Minimum Recombinant Haplotype Configuration on Tree Pedigrees (Extended Abstract)

Koichiro Doi[1], Jing Li[2], and Tao Jiang[2]

[1] Department of Computer Science
Graduate School of Information Science and Technology, University of Tokyo
7–3–1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
doi@is.s.u-tokyo.ac.jp
[2] Department of Computer Science and Engineering, University of California
Riverside, CA 92521, USA
{jili,jiang}@cs.ucr.edu

Abstract. We study the problem of reconstructing haplotype configurations from genotypes on pedigree data under the Mendelian law of inheritance and the minimum recombination principle, which is very important for the construction of haplotype maps and genetic linkage/association analysis. Li and Jiang [9, 10] recently proved that the *Minimum Recombinant Haplotype Configuration* (MRHC) problem is NP-hard, even if the number of marker loci is 2. However, the proof uses pedigrees that contain complex mating loop structures that are not common in practice. The complexity of MRHC in the loopless case was left as an open problem. In this paper, we show that loopless MRHC is NP-hard. We also present two dynamic programming algorithms that can be useful for solving loopless MRHC (and general MRHC) in practice. The first algorithm performs dynamic programming on the members of the input pedigree and is efficient when the number of marker loci is bounded by a small constant. It takes advantage of the tree structure in a loopless pedigree. The second algorithm performs dynamic programming on the marker loci and is efficient when the number of the members of the input pedigree is small. This algorithm also works for the general MRHC problem. We have implemented both algorithms and applied the first one to both simulated and real data. Our preliminary experiments demonstrate that the algorithm is often able to solve MRHC efficiently in practice.

Keywords: haplotype reconstruction, genotype, pedigree, NP-harness, dynamic programming algorithm, mating loop.

1 Introduction

In order to understand and study the genetic basis of complex diseases, the modeling of human variation is very important. *Single nucleotide polymorphisms* (SNPs), which are mutations at single nucleotide positions, are typical variations central to ongoing development of high-resolution maps of genetic variation and haplotypes for human [8]. While a dense SNP haplotype map is being built, various new methods [11, 14, 19, 23] have been developed to use haplotype information in linkage disequilibrium mapping. Some existing statistical methods for gene mapping have also shown increased power by incorporating SNP haplotype information [18, 25]. But, the use of haplotype maps has been limited due to the fact that the human genome is 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 [3] 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 can be divided into two categories: statistical methods and rule-based (*i.e. combinatorial*) methods. Both methods can be applied to population data and pedigree data. Statistical methods [4, 12, 15, 20, 22] often estimate the haplotype frequencies in addition to the haplotype configuration for each individual, but their algorithms are usually very time consuming and not suitable for large data sets. On the other hand, rule-based methods are usually very fast although they normally do not provide any numerical assessment of the reliability of their results. Nevertheless, by utilizing some reasonable biological assumptions, such as the minimum recombination principle, rule-based methods have proven to be powerful and practical[7, 13, 16, 17, 21, 24].

The minimum recombination principle states that the genetic recombinants are rare and thus haplotypes with fewer recombinants should be preferred in a haplotype reconstruction [7, 16, 17]. The principle is well supported by real data from the practice. For example, recently published experimental results [2, 5, 8] showed that the human genomic DNA can be partitioned into long *blocks* such that recombinants within each block are rare or even non-existent. Thus, within a single block, the true haplotype configuration is most likely one of the configurations with the minimum number of recombinants.

1.1 The minimum recombinant haplotype configuration problem and previous work

Qian and Beckman [17] formulated the problem of how to reconstruct haplotype configurations from genotype data on a pedigree under the Mendelian law of inheritance such that the resulting haplotype configurations require the minimum number of recombinants (*i.e.* recombination events). The problem is called *Minimum Recombinant Haplotype Configuration* (MRHC). They proposed a rule-based heuristic algorithm for MRHC that seems to work well on small pedigrees but is very slow for medium-sized pedigrees, especially in the case of biallelic genotype data.

Recently, Li and Jiang [9, 10] proved that MRHC is NP-hard, even if the number of marker loci is 2. They also devised an efficient (iterative) heuristic algorithm for MRHC, which is more efficient than the algorithm in [17] and can handle pedigrees of any practical sizes. However, their NP-hardness proof uses pedigrees with complex mating loop structures that may not ever rise in practice. The complexity of MRHC on tree pedigrees (*i.e.* loopless pedigrees, or pedigrees without mating loops) was left as an open question in [9, 10].

Table 1. Computational complexities of MRHC and loopless MRHC on pedigrees with n members, m marker loci, and at most m_0 heterozygous loci in each member.

