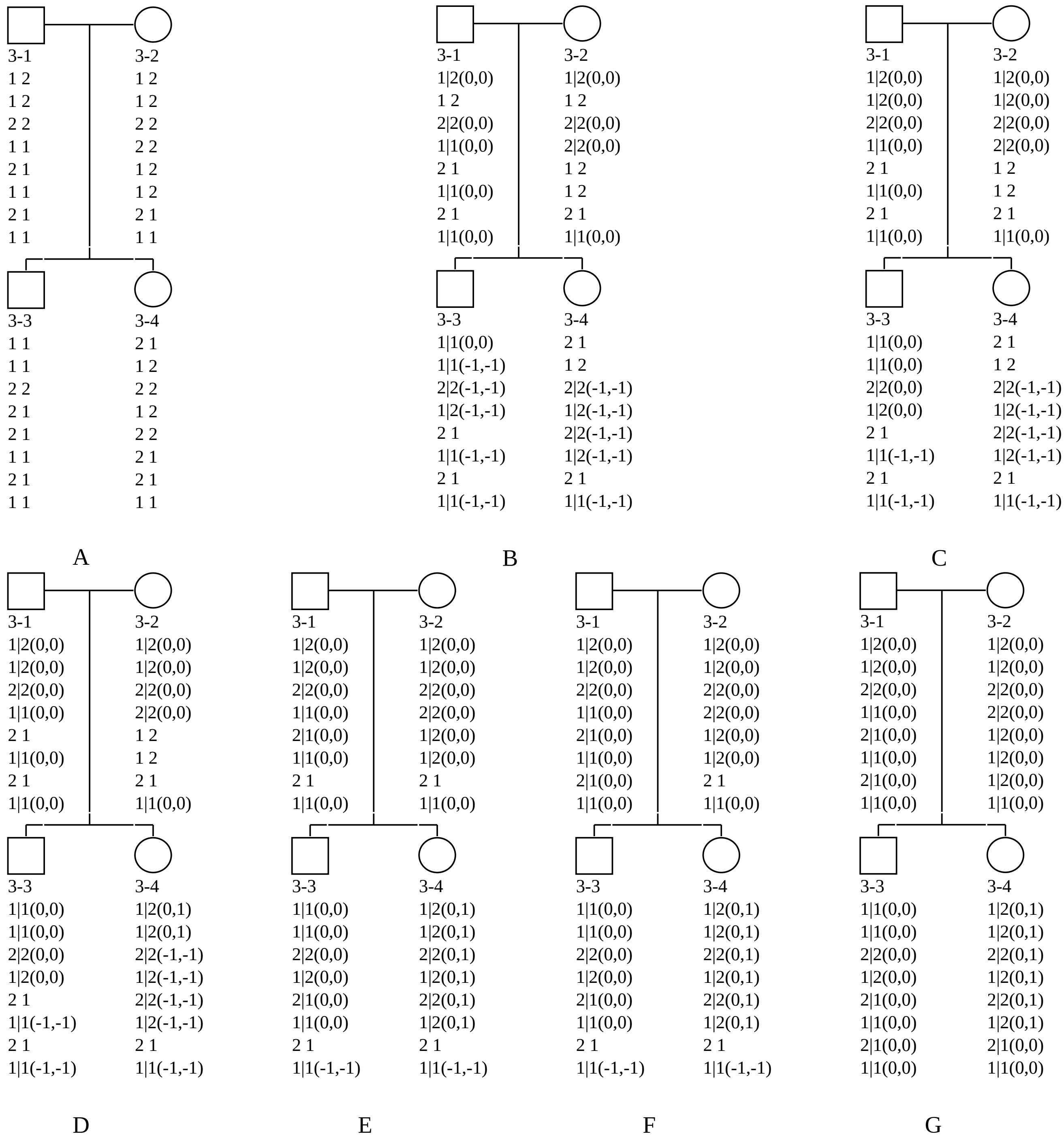

Figure 5: An illustration of the block-extension algorithm. The blank between two alleles at a locus indicates that the locus is PS-unresolved. Again, a | indicates that the locus is PS-resolved. For a PS-resolved locus, we use two numbers in parentheses to indicate the GS values of the paternal and maternal alleles.

is equivalent to reducing the degree of freedom in the PS/GS assignment for each locus/allele in every individual. We may thus find all necessary *constraints* on the PS/GS assignments first so that the freedom left is really *free* (*i.e.* the choices will always lead to 0-recombinant haplotype configurations). Then we can simply enumerate all haplotype configurations satisfying the constraints.

