Algorithmica (2012) 62:951–981
DOI 10.1007/s00453-011-9494-5

An Efficient Algorithm for Haplotype Inference on Pedigrees with a Small Number of Recombinants

Jing Xiao · Tiancheng Lou · Tao Jiang

Received: 14 February 2010 / Accepted: 24 January 2011 / Published online: 9 February 2011

Abstract Combinatorial (or rule-based) methods for inferring haplotypes from genotypes on a pedigree have been studied extensively in the recent literature. These methods generally try to reconstruct the haplotypes of each individual so that the total number of recombinants is minimized in the pedigree. The problem is NP-hard, although it is known that the number of recombinants in a practical dataset is usually very small. In this paper, we consider the question of how to efficiently infer haplotypes on a large pedigree when the number of recombinants is bounded by a small constant, i.e. the so called k-recombinant haplotype configuration (k-RHC) problem. We introduce a simple probabilistic model for k-RHC where the prior haplotype probability of a founder and the haplotype transmission probability from a parent to a child are all assumed to follow the uniform distribution and k random recombination events are assumed to have taken place uniformly and independently in the pedigree. We present an $O(mn\log^{k+1} n)$ time algorithm for k-RHC on tree pedigrees without mating loops, where m is the number of loci and n is the size of the input pedigree, and prove that when $90\log n < m < n^3$, the algorithm can correctly find a feasible haplotype configuration that obeys the Mendelian law of inheritance and requires no more than k recombinants with probability $1 - O(k^2\frac{\log^2 n}{mn} + \frac{1}{n^2})$. The algorithm is efficient when k is of a moderate value and could thus be used to infer haplotypes from

J. Xiao (✉)
IBM China Research Lab, Beijing, China
e-mail: xiaojing82@gmail.com

T. Lou · T. Jiang
Department of Computer Science and Technology, Tsinghua University, Beijing, China

T. Lou
e-mail: loutiancheng860214@gmail.com

T. Jiang
Department of Computer Science and Engineering, University of California, Riverside, CA, USA
e-mail: jiang@cs.ucr.edu

genotypes on large tree pedigrees efficiently in practice. We have implemented the algorithm as a C++ program named TREE-k-RHC. The implementation incorporates several ideas for dealing with missing data and data with a large number of recombinants effectively. Our experimental results on both simulated and real datasets show that TREE-k-RHC can reconstruct haplotypes with a high accuracy and is much faster than the best combinatorial method in the literature.

Keywords Computational biology · Haplotype inference · Pedigree · Recombination · Combinatorial algorithm · Probabilistic model

1 Introduction

As more progress has been made in science and technology, scientists believe that genetic factors should play a significant role in preventing, diagnosing and treating important human diseases such as diabetes, cancer, stroke, heart disease, depression, and asthma. With the discovery of genetic markers such as microsatellite DNA sequences and single nucleotide polymorphisms (SNPs), it is now possible to provide a unique genetic map to establish connections between diseases and specific genetic variations. One of the main objectives of the International HapMap Project launched in October 2002 [27] is to discover the haplotype structure of human beings and examine the common haplotypes in different populations. This information will greatly facilitate the mapping of many important disease-susceptibility genes. However, the diploid structure of humans makes it very expensive to collect haplotype data directly. Instead, genotype data are collected routinely. Since haplotype data are required (or at least desirable) in many genetic analysis including linkage disequilibrium analysis and disease association mapping, efficient and accurate computational methods for the inference of haplotypes from genotypes, which is also commonly referred to as *phasing*, have been extensively studied in the literature. A recent survey on these methods can be found in [19].

The existing haplotyping algorithms can be classified as either statistical or combinatorial (or rule-based). Both paradigms can be applied to *pedigree* data, *population* data, or *pooled samples*. In this paper, we are interested in pedigree data and the combinatorial paradigm. Although many (statistical or combinatorial) algorithms have been proposed for haplotype inference on pedigrees in the literature [19], they are mostly effective for pedigrees of small to moderate sizes. For example, it took the exact algorithm based on *integer linear programming* (ILP) in PedPhase 5 hours to solve a pedigree with 29 individuals and 51 SNP loci [17, 18] on a standard PC. The same data took the popular program SimWalk2 [26] based on a statistical approach 6 days. The well-known Lander-Green algorithm [15] based on the *maximum likelihood* (ML) framework and its subsequent improvements [1, 12, 14] run in time linear in the number of loci but exponential in the pedigree size [2, 19]. These algorithms are thus limited to relatively small pedigrees.

With the advance in sequencing technology, larger and larger pedigrees are being genotyped and scientists are becoming increasingly interested in haplotype inference on large pedigrees. For example, in [2, 4], the inference was performed on pedigrees of sizes 368 and 1149, respectively. The existing haplotype inference methods either are very slow (e.g. those based on ML or ILP) or have less than desirable accuracies (e.g. the block extension heuristic algorithm in PedPhase) when the input pedigree gets large. In fact, the question of how to efficiently and accurately infer haplotypes from genotypes on large pedigrees is one of the challenges raised at the recently held 2008 Haplotype Conference [13].