	$m = 2$	m =constant	n =constant	no restriction
MRHC	NP-hard [9, 10]	NP-hard [9, 10]	$O(nm2^{4n})$	NP-hard [9, 10]
loopless MRHC	$O(n)$	$O(nm_0 2^{3m_0})$	$O(nm2^{4n})$	NP-hard

1.2 Our results

This paper is concerned with the MRHC problem on tree pedigrees (called *loopless MRHC*). First, we show that loopless MRHC is NP-hard, answering an open question in [9, 10]. Therefore, loopless MRHC does not have a polynomial time (exact) algorithm unless P = NP. The proof is based on a simple reduction from the well-known MAX CUT problem. Then we observe that the complexity of an instance of MRHC is defined by two independent parameters, namely, the number of members in the pedigree (*i.e.* the size of the pedigree) and the number of marker loci in each member, and we can construct a polynomial time algorithm for MRHC if one of these parameter is bounded by a constant (which is often the case in practice). This gives rises to two dynamic programming algorithms for loopless MRHC. The first algorithm assumes that the number of marker loci is bounded by a small constant and performs dynamic programming on the members of the input pedigree. The algorithm, called the *locus-based* algorithm, takes advantage of the tree structure of the input pedigree and has a running time linear in the size of the pedigree. Recall that the general MRHC problem is NP-hard even if the number of marker loci is 2 [9, 10]. This algorithm will be very useful for solving MRHC in practice because most real pedigrees are loopless and involve a small number of marker loci in each block. For example, the pedigrees studied in [5] are all loopless and usually contain four to six marker loci. The second algorithm assumes that the input pedigree is small and performs dynamic programming on the marker loci in each member of the pedigree simultaneously. The algorithm, called the *member-based* algorithm, works in fact for any input pedigree and has a running time linear in the number of marker loci. This algorithm will be useful as a subroutine for solving MRHC on small pedigrees, which could be nuclear families from a large input pedigree (*i.e.* components of the pedigree consisting of parents and children) or independent nuclear families from a (semi-)population data.

Table 1 exhibits the difference between the computational complexities of (general) MRHC and loopless MRHC. We have implemented both algorithms and applied the locus-based algorithm to both simulated and real datasets. Our preliminary experiments demonstrate that the algorithm is often able to solve MRHC efficiently in practice.

The rest of the paper is organized as follows. Some necessary definitions and notations used in MRHC are reviewed in Section 2. Section 3 presents the NP-hardness proof for loopless MRHC and Section 4 describes the two dynamic programming algorithms. The experimental results are summarized in Section 5.

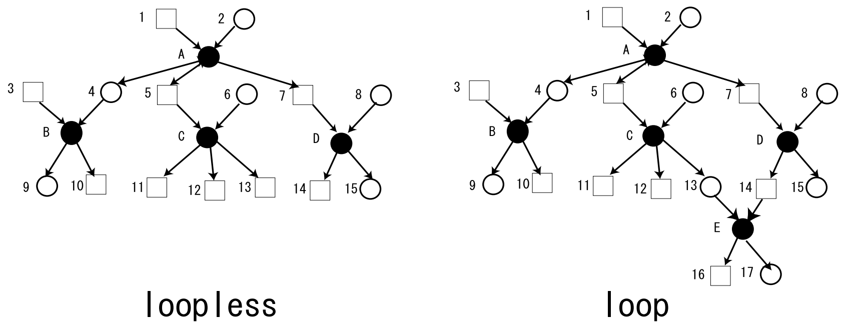

Fig. 1. Example pedigrees without mating loops and with mating loops. Here, a square box represents a male node, a circle represents female node, and a solid circle represents a mating node. The pedigree on the left has 15 members and 4 nuclear families.

2 Preliminaries

This section presents some concepts, definitions, and notations required for the MRHC problem. First, we define pedigrees.

Definition 1. *A pedigree is a connected directed acyclic graph $G = \{V, E\}$, 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 or members of the pedigree. The in-degree of each individual node is at most 1. The in-degree of a mating node must be 2, with one 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 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). 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. The individual nodes that have no parents are called the *founders*. A *mating loop* is a cycle in the pedigree graph when the directions of edges are ignored. A loopless pedigree is also called a *tree pedigree*. Fig. 1 shows two example pedigrees, one without mating loops and one with mating loops.

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. The position of a marker on a chromosome is called a *marker locus* and the state of a marker locus is 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.

In a *diploid* organism such as human, the status of two alleles at a particular marker locus of a pair of *homologous* 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 in each locus, that are on the same chromosome.

The Mendelian law of inheritance states that the genotype of a child must come from the genotypes of its parents at each maker 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 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.