We define four levels of constraints. The first level of constraints are the strongest and specify specific nonnegative values for the involved *alleles* (*i.e.* these alleles are GS-resolved). The second level of constraints specify specific nonnegative values for the involved *loci* (*i.e.* these loci are PS-resolved). The last two levels of constraints are concerned with *two loci*. The third level of constraints describe what alleles should be on the same haplotype in *one member* without specifying the actual PS values. For example, in Figure 6 (left), the PS-resolved loci in member 3 forces the parents have the same haplotype grouping in order to obtain a solution without recombinants, although we do not know the actual PS of the two haplotypes in the grouping. The forth level constraint are the weakest. Each level 4 constraint is concerned with the relationship between the haplotype groupings in some *different individuals* (*i.e.* parent-child or mates) at two loci, although it does not specify the actual haplotype grouping. For example, in Figure 6 (right), the three members of the parents-offspring trio should either all have the haplotype grouping $\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$ or all have the haplotype groupings $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$, in order not to incur any recombinant in the trio. But, we do not have information to determine which case must hold. All possible types of level 3 and level 4 constraints will be listed below.

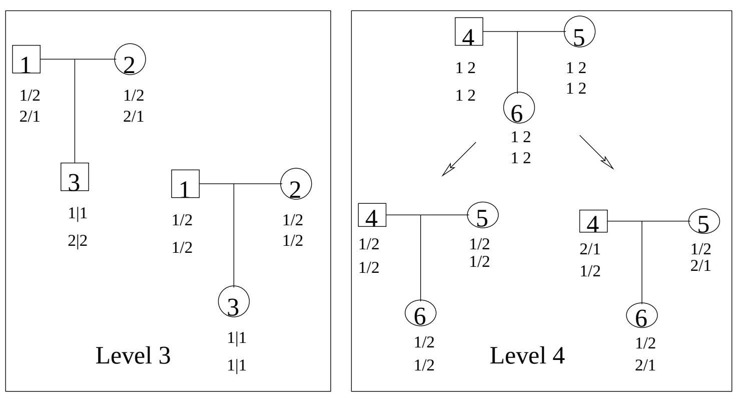

Figure 6: An illustration of level 3 and level 4 constraints.

Although the first level of constraints are useful in detecting recombinants, Lemma 2.4 suggests that only the last three levels of constraints are really necessary for computing feasible (0-recombinant) solutions. For each locus i in member j, we define a binary variable $x_{i,j}$ to represent the PS value of the locus. The basic idea of our algorithm is to identify all the level 2-4 constraints based on the Mendelian law and the 0-recombinant assumption by examining every parents-offspring trio, and represent the constraints as linear equations on $x_{i,j}$'s over the cyclic group Z_2. It then finds all feasible 0-recombinant solutions by solving the equations.

	x	y	z	Constraint equations
1	$\begin{bmatrix}1&2\\1&2\end{bmatrix}$	$\begin{bmatrix}1&*\\1&*\end{bmatrix}$	$\begin{bmatrix}1&1\\1&1\end{bmatrix}$	$x_1 = x_2$
2	$\begin{bmatrix}1&2\\2&1\end{bmatrix}$	$\begin{bmatrix}1&*\\2&*\end{bmatrix}$	$\begin{bmatrix}1&1\\2&2\end{bmatrix}$	$x_1 + x_2 = 1$
3	$\begin{bmatrix}1&2\\2&1\end{bmatrix}$	$\begin{bmatrix}1&*\\1&1\end{bmatrix}$	$\begin{bmatrix}1&1\\2&1\end{bmatrix}$	$x_1 + x_2 = 1$
4	$\begin{bmatrix}1&2\\1&2\end{bmatrix}$	$\begin{bmatrix}1&1\\1&1\end{bmatrix}$	$\begin{bmatrix}2&1\\2&1\end{bmatrix}$	$x_1 = x_2$

Table 1: The possible level 3 constraints.

	x	y	z	Constraint equations
1	$\begin{bmatrix}1&2\\1&2\end{bmatrix}$	$\begin{bmatrix}1&2\\1&2\end{bmatrix}$	$\begin{bmatrix}1&2\\1&2\end{bmatrix}$	$x_1 + x_2 = y_1 + y_2 = z_1 + z_2$
2	$\begin{bmatrix}1&2\\1&2\end{bmatrix}$	$\begin{bmatrix}1&2\\2&1\end{bmatrix}$	$\begin{bmatrix}1&2\\1&1\end{bmatrix}$	$x_1 + x_2 = z_1, y_1 + y_2 + z_1 = 1$
3	$\begin{bmatrix}1&2\\1&2\end{bmatrix}$	$\begin{bmatrix}1&2\\1&1\end{bmatrix}$	$\begin{bmatrix}2&1\\2&1\end{bmatrix}$	$x_1 + x_2 = z_1 + z_2$

Table 2: The possible level 4 constraints.