In general, combinatorial methods for haplotype inference are faster (or intended to be faster) than statistical methods that attempt to maximize the likelihood of the haplotype solution [19]. To our knowledge, all combinatorial algorithms for haplotyping pedigree data aim at solving the *minimum-recombinant haplotype configuration* (MRHC) problem [16–18, 25] where the goal is to find a haplotype solution requiring the minimum number of recombinants (i.e. recombination events). The problem is sensible since it is known that recombinants are rare in a typical human pedigree [11]. This is especially true when the marker loci considered are from a same haplotype block. For instance, the analysis performed in [17, 18] on a HapMap data shows that the average number of recombinants per haplotype block of each chromosome on a (relatively small) pedigree is in fact close to 0 (although not exactly 0). Thus, a minimum-recombinant solution is likely to be the true solution. Unfortunately, MRHC is NP-hard [16]. It remains NP-hard even if the input pedigree is a tree without mating loops [7, 22]. The ILP-based exact algorithm for MRHC in PedPhase [19] works well for small pedigrees but its worst case running time is exponential in both the number of loci and pedigree size. The heuristic algorithm in [25] runs for days on a PC even for medium-sized datasets. Hence, recent work on MRHC has been focused on the special case where the number of recombinants is zero, the so called *zero-recombinant haplotype configuration* (ZRHC) problem [5, 16, 20, 21, 23, 28, 30, 31]. In particular, Li and Jiang [16] formulated ZRHC as a system of linear equations over the field $F(2)$ and devised an $O(m^3n^3)$ time algorithm using Gaussian elimination, where m is the number of loci and n is the size of the input pedigree. Xiao et al. [30, 31] improved the running time to $O(mn^2 + n^3 \log^2 n \log\log n)$ by using a compact system of linear equations, taking advantage of some special properties of a pedigree graph, and the low-stretch spanning tree technique. The recent results in [5, 20, 21] presented linear (i.e. $O(mn)$) time algorithms for ZRHC on tree pedigrees using different techniques. Note that tree pedigrees are very common in human pedigrees [2]. They also play important roles in the analysis of general complex pedigrees [3, 29].

Since the number of recombinants in a real pedigree studied in a typical haplotyping instance (e.g. when a single haplotype block is considered) is usually very small, a plausible approach to solving MRHC that could potentially be very efficient in practice is to try to infer a haplotype configuration that requires at most k recombinants in the input pedigree, where k is some fixed small constant. We will refer to this parameterized problem as the *k-recombinant haplotype configuration* (k-RHC) problem. Although ZRHC (or 0-RHC) seems easy to solve [5, 20, 21, 30, 31], the general k-RHC problem remains very hard to tackle. Observe that, we could obtain a trivial algorithm for k-RHC with time complexity $O((mn)^k(mn^2 + n^3 \log^2 n \log\log n))$ by using the algorithm in [30, 31] for ZRHC and exhaustively enumerating the possible locations of the k recombinants. This is because the k-RHC instance can be easily transformed into a ZRHC instance once the recombinant locations are known. Similarly, one could obtain a trivial algorithm for k-RHC on tree pedigrees with time complexity $O((mn)^{k+1})$ by using the linear time algorithms in [5, 20, 21] for tree ZRHC. We note in passing that the dynamic programming algorithm in [6] solves MRHC on tree pedigrees in $O(nm^{3k+1}2^m)$ time when each parent-child pair is allowed to have at most k recombinants. This algorithm is inefficient when the number of loci exceeds 30 even if k is very small.

In this paper, we present an algorithm for k-RHC that is efficient in the average sense. More precisely, we consider a simple probabilistic model for k-RHC where the haplotypes of the founders (i.e. individuals without parents in the input pedigree) are generated randomly from a uniform distribution, a uniform random haplotype of each parent is passed to a child, and k uniform random recombinants are assumed to have taken place independently in the pedigree. This model is a special case of the general probabilistic model in the genetics literature (see e.g. [24]) where the prior founder haplotype probabilities and haplotype transmission probabilities could follow arbitrary distributions. In other words, our model is a primitive Mendelian model. We present an $O(mn\log^{k+1} n)$ time algorithm for k-RHC on tree pedigrees, and prove that when $90\log n < m < n^3$, the algorithm can correctly find a feasible haplotype configuration that obeys the Mendelian law of inheritance and requires no more than k recombinants with probability $1 - O(k^2\frac{\log^2 n}{mn} + \frac{1}{n^2})$. (Note that this result does not imply that k-RHC is fixed-parameter tractable as defined in [8].) The algorithm is fast when k is of a moderate value and could thus be used to infer haplotypes from genotypes on large tree pedigrees in practice. We have implemented the algorithm as a C++ program named TREE-k-RHC. The implementation incorporates several effective ideas for dealing with missing data and data with an unexpectedly large number of recombinants. Our preliminary experimental results on both simulated and real datasets show that TREE-k-RHC can reconstruct haplotypes with a high accuracy and speed. In fact, it runs more than 20 times faster than the ILP-based exact algorithm in PedPhase [17, 18] and is more accurate than the heuristic algorithm in PedPhase [16]. We expect that the speed up will grow quickly with m and n as the worst-case time complexity of the ILP-based algorithm is at least $(mn)^k$.