Following [9, 10], 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 $-1, 0, 1$, 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 homozygous locus. A locus is *PS-resolved* if its PS value is 0 or 1. For convenience, we will use GS (*grand-parental source*) to indicate if an allele at a PS-resolved locus comes from a grand parental allele or a grand-material allele. Similar to a PS value, a GS value can also be -1, 0, or 1. An allele GS-resolved if its value is 0 or 1. A locus is GS-resolved if both of its alleles are GS-resolved. 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 if their GS values are 0 and 1.

Definition 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 of inheritance.*

Now, the MRHC problem [9, 10, 17, 21] can be defined precisely as follows:

Definition 3 (Minimum Recombinant Haplotype Configuration (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 the recombinants.*

It is known that MRHC is NP-hard even when the number of loci is 2 [9, 10]. In this paper, we are interested in the restricted case of MRHC on tree pedigrees like the left one in Fig. 1.

Definition 4. *The loopless MRHC problem is MRHC on pedigrees without mating loops.*

3 Loopless MRHC is NP-hard

In this section, we show the NP-hardness of the loopless MRHC problem, thus answering an open question in [9, 10]. The proof is via a reduction from the well-known MAX CUT problem.

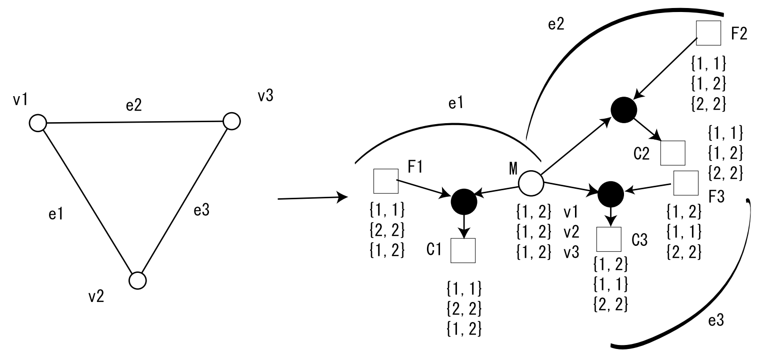

Fig. 2. An example reduction from MAX CUT to loopless MRHC.

Theorem 1. *Loopless MRHC problem is NP-hard.*

Proof. We give a polynomial time reduction from MAX CUT, which is known NP-hard [6]. Recall that an instance of MAX CUT is a graph $G = (V, E)$ and the objective is to partition V into two disjoint subsets V_1 and V_2 such that the number of edges of E that have one endpoint in V_1 and other endpoint in V_2 is maximized.

Assume that $V = \{v_1, v_2, \ldots, v_n\}, E = \{e_1, e_2, \ldots, e_m\}$, and the value of an optimal solution of MAX CUT on G is OPT. We construct a pedigree with n loci. First, we introduce an individual node M_0. The memberM_0 has n loci. Each locus corresponds to a vertex v_i with a heterozygous genotype $\{1, 2\}$. Observe that an assignment of PS values to the loci (in M_0) naturally divide the loci into two groups, which also bipartitions the corresponding vertices of G. Therefore, the PS values of the loci in M_0 encode a solution of the MAX CUT problem. We construct a nuclear family with mother M_0, father F_j, and child C_j for each edge $e_j = (v_{j1}, v_{j2})$, as shown in Fig. 2. The loci in F_j and C_j corresponding to v_{j1} are set to have a homozygous genotype $\{1, 1\}$, and the loci in F_j and C_j corresponding to v_{j2} are set to have a homozygous genotype $\{2, 2\}$. Other loci in F_j and C_j are all of heterozygous genotype $\{1, 2\}$.

Consider an assignment of PS values to the loci in M_0, which specifies a solution to the MAX CUT problem on G. Recall that the assignment partitions V into two groups. If v_{j1} and v_{j2} are in the same group (*i.e.* the loci in M_0 corresponding to v_{j1} and v_{j2} have the same PS value), the parents-child trio consisting of M_0, F_j, and C_j requires at least 1 recombinant, no matter how the PS values at other loci in M_0 and the PS values at the loci in F_j and C_j are assigned. Moreover, the parents-child trio can be made to require exactly 1 recombinant if the PS values at the loci in F_j and C_j are assigned appropriately On the other hand, if v_{j1} and v_{j2} are in different groups, the parents-child trio of M_0, F_j, and C_j requires no recombinants, no matter how the PS values at other loci in M_0 are assigned and if the PS values at the loci in F_j and C_j are assigned appropriately. Therefore, the number of the minimum recombinants required for the above instance of loopless MRHC is equal to $m - OPT$. Hence, MAX CUT reduces to loopless MRHC and loopless MRHC is NP-hard. □