More specifically, the level 2 constraints are collected locus by locus by examining every parents-offspring trio, which is the same as step 2 of the block-extension algorithm. In order to collect level 3 and level 4 constraints in the form of linear equations on the binary PS variables, we need to consider pairs of loci for each parents-offspring trio. Without distinguishing the two parents in a parents-offspring trio, there are essentially four types of level 3 constraints and three types of level 4 constraints as summarized in Tables 1 and 2. In the tables, x, y are the parents and z is the child. An * indicates any allele. x_1 represents the binary PS variable for locus 1 in member x. These constraints are collected trio by trio.

Definition 4.1 *Given the level 2-4 constraints defined above, a consistent solution is an assignment of binary values to all the PS variables that satisfies every constraint.*

Clearly, every feasible 0-recombinant solution is consistent. The following theorem shows that the converse is also true.

Theorem 4.1 *Every consistent solution is a feasible* 0*-recombinant solution.*

Proof: Consider a haplotype configuration that is consistent with all constraints. By Lemma 2.4, we can find a GS assignment for each allele so that the pedigree has the minimum number of recombinants. Suppose that the number of recombinants is not zero. Let member B involve a recombinant between loci i and $i+1$ and A the corresponding parent (*i.e.* father) of B. Find the largest j that $j \leq i$ and the smallest k that $k \geq i+1$ such that A is heterozygous at both loci j and k. Such j and k must exist, because otherwise we could remove the recombinant by modifying the GS values of relevant paternal alleles in B. Since A and B are involved in some level 4 (or level 3 or level 2) constraint at loci j and k, the consistency of the solution means that the PS assignments of A and B at loci j and k does not involve any recombinant between the two loci. Hence, there must be recombinants in the paternal haplotype of B between loci j and i or between loci $i+1$ and k. We could easily modify the GS values of the paternal alleles of B in the involved segment(s) to reduce the number of recombinants without affecting the PS assignments, since one of the involved loci must be homozygous. This contradicts the the assumption that the GS values were optimized. ■

The above constraints form a system of (sparse) linear equations over the group Z_2, which could be solved by the classical Gaussian elimination method running in cubic time (see *e.g.* [24]). Since we are dealing with Z_2, a much simpler algorithm (adapted from Gaussian elimination) is presented below for the completeness of the paper. Before we describe the algorithm, let us reduce the number of level 3 and level 4 constraints (equations) required since many of them are easily seen as redundant. For any given parents-offspring trio, we introduce a level 3 (or level 4) constraint for a pair of loci i and j if and only if at least one of the parents is heterozygous at both i and j and is homozygous at every locus between i and j. These constraints are sufficient to guarantee a feasible solution by the proof of Theorem 4.1. Hence, each parents-offspring trio may give rise to at most $2m-2$ level 3 (or level 4) constraints, where m is the number of loci. If the number of individuals is n in the input pedigree, then we have at most $2(m-1)n$ level 3 and level 4 constraints.

Suppose that the PS variables are denoted as $x_1, x_2, \ldots, x_{mn}$. Our algorithm first removes all equations that contain only one variable (*i.e.* level 2 constraints), and replaces the involved variables by their constant PS values in other equations. This may result in more single-variable equations, and we iterate the process until all equations contain two or more variables. We then process two-variable equations by substituting variables with lower indices for variables with higher indices that are supposed to have equal values. The remaining equations now have three or more variables, and we perform the general Gaussian elimination over Z_2. Suppose that x_j has the highest index among the remaining variables, and it is defined by equations:

$$\begin{aligned}
x_j &= x_{a_1} + x_{a_2} + \ldots + x_{a_p}, \\
x_j &= x_{b_1} + x_{b_2} + \ldots + x_{b_q}, \\
& \quad \ldots \\
x_j &= x_{c_1} + x_{c_2} + \ldots + x_{c_r}, \\
x_j &= x_{d_1} + x_{d_2} + \ldots + x_{d_s},
\end{aligned}$$

where $a_1, a_2, \ldots, a_p, b_1, b_2, \ldots, b_q, c_1, c_2, \ldots, c_r, d_1, d_2, \ldots, d_s < j$. We can transform the equations as follows:

$$\begin{aligned}
x_j &= x_{a_1} + x_{a_2} + \ldots + x_{a_p}, \\
0 &= x_{a_1} + x_{a_2} + \ldots + x_{a_p} + x_{b_1} + x_{b_2} + \ldots + x_{b_q}, \\
& \quad \ldots \\
0 &= x_{c_1} + x_{c_2} + \ldots + x_{c_r} + x_{d_1} + x_{d_2} + \ldots + x_{d_s}.
\end{aligned}$$