The crux of our algorithm is to formulate k-RHC as an ILP based on the system of linear equations developed in [30, 31] (also in [21]). For each instance generated by the probabilistic model, we try to identify small areas of the pedigree where a recombinant might have occurred by comparing the linear (equality) constraints in the ILP. Once the locations of all k recombinants are determined (or enumerated), the instance is transformed to a tree ZRHC instance and solved in linear time by using one of the algorithms in [5, 20, 21].

The rest of the paper is organized as follows. In Sect. 2, we present an ILP formulation of k-RHC based on the system of linear equations introduced in [30, 31]. Section 3 reviews some graph data structures and constraint generation techniques from [30, 31] that can be used to make the ILP more compact. We present the efficient algorithm for k-RHC on tree pedigrees and analyze its success probability in Sect. 4. In Sect. 5, we describe the implementation of this algorithm. The experimental results are discussed in Sect. 6.

2 An Integer Linear Program for k-RHC

In this section, we formulate k-RHC as an ILP based on the system of linear equations in [30, 31] for solving ZRHC. All the definitions are the same as in [30, 31] except for the definition of the h-variables. Throughout this paper, n denotes the number of the individuals (or members) in the input pedigree and m the number of marker loci. Without loss of generality, suppose that each allele in the given genotypes is numbered numerically as 1 or 2 (i.e. the markers are assumed to be *bi-allelic*, which makes the hardest case for MRHC [16]), and the pedigree is free of genotype errors (i.e. the two alleles at each locus of a child can always be obtained from its respective parents). Hence, we can represent the genotype of member j as a ternary vector $\vec{g}_j$ as follows: $g_j[i]=0$ if locus i of member j is homozygous with both alleles being 1's, $g_j[i]=1$ if the locus is homozygous with both alleles being 2's, and $g_j[i]=2$ otherwise (i.e. the locus is heterozygous). For any heterozygous locus i of member j, we use a binary variable $p_j[i]$ to denote the *phase* at the locus as follows: $p_j[i]=0$ if allele 1 is paternal, and $p_j[i]=1$ otherwise. When the locus is homozygous, the variable is set to $g_j[i]$ for some technical reasons (so that the equations below involving $p_j[i]$ will hold). Hence, the vector $\vec{p}_j$ describes the paternal and maternal haplotypes of member j. Observe that the vectors $\vec{p}_1,\dots,\vec{p}_n$ represent a complete haplotype configuration of the pedigree. Also, for technical reasons, define a vector $\vec{w}_j$ for each member j such that $w_j[i]=0$ if its i-th locus is homozygous and $w_j[i]=1$ otherwise.

Suppose that member j_r is a parent of member j. We introduce an auxiliary binary variable $h_{j_r,j}[i]$ to indicate which allele of j_r is passed to j at locus i. If j_r gives its paternal allele to j at locus i, then $h_{j_r,j}[i]=0$; otherwise $h_{j_r,j}[i]=1$. Suppose that j is a non-founder member with its father and mother being j_1 and j_2, respectively. We can define two linear (constraint) equations over the field $F(2)$ to describe the inheritance of paternal and maternal haplotypes at j on locus i respectively:

$$\begin{cases} p_{j_1}[i]+h_{j_1,j}[i]\cdot w_{j_1}[i]=p_j[i] \\ p_{j_2}[i]+h_{j_2,j}[i]\cdot w_{j_2}[i]=p_j[i]+w_j[i]. \end{cases} \tag{1}$$

Denoting $\vec{d}_{j_1,j}=\vec{0}$ and $\vec{d}_{j_2,j}=\vec{w}_j$, the above equations can be unified into a single equation as:

$$\vec{p}_{j_r}[i]+h_{j_r,j}[i]\cdot \vec{w}_{j_r}[i]=\vec{p}_j[i]+\vec{d}_{j_r,j}[i]\quad (r=1,2). \tag{2}$$

If there are no recombinants in the pedigree, $h_{j_r,j}[i]=c$ (which is some constant) for all i. Conversely, if $h_{j_r,j}[i]\neq h_{j_r,j}[i+1]$, there must be a recombinant from member j_r to member j between locus i and locus $i+1$. Formally, we can express the k-RHC problem as an ILP:

$$\sum_{\text{for all parent-child pairs } (j_r, j)} \sum_{i=1}^{m-1} \left| h_{j_r,j}[i] - h_{j_r,j}[i+1] \right| \le k$$

$$\begin{cases} p_l[i] + h_{l,j}[i] \cdot w_l[i] = p_j[i] + d_{l,j}[i] & 1 \le i \le m,\ 1 \le j,\ l \le n, \\ & \quad l \text{ is a parent of } j \\ p_j[i] = g_j[i] & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] \ne 2 \\ w_j[i] = 1 & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] = 2 \\ w_j[i] = 0 & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] \ne 2 \\ d_{l,j}[i] = w_j[i] & 1 \le i \le m,\ 1 \le j, l \le n, \\ & \quad l \text{ is the mother of } j \\ d_{l,j}[i] = 0 & 1 \le i \le m,\ 1 \le j, l \le n, \\ & \quad l \text{ is the father of } j \end{cases} \tag{3}$$

where $g_j[i], w_j[i], d_{l,j}[i]$ are all constants depending on the input genotypes, and $p_j[i], h_{l,j}[i]$ are the unknowns. Again, the equality constraints are defined over $F(2)$ whereas the (only) inequality constraint is defined over all integers. Note that, the number of p-variables is exactly mn and the number of h-variables is at most $2mn$. This ILP is different from the ILP for MRHC used in [17, 18] which is not based on the system of linear equations. Observe that for any member j, if j or any of its parents are homozygous at locus i, then $p_j[i]$ is fixed based on (3). Such p-variables are called *pre-determined*.