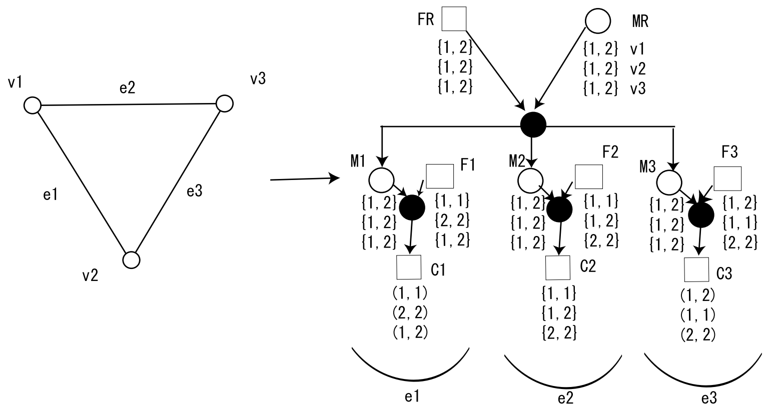

Fig. 3. An example reduction from MAX CUT to restricted loopless MRHC where each member of the pedigree has at most one mating partner.

In the above construction, the member M_0 has many mating partners. This does not happen very often in real datasets. We can convert the above construction to one where each member of the input pedigree has exactly one mating partner. This shows that loopless MRHC is still NP-hard if we further require that each member of the pedigree to have at most one mating partner.

Corollary 1. *Loopless MRHC is still NP-hard even if each member of the pedigree has at most one mating partner.*

Proof. This corollary can be proved by a simple modification of the proof of Theorem 1 as follows. We replace member M_0 by a nuclear family consisting father F_R, mother M_R, and female children $M_1, \ldots, M_m$, whose loci are of heterozygous genotype $\{1, 2\}$. We construct a nuclear family with mother M_j, father F_j, and child C_j for each edge $e_j = (v_{j1}, v_{j2})$. The loci in F_j and C_j are defined in the same way as in the proof of Theorem 1. Fig. 3 illustrates the new complete construction.

We would like to show that the minimum number of recombinants required for the above constructed instance of MRHC is also equal to $m - OPT$, and the PS values of the loci in M_R define an optimal solution of the MAX CUT problem. If the PS values at each locus in M_R, F_R, and $M_1, \ldots, M_m$ are the same, this would follow from the above proof. To show that the PS values at each locus in M_R, F_R, and $M_1, \ldots, M_m$ are the same, we first observe that we can always fix the PS value at the one of the loci (*e.g.* the first locus) in F_R to be the same as the PS value at the corresponding locus in M_R, because F_R is a founder. Moreover, we claim that the PS values at each of the remaining loci in M_R and F_R must be the same. Otherwise, there would be recombinants in each parents-offspring trio consisting of M_R, F_R, and M_j no matter how the PS values in M_j are assigned, for all j, and the total number of recombinants would be at least m. Similarly, we can assume, without loss of generality, that the PS value at the first locus in M_j is the same as the PS value at the first locus in M_R (because both parents M_R and F_R of M_j have the identical haplotype configurations), and claim that the PS value at each of the remaining loci in M_j must be the same as the PS value at the corresponding locus in M_R, in any optimal solution for the MRHC problem. Otherwise, an additional recombinant would occur in the trio M_R, F_R, and M_j. Hence, the PS values at each locus in M_R, F_R, and $M_1, \ldots, M_n$ are all identical, and the same arguments in the proof of Theorem 1 apply. $\square$

4 Two dynamic programming algorithms for loopless MRHC

Observe that the complexity of a loopless MRHC instance is defined by two independent parameters, namely, the number of loci and the number of members in the input pedigree. Although the problem of loopless MRHC is NP-hard, it is possible to solve it in polynomial time if we assume that one of the parameters is bounded from above by a constant. Here, we present two such dynamic programming algorithms. One assumes that the number of loci is bounded and is called the *locus-based* algorithm and the other assumes that the number of members is bounded and is called the *member-based* algorithm. As mentioned in Section 1, both algorithms are useful in practice.