This leaves only one equation that defines x_j in terms of other variables. We can continue this process to remove all but one constraint equation for each of the variables in the system. If we detect any conflicts in the process, we know that there are no feasible solutions. Any variable that is not defined by any equation is free and can be given any PS value in a feasible solution. Thus, if there are p free variables at the end, the total number of feasible 0-recombinant solutions is 2^p.

The running time of the above algorithm is $O(m^3n^3)$, since the above elimination process takes at most mn iterations and each iteration takes at most $O(m^2n^2)$ time because the number of equations never grows and the size of each equation is at most mn.

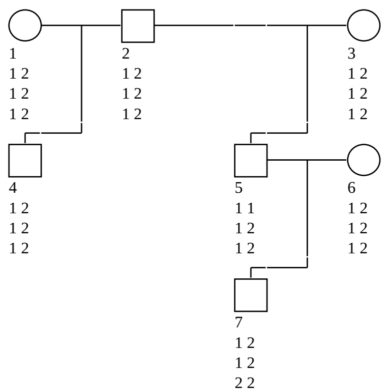

Figure 7: An example to illustrate the constraint-based algorithm.

We use an example to illustrate how the above constraint-based algorithm works. The input pedigree and genotypes are shown in Figure 7. There are 3 loci, 7 individuals and 3 parents-offspring trios. For the first trio $(1, 2, 4)$, we derive constraints

$$\begin{aligned} x_{1,1} + x_{2,1} &= x_{1,4} + x_{2,4} \\ x_{2,1} + x_{3,1} &= x_{2,4} + x_{3,4} \\ x_{1,2} + x_{2,2} &= x_{1,4} + x_{2,4} \\ x_{2,2} + x_{3,2} &= x_{2,4} + x_{3,4}. \end{aligned}$$

For the second trio $(2, 3, 5)$, we derive constraints

$$\begin{aligned} x_{1,2} + x_{2,2} &= x_{2,5} \\ x_{2,2} + x_{3,2} &= x_{2,5} + x_{3,5} \\ x_{1,3} + x_{2,3} &= x_{2,5} + 1 \\ x_{2,3} + x_{3,3} &= x_{2,5} + x_{3,5}. \end{aligned}$$

For the third trio $(5, 6, 7)$, we derive constraints

$$x_{1,5} = 0$$

$$
\begin{aligned}
x_{1,7} &= 0 \\
x_{2,5} + x_{3,5} &= x_{2,7} + 1 \\
x_{1,6} + x_{2,6} &= x_{1,7} + x_{2,7} \\
x_{2,6} + x_{3,6} &= x_{2,7} \\
x_{3,7} &= 0.
\end{aligned}
$$

By performing the above adapted Gaussian elimination, we end up with a system of equations that define all but 7 (free) variables.

5 Preliminary experimental results

We have implemented the above block-extension algorithm as a C++ program, *PedPhase*, which is available to the public upon request to either of the authors. To evaluate the performance of PedPhase, we compared PedPhase and MRH on simulated genotype data in terms of accuracy and efficiency using three different pedigree structures. The results show that PedPhase is comparable with MRH in terms of accuracy when the number of recombination events is small and is much faster than MRH on large dataset (large pedigree/large number of loci). We further compared the performance of PedPhase, MRH and an EM algorithm on a real data set that consists of 12 multi-generation pedigrees from a recent paper [8]. The results show that both rule-based haplotyping methods (PedPhase and MRH) could discover almost all *common* haplotypes (*i.e.* haplotypes with frequencies $> 5\%$) that were inferred by the EM algorithm [8]. The haplotype frequencies estimated from the haplotype results of PedPhase and MRH (by simple counting) are also similar to the estimations by the EM algorithm. Most of the blocks ($> 85\%$) were found by PedPhase and MRH to have involved 0 recombinants. The assumptions and observations made in Section 3 and the minimum recombination principle are well supported by this real dataset.