3 Some Graph Structures and a Compact ILP in *h*-Variables

As in [30, 31], the above ILP can be transformed to one concerning only the h-variables. This is not surprising because the h-variables completely describes the inheritance relationship in the pedigree, including the locations of recombinants. In this section, we review some useful graph structures and the generation of a sufficient set of equality constraints on h-variables introduced in [30, 31]. Again, all the definitions are the same as in [30, 31] except for the definition of the h-variables.

3.1 The Pedigree Graph and Locus Graphs

In [30, 31], the input pedigree is transformed into a *pedigree graph* by connecting each parent directly to his children. See Fig. 1(B) for an example. Although the edges in the pedigree graph representing the inheritance relationship between a parent and a child are directed, we consider them as undirected when dealing with linear constraints. Thus, these edges will sometimes be thought of as directed but other times as undirected according to the context.

Clearly, such a pedigree graph $G = (V, E)$ may be cyclic due to mating loops or multiple children shared by a pair of parents. Let $\mathcal{T}(G)$ be any spanning tree on G.

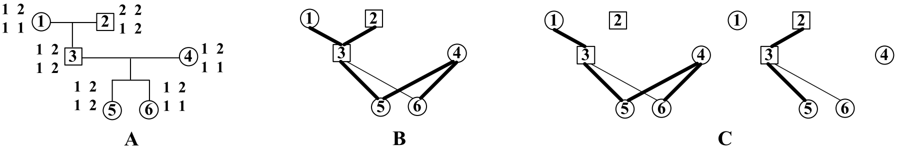


Fig. 1 **A** An example pedigree with genotype data. Here, the alleles at a locus are ordered according to their id numbers instead of phase (which is unknown). **B** The pedigree graph with a spanning tree. The tree edges are highlighted. Observe that the pedigree graph has a cycle of length 4 although the given pedigree is a tree. **C** The locus graphs. The *left graph* is for the first locus, which has a cycle, while the *right graph* is for the second locus. The locus forests are highlighted

$\mathcal{T}(G)$ partitions the edge set E into two subsets: the *tree edges* and the *non-tree edges* (or *cross edges*). Let $E^{\mathbb{X}}$ denote the set of cross edges. Since $|E| \leq 2n$ and the number of edges in $\mathcal{T}(G)$ is $n-1$, we have $|E^{\mathbb{X}}| \leq n+1$. Figure 1(B) gives an example of the tree edges and the cross edges.

For any fixed locus i, the value $w_l[i]$ can be viewed as the weight of each edge $(l, j) \in E$, where l is a parent of j. We construct the *i-th locus graph G_i* as the subgraph of G induced by the edges with weight 1. Formally, $G_i = (V, E_i)$, where $E_i = \{(l, j) | \, l \text{ is a parent of } j, w_l[i] = 1\}$. The i-th locus graph G_i induces a subgraph of the spanning tree $\mathcal{T}(G)$. Since the subgraph is a forest, it will be referred to as the *i-th locus forest* and denoted by $\mathcal{T}(G_i)$. Figure 1(C) shows the locus graphs and the locus forests of the given pedigree. The locus graphs can be used to identify some implicit constraints on the h-variables as follows. First, for any edge $(l, j) \in E$, define $h_{l,j}[i] = h_{j,l}[i]$ and $\vec{d}_{l,j} = \vec{d}_{j,l}$.

Lemma 1 [30, 31] *For any path $P = j_0, \ldots, j_t$ in locus graph G_i connecting vertices j_0 and j_t, we have*

$$p_{j_0}[i] + p_{j_t}[i] + \sum_{r=0}^{t-1} \bigl(h_{j_r, j_{r+1}}[i] + d_{j_r, j_{r+1}}[i]\bigr) = 0. \tag{4}$$

Corollary 2 [30, 31] *For any cycle $C = j_0, \ldots, j_t, j_0$ in G_i, there exists a binary constant b defined as $b = \sum_{r=0}^{t} d_{j_r, j_{r+1 \bmod t+1}}[i]$ such that $\sum_{r=0}^{t} h_{j_r, j_{r+1 \bmod t+1}}[i] = b$.*

Corollary 3 [30, 31] *Suppose that $P = j_0, \ldots, j_t$ is a path in G_i connecting vertices j_0 and j_t, and the variables $p_{j_0}[i]$ and $p_{j_t}[i]$ are pre-determined. There exists a binary constant b defined as $b = p_{j_0}[i] + p_{j_t}[i] + \sum_{r=0}^{t-1} d_{j_r, j_{r+1}}[i]$ such that $\sum_{r=0}^{t-1} h_{j_r, j_{r+1}}[i] = b$.*

3.2 Linear Equality Constraints on the h-Variables

As in [30, 31], we generate a sufficient set of linear equality constraints on the h-variables by considering each edge in a locus graph. Such a set of constraints will guarantee a feasible solution to the ILP in (3). Note that since the edges broken (i.e.

deleted) in a locus graph involve pre-determined p-variables, we do not have to introduce constraints to cover them. The constraints can be classified into two categories with respect to the spanning tree $\mathcal{T}(G)$: constraints for cross edges and constraints for tree edges.