4.1 The locus-based dynamic programming algorithm

The algorithm starts by converting the input tree pedigree into a rooted tree (at an arbitrary member). An example conversion is shown in Fig. 4. Then we traverse the tree in postorder and solve MRHC for the subtree rooted at each member when the member is required to have a certain haplotype configuration (*i.e.* PS assignment). A key step in the computation is the processing of a nuclear family. First, observe that if the PS values of all members in a nuclear family are known, we can easily compute the GS values of the children in the family to minimize the number of recombinants required in the nuclear family. Our second observation is that, once the PS values of the parents are fixed, the PS values for each child can be assigned independently. In other words, we can think of a nuclear family as a collection of (independent) parents-offspring trios and process the trios accordingly.

A more detailed description of the algorithm is given below. Let $numtrio(p, q, c)$ denote the minimum number of recombinants required for a parents-offspring trio consisting of a father, a mother, and a child with PS assignments p, q, and c, respectively.

Step 1: Root the pedigree at an arbitrary member R.

Step 2: Traverse the tree in postorder. For each individual node (*i.e.* a member) r and each PS assignment s at r, compute the minimum number of recombinants required in the subtree rooted at r. If r has more than 2 children (*i.e.* mating nodes), we do the above computation for each child mating node separately. Each child mating node of r defines a unique nuclear family, which may contain r as a parent or a child. Suppose that the nuclear family consists of father F, mother M, and children $C_1, \ldots, C_l$. We construct an array $num[node][ps]$, where $node$ denotes an individual node and ps denotes a PS assignment at the node.

If r is M (or F), then for each PS assignment s at r, we do the following.

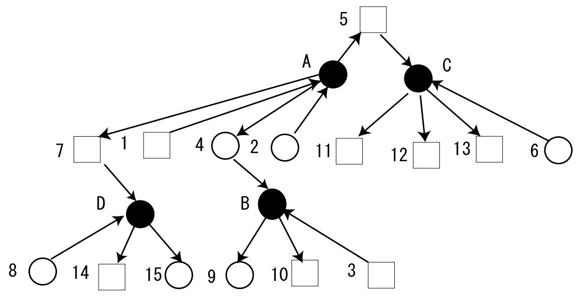

Fig. 4. A rooted tree for the tree pedigree in Fig. 1, where node 5 is selected as the root.

1. For each PS assignment p (or q) at F (or M, respectively), compute

$$num[r][s] = \min_{p} \sum_{1 \le i \le l} \min_{c}(num[F][p] + num[C_i][c] + numtrio(p, s, c))$$
$$(\text{or} = \min_{q} \sum_{1 \le i \le l} \min_{c}(num[M][m] + num[C_i][c] + numtrio(s, q, c)))$$

2. Keep pointers to the corresponding PS assignments at F (or M) and C_i $(1 \le i \le l)$ in an optimal solution.

On the other hand, if r is C_j for some j, then for each PS assignment s at r, we do the following.

1. For each PS assignments p and q at F and M, respectively, compute

$$num[r][s] = \min_{p,q}(numtrio(p, q, s) + \sum_{1 \le i \le l, i \ne j} \min_{c}(num[F][p]$$
$$+num[M][q] + num[C_i][c] + numtrio(p, q, c)))$$

2. Keep pointers to the corresponding PS assignments at F, M, and C_i $(1 \le i \le l)$ in an optimal solution.

Step 3: At root R, we compute the minimum number of recombinants required for the whole tree for any PS assignment at R, and the corresponding PS assignments at all members by backtracing the pointers.

The time and space complexities of this algorithm are given below.

Theorem 2. *Let n denote the number of members and m_0 the maximum number of heterozygous loci in any member. The above locus-based dynamic programming algorithm runs in $O(nm_0 2^{3m_0})$ time and $O(n2^{m_0})$ space.*

Proof. The rooted tree can be constructed in $O(n)$ time. In *Step 2*, we have to consider all combinations of PS assignments at the members of a parents-offspring trio, which is at most 2^{3m_0}. The computation of $numtrio$ is $O(m_0)$ for each trio and combination of PS assignments. Therefore, we can process each trio in $O(m_0 2^{3m_0})$ time. The number of parents-offspring trios is at most n. The running time of *Step 3* is $O(n)$. Therefore, the total time complexity of this algorithm is $O(nm_0 2^{3m_0})$.

For this computation, we need maintain the array num and pointers for backtracing. The size of num is clearly $O(n2^{m_0})$, which is also the number of pointer needed. Therefore, the total space complexity of this algorithm is $O(n2^{m_0})$. □

This result shows an interesting contrast to the general MRHC problem, which is still NP-hard when $m = 2$. The algorithm would be reasonably fast on a PC for $m_0 \leq 8$ (which means $m \leq 15$ in real datasets because many loci are often homozygous). In practice, the algorithm can be sped up by preprocessing the pedigree and fixing as many PS values as possible using the Mendelian law of inheritance. This could reduce the number of "free" heterozygous loci significantly. Moreover, if each nuclear family in the input pedigree has only one child, the algorithm can be made much faster.