5.1 PedPhase and MRH on a simulated dataset

We compared the two programs in terms of accuracy and efficiency. For both programs, a solution is regarded as correct if its number of recombinants is smaller or equal to the actual number of recombinants used to generate the data. [7] Three different pedigree structures were considered. One is a small pedigree with 15 members as shown in Figure 1. The second is a middle sized pedigree with 29 members as shown in Figure 8 and the third is a pedigree of 17 members but with a mating loop as shown in Figure 9. Both multi-allelic (with 6 alleles per locus) and biallelic data were considered. The alleles were generated following a uniform frequency distribution. 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. For each data set, 100 copies

[7] Because we consider small numbers of recombinants in the simulation, the true haplotype configurations are usually optimal solutions for MRHC. In fact, in most cases they are the unique optimal solutions.

of random genotype data were generated. The total number of data sets used is 9000 ($= 3 \cdot 2 \cdot 3 \cdot 5 \cdot 100$). However, we were not able to run MRH on all these data sets because of its speed, especially for biallelic genotypes.

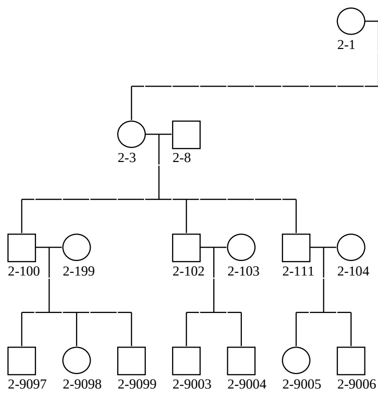

Figure 8: A pedigree with 29 members.

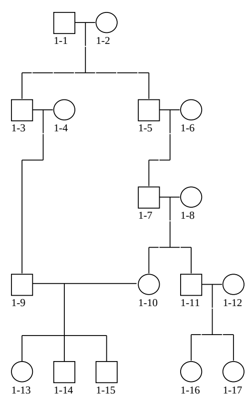

Figure 9: A pedigree with 17 members and a mating loop.

The experimental results demonstrate that our block-extension algorithm is much faster than MRH on both multi-allelic and biallelic data (Table 3 and Table 4). The first column of the tables shows the combination of parameters: the number of members in 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, respectively. The time used by each program is the total time for 100 random runs for each parameter combination, on a Pentium III with 500MHz CPU and 218MB RAM. The gap between the speeds of the two programs is drastic, especially on biallelic data. In all cases, our program could finish 100 runs within one minute. MRH, on the other hand, scaled very poorly. For small pedigree size or a small number of loci, its running time was acceptable although it was much slower than our program. When the pedigree size and the number of loci increase, MRH's speed decreased drastically, especially on biallelic markers that are becoming more and more popular because of SNPs. This is also true for zero-recombinant data. In fact, MRH cannot handle pedigrees of size 29 in general [20], although such pedigrees are only considered moderately large in practice. On the other hand, Qian and Beckmann [21] showed that MRH is faster on average than the genetic algorithm of Tapadar *et al.* [26].

Parameters	Time used by the block-extension algorithm	Time used by MRH
(17,10,6,0)	2.1s	1m16s
(17,10,6,4)	2.1s	2m14s
(15,25,6,0)	2.7s	28m
(15,25,6,4)	2.9s	30m
(29,10,6,0)	3.2s	6m17s
(29,10,6,4)	3.1s	8m47s
(29,25,6,0)	15s	2h58m
(29,25,6,4)	10s	3h9m

Table 3: Speeds of the block-extension algorithm and MRH on multi-allelic markers.

Parameters	Time used by the block-extension algorithm	Time used by MRH
(17,10,2,0)	1.9s	6m17
(17,10,2,4)	2.3s	16m16s
(15,25,2,0)	4.7s	3h43m
(15,25,2,4)	4.8s	4h44m
(29,10,2,0)	2.8s	1h3m
(29,10,2,4)	2.7s	57m
(29,25,2,0)	2.3s	28h
(29,50,2,0)	16s	$\geq$ 20h/run

Table 4: Speeds of the block-extension algorithm and MRH on biallelic markers.

In terms of accuracy, MRH is very good ($> 96\%$ overall) whenever it is able to finish the computation, because of its exhaustive search within a nuclear family. However, the performance of the block-extension algorithm is comparable to that of MRH in most cases. For example, on multi-allelic data, the block-extension algorithm could recover most haplotype configurations correctly, even when the input involves a big pedigrees and a large number of marker loci (Table 5). For biallelic data, the performance of the algorithm is very good when the number of required recombinants is small but becomes worse when the number of recombinants increases (Tables 6 and 7).