Cross Edge Constraints Adding a cross edge e to the spanning tree $\mathcal{T}(G)$ yields a cycle C in the pedigree graph G. Suppose the edge e exists in the i-th locus graph G_i, and consider two cases of the cycle C with respect to G_i.

Case 1: The cycle exists in G_i. We introduce a constraint along the cycle as in Corollary 2. This constraint is called a *cycle constraint*. The set of such cycle constraints for edge e in all locus graphs is denoted by $C^{\mathbb{C}}(e)$, i.e.

$$C^{\mathbb{C}}(e) = \{(b, e) \mid b \text{ is associated with the cycle in } \mathcal{T}(G_i) \cup \{e\},\ 1 \le i \le m\}.$$

The set of cycle constraints for all cross edges is denoted by $C^{\mathbb{C}} = \biguplus_{e \in E^{\mathbb{X}}} C^{\mathbb{C}}(e)$.

Case 2: Some of the edges of the cycle do not exist in G_i. This means that the cycle C is broken into several disjoint paths in G_i by the pre-determined vertices. Since e exists in G_i, exactly one of these paths, denoted as P, contains e. Observe that both endpoints of P are pre-determined and thus Corollary 3 could give us a constraint concerning the h-variables along the path. Such a constraint will be called a *path constraint*. The set of such path constraints for e in all locus graphs G_i is denoted by $C^{\mathbb{P}}(e)$, i.e.

$$C^{\mathbb{P}}(e) = \left\{ (l, j, b, e) \,\middle|\, \begin{array}{l} \text{in } \mathcal{T}(G_i) \cup \{e\}, b \text{ is associated with the path containing } e \\ \text{connecting two pre-determined vertices } l \text{ and } j,\ 1 \le i \le m \end{array} \right\}.$$

The set of path constraints for all cross edges is denoted by $C^{\mathbb{P}} = \biguplus_{e \in E^{\mathbb{X}}} C^{\mathbb{P}}(e)$.

Tree Edge Constraints By Corollary 3, there is an implicit constraint concerning the h-variables along each path between two pre-determined vertices in the same connected component of $\mathcal{T}(G_i)$. Therefore, for each connected component $\mathcal{T}$ of $\mathcal{T}(G_i)$, we arbitrarily pick a pre-determined vertex in the component as the *seed* vertex, and generate a constraint for the unique path in $\mathcal{T}(G_i)$ between the seed and each of the other pre-determined vertices in the component, as in Corollary 3. Such a constraint will be called a *tree constraint*.

To conform with the notation of path constraints and for the convenience of presentation, we arbitrarily pick a tree edge denoted as e_0, and write the set of tree constraints at all loci as

$$C^{\mathbb{T}} = \left\{ (l, j, b, e_0) \,\middle|\, \begin{array}{l} \text{in a connected component of } \mathcal{T}(G_i) \text{ with seed } l, b \text{ is} \\ \text{associated with the path connecting vertices } l \text{ and a} \\ \text{predetermined vertex } j,\ 1 \le i \le m \end{array} \right\}.$$

Define $\mathcal{C} = C^{\mathbb{C}} \cup C^{\mathbb{P}} \cup C^{\mathbb{T}}$. The subset of all the constraints in $\mathcal{C}$ generated in locus graph G_i will be denoted as $\mathcal{C}_i$. The next two lemmas are easy to prove.

Lemma 4 [30, 31] $|C| = |C^{\mathbb{C}}| + |C^{\mathbb{P}}| + |C^{\mathbb{T}}| = O(mn)$.

Lemma 5 *None of the constraints in* $C^{\mathbb{P}} \cup C^{\mathbb{T}}$ *are defined on a path that begins or ends at a founder.*

Proof A constraint may only begin or end at a pre-determined vertex. A founder is pre-determined at locus i if and only if it is homozygous at i. When it is homozygous, all its children are pre-determined at locus i and thus disconnected from the founder in locus graph G_i. In other words, the founder would be an isolated vertex in G_i and not involved in any constraint in $\mathcal{C}_i$. □

As in [30, 31], we can prove that $\mathcal{C}$ forms a sufficient set of constraints, i.e. any solution in terms of the h-variables satisfying all these constraints would imply a feasible solution in terms of both the h- and p-variables satisfying (3). The proof is very similar to the corresponding proof in [30, 31] and therefore omitted. The following lemma hence follows.