Theorem 3. *If each nuclear family in the input pedigree has only one child, we can solve loopless MRHC in $O(nm_0 2^{2m_0})$ time and $O(n2^{m_0})$ space.*

Proof. We need only modify *Step 2* in the above algorithm. For parents-offspring trio, instead of considering all combinations of PS assignments at both the father and mother for each PS assignment at the child, we can consider each parent independently and optimize the number of recombinants required from the parent to the child. More precisely, if r is M (or F), we compute the minimum number of recombinants from F (or M) to C_1 for all PS assignments at F (or M) and C_1 and memorize the minimum number of recombinants for each PS assignment at C_1. Next, we compute the minimum number of recombinants in the parents-offspring trio for all PS assignments at M (or F) and C_1. Similarly, if r is C_1, we compute the minimum number of recombinants from M to C_1 for all PS assignments at M and C_1 and the minimum number of recombinants from F and C_1 for all PS assignments at F and C_1, separately. Then, we add up the corresponding numbers for each PS assignment at C_1. The running time of the modified algorithm is $O(m_0 2^{2m_0})$ for each parents-offspring trio and $O(nm_0 2^{2m_0})$ for the whole pedigree. □

4.2 The member-based dynamic programming algorithm

Many real pedigrees in practice are often of small or moderate sizes. For example, the dataset studied in [2] consists of a collection of parents-offspring trios and the dataset in [5] consists of pedigrees of sizes between 7 and 8. Here, we present an algorithm that is efficient when the size of the input pedigree is bounded from above by a small constant. The algorithm considers the PS and GS values at each locus across all members of the pedigree, and performs dynamic programming along the loci. A detailed description of the algorithm is given below. The algorithm in fact works for MRHC on general pedigrees.

We construct an array $num[i][t]$ that denotes the minimum number of recombinants required in the pedigree from locus 1 to locus i, if the PS/GS assignment at locus i in all members is t. Let $num1(s,t)$ denote the number of recombinants between any two adjacent loci with PS/GS assignment s and t in the whole pedigree. The algorithm works as follows. For $i = 1$ to $m - 1$,

1. Compute the minimum number of recombinants required in the pedigree from locus 1 to $i + 1$ for every possible PS/GS assignment t at locus $i + 1$, by considering all possible PS/GS assignment s at locus i, using the recurrence

$$num[i + 1][t] = \min_s (num[i][s] + num1(s, t))$$

2. Keep track of the PS/GS assignment s achieving the minimum in the above.

Finally, we find the minimum number of the recombinants from locus 1 to m and the corresponding PS/GS assignments at all loci by a standard backtracing procedure.

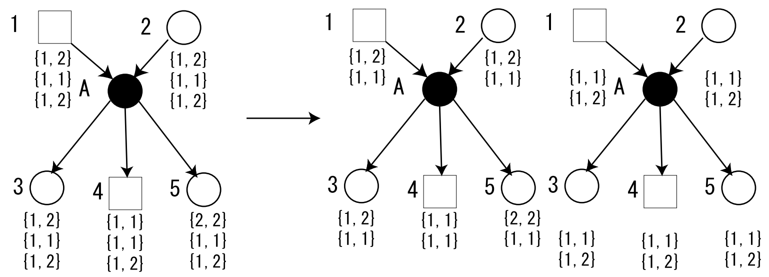

Fig. 5. An example pedigree with genotype information. The figure shows that the PS/GS assignments at locus 3 can be computed by considering all possible PS/GS assignments at locus 2 (independently from the PS/GS assignments at locus 1).

Theorem 4. *Let n denote the number of members in the pedigree and m the number of loci. The time complexity of the above member-based dynamic programming algorithm is $O(nm2^{4n})$. Its space complexity is $O(m2^{2n})$.*

Proof. The number of all possible PS/GS assignments at a single locus is trivially bounded 2^{3n}. However, this number includes many impossible assignments, *i.e.* assignments inconsistent with the Mendelian law of inheritance. It is easy to prove that the number of all possible PS/GS assignments at a locus is maximized if all genotypes at this locus are homozygous, such as locus 2 in Fig. 5. Therefore, the number of all possible (feasible) PS/GS assignments at a locus is $O(2^{2n})$, and the running time of the above algorithm is $O(mn2^{4n})$. The space for keeping the pointers used in backtracing is clearly $O(m2^{2n})$. □

In practice, the member-based algorithm runs reasonably fast when $n \leq 6$.