5.2 PedPhase, MRH and the EM algorithm on a real dataset

We have also tested PedPhase and MRH on a public dataset from Whitehead/MIT Center for Genome Research. Recently, Gabriel *et al.* [8] reported results on a large scale SNP haplotype block partition and haplotype frequency estimation project. Their original dataset consists of 4 populations and 54 autosomal regions, each with an average size of 250K bps, spanning 13.4M bps (about 0.4%) of the human genome. Haplotype blocks were defined using the normalized linkage disequilibrium parameter D'. Within each block, haplotypes and their frequencies were calculated via an EM algorithm. One of the populations (European) has pedigree information and is of interest to us. There are totally 93 members in the European

Parameters	Percentage correctly recovered out of 100 runs
(15,50,6,0)	100
(15,50,6,4)	91
(17,50,6,0)	100
(17,50,6,4)	91
(29,10,6,0)	100
(29,10,6,4)	99
(29,25,6,0)	100
(29,25,6,4)	95
(29,50,6,0)	100
(29,50,6,1)	96
(29,50,6,2)	93
(29,50,6,3)	95
(29,50,6,4)	91

Table 5: Accuracy of the block-extension algorithm on multi-allelic markers.

Parameters	Percentage correctly recovered out of 100 runs
(15,10,2,0)	100
(15,10,2,4)	96
(15,25,2,0)	98
(15,25,2,4)	78
(15,50,2,0)	100
(15,50,2,1)	82
(17,10,2,0)	97
(17,10,2,4)	92
(17,25,2,0)	100
(17,25,2,1)	84
(17,50,2,0)	100
(17,50,2,1)	72
(29,10,2,0)	95
(29,10,2,4)	93
(29,25,2,0)	100
(29,25,2,1)	91
(29,25,2,2)	87
(29,50,2,0)	100
(29,50,2,1)	88

Table 6: Accuracy of the block-extension algorithm on biallelic markers.

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 1-10 regions. We obtained the results concerning common haplotypes and their frequencies in the European population, as given by the EM algorithm, from the authors of [8]. In this paper, we focus on a randomly selected autosome (*i.e.* chromosome

Number of recombinants in the pedigree	0	1	2	3	4
Number of correct reconstructions out of 100 runs	100	88	72	64	54

Table 7: Accuracy decreases when the number of recombination events increases for the pedigree in Figure 8 with 50 biallelic marker loci.

3). There are 4 regions in the chromosome 3 data and each region is partitioned into 1-4 blocks according to [8]. The physical location and partitioned block information of each region from [8] are summarized in Table 8. We take the genotypes of each of the 10 blocks as our initial input dataset.

Region name	Physical length (kbps)	Genotyped SNPs	Block	SNPs in each block
16a	40	14	1	5
16b	106	53	1	6
			2	4
17a	186	70	1	6
			2	5
			3	4
			4	6
18a	286	74	1	16
			2	6
			3	4

Table 8: The regions and blocks on chromosome 3.

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`). The genotypes have been preprocessed and are consistent with Mendelian law. However, as much as 25% of the alleles could be missing at a particular locus. Since the current version (version 0.1) of MRH cannot handle missing data, PedPhase is used first to impute missing alleles according to the first step in the block-extension algorithm. Completed genotype data are then fed to MRH. Once haplotypes are inferred for the members all pedigrees, haplotype frequencies (in the population) are estimated by simple counting. The common haplotypes and their frequencies in each block, estimated by PedPhase, MRH and the EM algorithm (obtained from the authors of [8]), are summarized in Table 9. The majority (36 out of 39) of the common haplotypes identified by PedPhase and MRH for all blocks are the same as those of the EM algorithm. Furthermore, for the common haplotypes shared by the three programs, all three programs gave frequencies very close to each other. In two of the three cases where PedPhase and MRH obtained slightly different common haplotypes (in blocks 17a-1 and 18a-2), the involved haplotypes have frequencies very close to the threshold (*i.e.* 5%) of common haplotypes. Only one haplotype in block 18a-1 was identified by the EM algorithm with a frequency of 12.5% but missed by both PedPhase and MRH.