Lemma 6 *The k-RHC problem can be expressed as the following ILP*:

$$\sum_{\substack{\textit{for all edges } (j_r, j) \\ \textit{plus all the linear} \\ \textit{equality constraints in } \mathcal{C}}} \sum_{i=1}^{m-1} \left| h_{j_r,j}[i] - h_{j_r,j}[i+1] \right| \le k. \tag{5}$$

4 An $O(mn \log^{k+1} n)$ Time Algorithm for k-RHC on Tree Pedigrees

As mentioned before, the basic idea of our algorithm is to locate all the k recombinants first. Once we know the locations of all the recombinants, we can define the relationship between h-variables at consecutive loci corresponding to the same edge in the pedigree graph. For example, if there is a recombinant between locus i and locus $i+1$ on edge (u, v), $h_{u,v}[i] = h_{u,v}[i+1] + 1$. If such a recombinant does not exist, $h_{u,v}[i] = h_{u,v}[i+1]$. In this way, all the h-variables at different loci corresponding to the same edge can be represented by a single h-variable in the ILP of (5), and the k-RHC ILP is effectively reduced to a ZRHC instance which can be solved by the linear-time algorithm in [21]. Hence, the challenge here is how to locate the recombinants without exhaustively enumerating all the possibilities in the entire pedigree. The key idea is that we compare the constraints of $\mathcal{C}$ (as well as some additional constraints involving one or two h-variables to be constructed in the next two subsections) at different loci to see if they imply the necessity of a recombinants. For example, suppose that we have a constraint along path $P = j_0, \dots, j_t$ at locus i and another constraint along the same path P at locus l ($l > i$). By Corollary 3, we have $\sum_{r=0}^{t-1} h_{j_r,j_{r+1}}[i] = b_i$ and $\sum_{r=0}^{t-1} h_{j_r,j_{r+1}}[l] = b_l$. If $b_i \neq b_l$, there is at least one pair of h-variables, say $h_{j_r,j_{r+1}}[i]$ and $h_{j_r,j_{r+1}}[l]$, that do not have the same value. This would suggest a recombinant on the edge (j_r, j_{r+1}) between the loci i and l. Consider the collection of the markers between of loci i and l of each member on the path P as the *region* where this recombinant could occur. The size of the region is $(t+1)(l-i+1)$. If the region is not very large, it contains at most one recombinant with a high probability (since k is a constant). Thus, we could enumerate all the possible locations of this recombinant in the region to locate it exactly.

Before we give the algorithm, we need some notations to describe a random instance of k-RHC. For each founder j, we use the random variable $q_{j,1}[i]$ to represent j's maternal allele at locus i and $q_{j,2}[i]$ to represent j's paternal allele at locus i. These q-variables are independent and they collectively represent the founder haplotypes. Random h-variables are used to represent the random inheritance. Although h-variables concerning different edges in the pedigree are independent, the h-variable concerning the same edge are not. The latter variables are related by the random recombinants. For convenience, we call the edges in the pedigree graph incident to the founders the *founder edges*. The other edges are called the *non-founder edges*. In the following subsections, we will show that we can determine many h-variable values (or summations of their values) on these two kinds of edges separately. These determined h-variables and summations will be used as additional constraints besides $\mathcal{C}$ to aid the search for the locations of recombinants.

4.1 Determining h-Variables on Non-founder Edges

For each founder j and locus i, the phase $p_j[i]$ is only determined by the random founder allele variables $q_{j,1}[i]$ and $q_{j,2}[i]$. The p-variables of non-founders are determined by both the q-variables and h-variables. When the h-variables are fixed, each phase $p_j[i]$ of a non-founder is only determined by two random q-variables $q_{f,s}[i]$ and $q_{g,t}[i]$. In other words, the paternal allele of member j at locus i is inherited from $q_{f,s}[i]$ and its maternal allele is from $q_{g,t}[i]$. If $(f,s)=(g,t)$, the two alleles of j at locus i are inherited from the same allele of some founder (see Fig. 2(B)). In this case, the locus i of member j is homozygous no matter what $q_{f,s}[i], q_{g,t}[i]$ are. We say that member j is *pre-homozygous* at locus i. If $(f,s)\neq(g,t)$, the two alleles of member j at locus i are inherited from different alleles of the founders (see Fig. 2(A)). Then the locus i of member j can be homozygous or heterozygous with equal probability. We say that member j is *pre-heterozygous* at locus i.

Clearly, for a tree pedigree, all its members are pre-heterozygous at every locus. Using this property, the next lemma shows that the phases of many loci are predetermined around non-founder edges in a random k-RHC instance and thus we can determine the h-variable values on many non-founder edges.

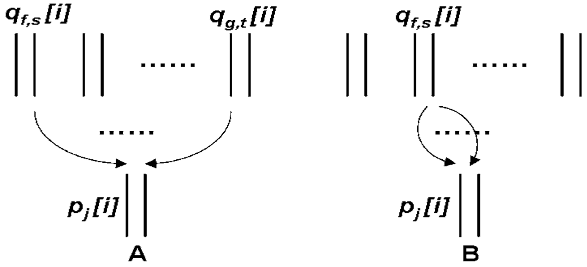
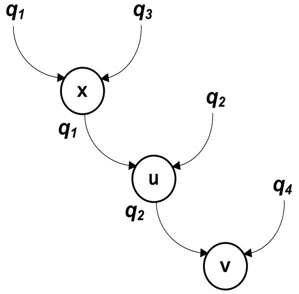


Fig. 2 **A** Member j is pre-heterozygous and its alleles are inherited from different alleles of the founders. **B** Member j is pre-homozygous and its alleles are inherited from the same allele of some founder

Fig. 3 The transmission of founder alleles



Lemma 7 *Consider a random instance of k-RHC on a tree pedigree. If* (u, v) *is a non-founder edge in the pedigree graph with u being the parent, then the probability for u to be heterozygous at locus i and both u and v to be pre-determined at locus i* (*and thus* $h_{u,v}[i]$ *to be determined*) *is at least* $1/8$.