5 Experimental results

We have implemented the above two algorithms in C++. To evaluate the performance of our program, we compare the locus-based algorithm with PedPhase [10] and MRH v0.1 [17] on simulated genotype data. Two different tree pedigree structures (Fig. 1 and

Table 2. Running times of the locus-based algorithm, MRH and PedPhase on biallelic markers. The parameters in the first column are the number of members, number of marker loci, and number of recombinants.

Parameters	locus-based algorithm	MRH	PedPhase
(15,10,0)	6.4s	16s	1.3s
(15,10,4)	3.4s	3m13s	1.7s
(29,10,0)	13.6s	16m49s	1.8s
(29,10,4)	5.1s	14m2s	1.4s
(15,15,0)	35m15s	14m29s	1.3s
(15,15,4)	8m7s	10m9s	1.8s
(29,15,0)	15m10s	1h11m	1.9s
(29,15,4)	23m12s	1h15m	2.1s

Fig. 8 in [10]) were used in the simulation, one with 15 members and the other with 29 members. For each pedigree, genotypes with 10 and 15 biallelic marker loci were considered. The two alleles at each locus in a founder (*i.e.* a member that has no parents) were independently sampled with a fixed frequency of 0.5 in order to maximize the chance of heterozygosity (to test the worst-case performance of the locus-based algorithm), resulting in an average of 5 and 7.5 heterozygous loci, respectively, in each member of the pedigree. The number of recombinants used in generating each pedigree ranged from 0 to 4. For each combination of the above parameters, 100 sets of random genotype data were generated and the average performance of the programs was recorded, as shown in Table 2. The experiments were done on a Pentium IV PC with 1.7GHz CPU and 512MB RAM. In terms of the quality of solutions, the locus-based algorithm always output a haplotyping solution with the minimum number of recombinants, while PedPhase and MRH do not guarantee optimal solutions. When the data was generated with zero or few recombinants, the optimal solution was often unique and both PedPhase and MRH could recover it. However, the performance of PedPhase and MRH decreases when the number of recombinants increases. In terms of efficiency, PedPhase was the fastest among the three programs in all cases. But, surprisingly, the locus-based algorithm outperformed MRH in many cases.

We further compared the performance of the locus-based algorithm, PedPhase, MRH, and the EM algorithm used in [5] on a real dataset consisting of 12 multi-generation tree pedigrees, each with 7-8 members, from [5]. The comparison was based on the inference of *common* haplotypes (*i.e.* haplotypes with frequencies $> 5\%$). We focused on a randomly selected autosome (*i.e.* chromosome 3). There are 10 blocks (*i.e.* regions with few recombinants) in the chromosome 3 data, of which one block consists of 16 marker loci and all the others have only 4-6 marker loci each. The test results of the locus-based algorithm, MRH and the EM algorithm are summarized in Table 3. (We obtained the results of the EM algorithm directly form the authors [5]. The results of PedPhase are very similar to those of MRH's [10], and are thus not shown here due to space limit.) The results show that both rule-based haplotyping methods could discover almost all the common haplotypes that were inferred by the EM algorithm [5]. The haplotype frequencies estimated from the haplotype configuration results by simple counting are also similar to the estimations by the EM algorithm. Most of the blocks ($> 85\%$) were found

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

Block	EM		locus-based		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	0.1989	4 2 2 2 4	0.1802
	3 4 2 2 4	0.1432	3 4 2 2 4	0.1667	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	0.5000	4 1 1 2	0.4826
	2 3 3 4	0.2812	2 3 3 4	0.2634	2 3 3 4	0.2616
	2 3 3 2	0.1562	2 3 3 2	0.1398	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	0.2527	1 3 3 2 4 2	0.2473
	3 3 2 4 2 4	0.1354	3 3 2 4 2 4	0.0968	3 3 2 4 2 4	0.0914
	3 3 3 4 4 4	0.1021	3 3 3 4 4 4	0.1129	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	0.3065	2 3 2 4 2	0.2903
	3 3 4 2 4	0.3333	3 3 4 2 4	0.3118	3 3 4 2 4	0.3118
	3 3 4 4 2	0.1458	3 3 4 4 2	0.1505	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	0.4408	4 4 3 1	0.4167
	3 1 1 2	0.2813	3 1 1 2	0.2312	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	0.3656	3 4 4 1 2 4	0.4429
	2 3 2 4 3 2	0.3333	2 3 2 4 3 2	0.3065	2 3 2 4 3 2	0.2357
	3 4 2 4 2 4	0.2500	3 4 2 4 2 4	0.1935	3 4 2 4 2 4	0.1857
18a-1	1444231214144132	0.2697	1444231214144132	0.2688	1444231214144132	0.1706
	1444111214144132	0.2396	1444111214144132	0.2097	1444111214144132	0.2357
	1444131214144132	0.1887	1444131214144132	0.1989	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	0.4839	3 1 2 4 4 2	0.4765
	1 3 2 4 3 4	0.2604	1 3 2 4 3 4	0.1989	1 3 2 4 3 4	0.1765
	3 1 2 2 4 2	0.1271	3 1 2 2 4 2	0.0806	3 1 2 2 4 2	0.0765
	1 3 4 4 4 4	0.0938	1 3 4 4 4 4	0.0860	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	0.4140	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	0.2097	2 3 1 1	0.1928
	4 3 1 3	0.1250	4 3 1 3	0.1613	4 3 1 3	0.1857