Since the pedigrees were very small, both PedPhase and MRH were very fast on this dataset. The common haplotypes given by PedPhase and MRH are always identical. PedPhase successfully reconstructed haplotypes in all 120 (12 pedigree on 10 blocks) cases but MRH failed in 12 cases (no results were given) for some unknown reason. Note that, PedPhase only gives one haplotype configuration for each input, while MRH may produce multiple configurations. In this study, MRH found multiple solutions for 21 cases and we randomly selected one for the calculation of haplotype frequencies. With regard to recombination, most pedigrees ($> 85\%$) can be realized with 0 recombinants. PedPhase and MRH agreed with each other on the number of recombinants in all but three cases. In one case, PedPhase found a solution with 2 recombinants and MRH gave a solution with 4 recombinants. In the other two cases, PedPhase reported

Block	EM		PedPhase		MRH	
	Common haplotypes	Frequencies	Common haplotypes	Frequencies	Common haplotypes	Frequencies
16a-1	4 2 2 2 2	0.4232	4 2 2 2 2	0.3817	4 2 2 2 2	0.3779
	3 4 3 4 4	0.2187	3 4 3 4 4	0.1720	3 4 3 4 4	0.1744
	4 2 2 2 4	0.2018	4 2 2 2 4	0.1935	4 2 2 2 4	0.1802
	3 4 2 2 4	0.1432	3 4 2 2 4	0.1613	3 4 2 2 4	0.1802
16b-1	3 2 4 1 1 2	0.8014	3 2 4 1 1 2	0.7634	3 2 4 1 1 2	0.7849
	1 3 2 3 3 4	0.0833	1 3 2 3 3 4	0.0753	1 3 2 3 3 4	0.0753
16b-2	4 1 2 2	0.5410	4 1 1 2	0.4892	4 1 1 2	0.4826
	2 3 3 4	0.2812	2 3 3 4	0.2581	2 3 3 4	0.2616
	2 3 3 2	0.1562	2 3 3 2	0.1344	2 3 3 2	0.1512
17a-1	3 1 3 4 4 4	0.3403	3 1 3 4 4 4	0.3172	3 1 3 4 4 4	0.3226
	1 3 3 2 4 2	0.3021	1 3 3 2 4 2	0.2419	1 3 3 2 4 2	0.2473
	3 3 2 4 2 4	0.1354	3 3 2 4 2 4	0.0914	3 3 2 4 2 4	0.0914
	3 3 3 4 4 4	0.1021	3 3 3 4 4 4	0.1183	3 3 3 4 4 4	0.1183
	3 3 2 4 4 4	0.0681	3 3 2 4 4 4	0.0806	3 3 2 4 4 4	0.0806
	1 3 3 2 4 4	0.0521				
17a-2	2 3 2 4 2	0.3542	2 3 2 4 2	0.2903	2 3 2 4 2	0.2903
	3 3 4 2 4	0.3333	3 3 4 2 4	0.2957	3 3 4 2 4	0.3118
	3 3 4 4 2	0.1458	3 3 4 4 2	0.1344	3 3 4 4 2	0.1237
	3 4 4 4 4	0.1250	3 4 4 4 4	0.1452	3 4 4 4 4	0.1452
17a-3	4 4 3 1	0.4129	4 4 3 1	0.4355	4 4 3 1	0.4167
	3 1 1 2	0.2813	3 1 1 2	0.2258	3 1 1 2	0.2051
	4 1 3 1	0.2363	4 1 3 1	0.1935	4 1 3 1	0.2115
	4 1 3 2	0.0696	4 1 3 2	0.0753	4 1 3 2	0.0705
17a-4	3 4 4 1 2 4	0.3854	3 4 4 1 2 4	0.3710	3 4 4 1 2 4	0.4429
	2 3 2 4 3 2	0.3333	2 3 2 4 3 2	0.2903	2 3 2 4 3 2	0.2357
	3 4 2 4 2 4	0.2500	3 4 2 4 2 4	0.1881	3 4 2 4 2 4	0.1857
18a-1	1444231214144132	0.2697	1444231214144132	0.2473	1444231214144132	0.1706
	1444111214144132	0.2396	1444111214144132	0.2151	1444111214144132	0.2357
	1444131214144132	0.1887	1444131214144132	0.2204	1444131214144132	0.2176
	4222133313412211	0.1250				
	1444231234144132	0.0833	1444231234144132	0.0699	1444231234144132	0.0764
18a-2	3 1 2 4 4 2	0.4967	3 1 2 4 4 2	0.4892	3 1 2 4 4 2	0.4765
	1 3 2 4 3 4	0.2604	1 3 2 4 3 4	0.1935	1 3 2 4 3 4	0.1765
	3 1 2 2 4 2	0.1271	3 1 2 2 4 2	0.0753	3 1 2 2 4 2	0.0765
	1 3 4 4 4 4	0.0938	1 3 4 4 4 4	0.0806	1 3 4 4 4 4	0.0941
			1 3 2 4 3 2	0.0538	1 3 2 4 3 2	0.0588
18a-3	2 2 1 1	0.4186	2 2 1 1	0.4032	2 2 1 1	0.4214
	4 3 3 3	0.2188	4 3 3 3	0.1935	4 3 3 3	0.1714
	2 3 1 1	0.2064	2 3 1 1	0.2204	2 3 1 1	0.1928
	4 3 1 3	0.1250	4 3 1 3	0.1559	4 3 1 3	0.1857