Proof Since all loci are pre-heterozygous, we assume that $p_u[i]$ is determined by two different random founder allele variables $q_1[i]$ and $q_2[i]$ when all the h-variables are fixed. See Fig. 3. Since u is not a founder, it has a parent x. Similarly, $p_x[i]$ is determined by variables $q_1[i]$ and $q_3[i]$. Since the pedigree is a tree, $q_3[i]$ is different from $q_1[i]$ and $q_2[i]$. Without loss of the generality, we assume that u passes allele $q_2[i]$ to v (according to the h-variables). Hence, $p_v[i]$ is determined by $q_2[i]$ and another different variable $q_4[i]$. The probability that the locus i is heterozygous at u and homozygous at both x and v is

$$\begin{aligned}
&\Pr(g_x[i] \neq 2, g_u[i] = 2, g_v[i] \neq 2)\\
&\quad = \Pr(q_3[i] = q_1[i], q_1[i] \neq q_2[i], q_4[i] = q_2[i])\\
&\quad = \Pr(q_3[i] = q_1[i]) \Pr(q_1[i] \neq q_2[i]) \Pr(q_4[i] = q_2[i])\\
&\quad = \frac{1}{8}.
\end{aligned}$$

Note that when the locus i of u is heterozygous and loci i of x and v are homozygous, the edge (u, v) exists in the locus graph i. By Lemma 1, $h_{u,v}[i]$ can be determined by $p_u[i]$ and $p_v[i]$ with probability at least 1/8. □

4.2 Determining h-Variables on Founder Edges

Without loss of generality, we assume that each founder has at least two children (otherwise recombinants on the edge between the founder and its only child cannot be detected and in fact are unnecessary). For a founder x, if it is homozygous at locus i, all the h-variables concerning locus i and founder edges incident on x will not appear in any constraints. If it is heterozygous at locus i, its phase will not be predetermined for it has no parents. So, we cannot determine the h-variables on founder edges directly like in Lemma 7. However, we can determine the summation of any pair of h-variables concerning the same founder.

Lemma 8 *Consider a random instance of k-RHC on a tree pedigree. If x is a founder with children u and v, then the probability for a locus i to be heterozygous at x but pre-determined at u and v (and thus the summation $h_{x,u}[i] + h_{x,v}[i]$ to be determined) is at least* $1/8$.

Proof Denote the random alleles of x at locus i as $q_1[i]$ and $q_2[i]$. Suppose that u has alleles $q_1[i]$ and $q_3[i]$. Let us first assume that u and v do not share the same parents. Without loss of generality, suppose that v has alleles $q_1[i]$ and $q_4[i]$. (The lemma trivially holds if v inherits a different allele than q_1 from x.) Thus, the probability that the locus i is heterozygous at x but homozygous at both u and v is

$$\begin{aligned}
&\Pr(g_x[i] = 2, g_u[i] \neq 2, g_v[i] \neq 2)\\
&\quad = \Pr(q_1[i] \neq q_2[i], q_1[i] = q_3[i], q_1[i] = q_4[i])\\
&\quad = \Pr(q_1[i] = q_2[i]) \Pr(q_1[i] \neq q_3[i]) \Pr(q_1[i] = q_4[i])\\
&\quad = \frac{1}{8}.
\end{aligned}$$

Again, by Lemma 1 we can determine $h_{x,u}[i] + h_{x,v}[i]$ with probability at least $1/8$.

Next, let us assume that u and v share the same parents (e.g. members x and y). Suppose that y has alleles $q_3[i]$ and $q_4[i]$. If y is homozygous at locus i, both $p_u[i]$ and $p_v[i]$ are pre-determined. So, the probability that we can determine $h_{x,u}[i] + h_{x,v}[i]$ is

$$\begin{aligned}
&\Pr\big(g_x[i] = 2, g_y[i] \neq 2\big)\\
&\quad = \Pr(q_1[i] \neq q_2[i], q_3[i] = q_4[i])\\
&\quad = \Pr(q_1[i] \neq q_2[i]) \Pr(q_3[i] = q_4[i])\\
&\quad = \frac{1}{4}.
\end{aligned}$$

Combining both cases, the lemma follows. □

Now we are ready to describe how to locate the recombinants. We divide the loci of each member (which could be a haplotype block) into $\frac{m}{a \log n}$ disjoint segments of size $a \log n$ each, where a is a constant to be decided later on, and treat the interior and boundary segments differently. (The boundary segments are the two segments at the end.) It turns out that the boundary segments are much tougher to deal with.