by the locus-based algorithm and MRH to have involved 0 recombinants. The speeds of all three programs on this real dataset were very fast (less 1 minute), even though one of the blocks has 16 marker loci.

6 Concluding remarks

Missing data usually occur in real data set due to many reasons. Unfortunately, Aceto *et al.* [1] recently showed that checking consistency (thus imputing missing data) for pedigree is in general NP-hard. The two algorithms above take data with no missing values as input. In our experiment on the real data set, we had to impute the missing values first before feeding the data to the locus-based algorithm. Our current imputation algorithm simply uses the Mendelian law and allele frequencies. It would be desirable to combine missing data imputation and haplotype inference in a unified framework.

Acknowledgements

KD's work was partially supported by a Grant-in-Aid for Scientific Research on Priority Areas (C) for "Genome Information Science" from the Ministry of Education, Culture, Sports, Science and Technology, Japan. JL is supported by NSF grant CCR-9988353. TJ is supported by NSF Grants CCR-9988353, ITR-0085910, DBI-0133265, and National Key Project for Basic Research (973).

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. M. Daly, J. Rioux, S. Schaffner, T. Hudson, and E. Lander. High-resolution haplotype structure in the human genome. *Nat Genet*, 29(2):229–232, 2001.
3. J. A. Douglas, M. Boehnke, E. Gillanders, J. Trent, and S. Gruber. Experimentally-derived haplotypes substantially increase the efficiency of linkage disequilibrium studies. *Nat Genet*, 28(4):361–364, 2001.
4. L. Excoffier and M. Slatkin. Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. *Mol Biol Evol*, 12:921–927, 1995.
5. S. B. Gabriel, *et al.* The structure of haplotype blocks in the human genome. *Science*, 296(5576):2225–29, 2002.
6. M. R. Gary, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1, 237–267, 1976.
7. D. Gusfield. Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. *Proc. RECOMB*, 166–175, 2002.
8. L. Helmuth. Genome research: Map of the human genome 3.0 *Science*, 293(5530):583–585, 2001.
9. J. Li and T. Jiang, Efficient rule-based haplotyping algorithms for pedigree data. *Proc. RECOMB'03*, pages 197–206, 2003.
10. J. Li and T. Jiang, Efficient inference of haplotypes from genotypes on a pedigree. *J. Bioinfo. and Comp. Biol.* 1(1):41-69, 2003.
11. J. C. Lam, K. Roeder, and B. Devlin. Haplotype fine mapping by evolutionary trees. *Am J Hum Genet*, 66(2):659–673, 2000.
12. S. Lin and T. P. Speed. An algorithm for haplotype analysis. J Comput Biol, 4(4):535–546, 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 haplotyping interface 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. D. Qian and L. Beckman. Minimum-recombinant haplotyping in pedigrees. *Am J Hum Genet*, 70(6):1434–1445, 2002.
18. H. Seltman, K. Roeder, and B. Delvin. 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.

19. 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):1728-1738, 1999.
20. M. Stephens, N. J. Smith, and P. Donnelly. A new statistical method for haplotype reconstruction from population data. *Am J Hum Genet*, 68(4):978-989, 2001.
21. P. Tapadar, S. Ghosh, and P. P. Majumder. Haplotyping in pedigrees via a genetic algorithm. *Hum Hered*, 50(1):43–56, 2000.
22. A. Thomas, A. Gutin, V. Abkevich, and A. Bansal. Multilocus linkage analysis by blocked gibbs sampling. *Stat Comput*, 259–269, 2000.
23. 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):133–145, 2000.
24. E. M. Wijsman. A deductive method of haplotype analysis in pedigrees. *Am J Hum genet*, 41(3):356–373, 1987.
25. S. Zhang, K. Zhang, J. Li and H. Zhao. On a family-based haplotype pattern mining method for linkage disequilibrium mapping. *Pac Symp Biocomput*, 100–111, 2002.