Table 9: Common haplotypes and their frequencies obtained by PedPhase, MRH and the EM method. In haplotypes, the alleles are encoded as 1=A, 2=C, 3=G, and 4=T.

solutions with 1 recombinant each and MRH found solutions with 0 recombinants. The complete results of PedPhase on this real dataset (and more datasets to be studied in the future) will be available at website `http://www.cs.ucr.edu/~jili/haplotyping.html`.

6 Concluding remarks

Pedigrees with mating loops are not a big problem for both rule-based algorithms block-extension and MRH, although they usually cause troubles for statistical methods. But, we have observed in the experiments that for both of the rule-based algorithms, the results on the pedigree structure with a loop (in Figure 9) are slightly worse than the results on the other two pedigree structures without loops. More investigation is needed on this issue. Notice that the pedigree structure used in the proof of the NP-hardness of the MRHC problem is very complicated and requires a large number of recombinants. An interesting question is if there is an efficient algorithm for solving MRHC (exactly) on pedigrees that require only a small (fixed) number of recombinants.

7 Acknowledgement

We thank the anonymous referee for a very thorough review and many helpful suggestions. 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 [8] with us.

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] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation*. Springer-Verlag, Berlin, 1999.

[3] C. D. A program to draw pedigrees using linkage or linksys data files. *Annals of Human Genetics*, 54:365–367, 1990.

[4] 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.

[5] K. Doi. MRHC2 and loopless MRHC are NP-hard. *Private Communication*, 2003.

[6] J. A. Douglas, M. Boehnke, E. Gillanders, J. M. Trent, and S. B. Gruber. Experimentally-derived haplotypes substantially increase the efficiency of linkage disequilibrium studies. *Nat Genet*, 28(4):361–4, 2001.

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

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

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

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

[11] J. C. Lam, K. Roeder, and B. Devlin. Haplotype fine mapping by evolutionary trees. *Am J Hum Genet*, 66(2):659–73, 2000.

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

[13] 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.

[14] J. S. Liu, C. Sabatti, J. Teng, B. J. Keats, and N. Risch. Bayesian analysis of haplotypes for linkage disequilibrium mapping. *Genome Res*, 11(10):1716–24, 2001.

[15] 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.

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

[17] J. R. O'Connell and D. E. Weeks. Pedcheck: a program for identification of genotype incompatibilities in linkage analysis. *Am J Hum Genet*, 63(1):259–66, 1998.

[18] J. R. O'Connell and D. E. Weeks. An optimal algorithm for automatic genotype elimination. *Am J Hum Genet*, 65(6):1733–40, 1999.

[19] A. Piccolbonii and D. Gusfield. On the complexity of fundamental computational problems in pedigree analysis. *Manuscript*, Submitted for publication.

[20] D. Qian, 2002. Personal communication.

[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. Devlin. Transmission/disequilibrium test meets measured haplotype analysis: family-based association analysis guided by evolution of haplotypes. *Am J Hum Genet*, 68(5):1250–63, 2001.

[23] S. K. Service, D. W. Lang, N. B. Freimer, and L. A. Sandkuijl. Linkage-disequilibrium mapping of disease genes by reconstruction of ancestral haplotypes in founder populations. *Am J Hum Genet*, 64(6):1728–38, 1999.

[24] A. H. Sherman. Algorithms for sparse gaussian elimination with partial pivoting. *ACM Transactions on Mathematical Software (TOMS)*, 4(4):330–338, 1978.

[25] 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.

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

[27] A. Thomas, A. Gutin, V. Abkevich, and A. Bansal. Multilocus linkage analysis by blocked gibbs sampling. *Stat Comput*, pages 259–269, 2000.

[28] H. T. Toivonen, P. Onkamo, K. Vasko, V. Ollikainen, P. Sevon, H. Mannila, M. Herr, and J. Kere. Data mining applied to linkage disequilibrium mapping. *Am J Hum Genet*, 67(1):133–45, 2000.

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

[30] S. Zhang, K. Zhang, J. Li, and H. Zhao. On a family-based haplotype pattern mining method for linkage disequilibrium mapping. *Pac Symp Biocomput*, pages 100–11, 2002.