4.3 Locating Recombinants in the Interior Locus Segments

Since we can determine each h-variable with probability $1/8$ for every non-founder edge, we can determine at least one h-variable in each segment with high probability for each non-founder edge (see Fig. 4). If there is at most one recombinant in any two consecutive segments associated with the same non-founder edge, we can locate such a recombinant in a small region of size $O(\log n)$ by comparing the determined h-variables of both segments. If the values of two neighboring determined h-variables

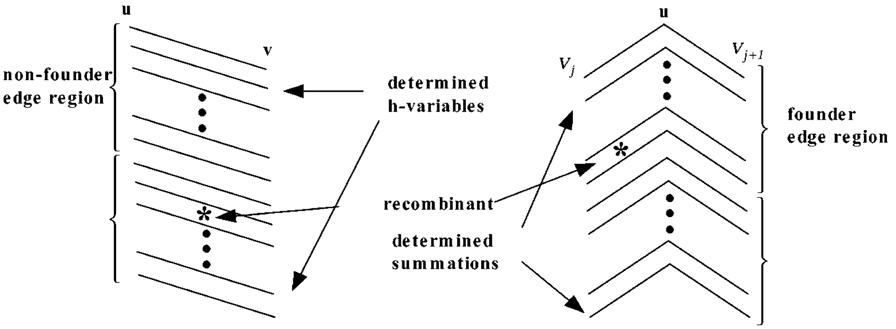


Fig. 4 Determining recombinants on founder and non-founder edges. The *figure* also illustrates two consecutive non-founder edge regions each of which is expected to contain a determined h-variable and two consecutive founder edge regions each of which is expected to contain a determined summation of h-variables. Each of these pairs of regions contains a recombinant sandwiched by the determined h-variables or summations

are equal, there is no recombinant between the loci of the h-variables (since there is at most one recombinant between the loci). Otherwise, there exists one. Similarly, we can locate recombinants associated with the founder edges. Suppose that u is a founder and $v_1, \dots, v_l$ are its children. Because we can determine $h_{u,v_s}[i] + h_{u,v_t}[i]$ for each pair of children v_s and v_t at locus i with probability at least $1/8$, we can determine at least one summation $h_{u,v_s} + h_{u,v_t}$ in each segment with high probability. If the values of two neighboring determined summations are equal, there is no recombinant between the associated loci. Otherwise, there exists one. The details of the location algorithm are given in algorithm LOCATE-INTERIOR-RECOMBINANTS as shown in Fig. 5.

Lemma 9 *The procedure* LOCATE-INTERIOR-RECOMBINANTS *can locate each recombinant from an interior locus segment in a small region of size at most* $4a \log n$ *correctly with probability at least* $1 - k^2 \frac{9a \log n}{(m-1)n} - \frac{2nm}{a \log n}(\frac{7}{8})^{a \log n}$.

Proof Call each pair of segments of the two end vertices of a non-founder edge concerning the same loci a *non-founder edge region* (see Fig. 4). Since we can determine the h-variable at each locus of a non-founder edge independently with probability at least $1/8$ according to Lemma 7, we can determine at least one h-variable in a non-founder edge region with probability at least $1 - (\frac{7}{8})^{a \log n}$. Assume that there are n_1 non-founder edges in the pedigree graph, and thus $\frac{n_1 m}{a \log n}$ non-founder edge regions. By union bound, we can determine at least one h-variable in each non-founder edge region with probability at least $1 - \frac{n_1 m}{a \log n}(\frac{7}{8})^{a \log n}$. So, the number of loci between two neighboring determined h-variables are at most $2a \log n$. If there is at most one recombinant in any two adjacent non-founder edge regions on the same edge, there exist at most one recombinant between any two neighboring determined h-variables (with high probability). For any two neighboring determined h-variables with different values, there should be a recombinant between them and we can locate it in a region of size at most $2a \log n$ since size of a non-founder edge region is $a \log n$.

```
Procedure: LOCATE-INTERIOR-RECOMBINANTS
Input:   A tree pedigree with genotype information $g_j[i]$.
Output:  Some small regions $R_1, \ldots, R_{k_1}$ that have recombinants.
  $k_1 = 0$.
  for each non-founder edge in a locus graph $G_i$
      if the $p$-variables of its end vertices are pre-determined then
          determine its corresponding $h$-variable.
for each non-founder edge $(u, v)$
for each pair of neighboring determined variables $h_{u,v}[i_1]$ and $h_{u,v}[i_2]$
          if $h_{u,v}[i_1] \neq h_{u,v}[i_2]$ then
              There exists a recombinant between loci $i_1$ and $i_2$ on the edge $(u, v)$.
              $k_1 = k_1 + 1$. Output this region $R_{k_1} = [u, v, i_1, i_2]$
          else
              for $i_1 \le i \le i_2$ do $h_{u,v}[i] = h_{u,v}[i_1]$.
  for each founder $u$ with children $v_1, \ldots, v_l$
      for each pair of children $v_j$ and $v_{j+1}$ and locus $i$
          if $g_u[i] = 2$ and $p_{v_j}[i]$ and $p_{v_{j+1}}[i]$ are pre-determined then
              Calculate the summation $h_{u,v_j}[i] + h_{u,v_{j+1}}[i]$.
      for each pair of children $v_j$ and $v_{j+1}$
          for each pair of neighboring determined summations on loci $i_1$ and $i_2$
              if $h_{u,v_j}[i_1] + h_{u,v_{j+1}}[i_1] \neq h_{u,v_j}[i_2] + h_{u,v_{j+1}}[i_2]$ then
                  There exists a recombinant between loci $i_1$ and $i_2$
                  on edge $(u, v_j)$ or edge $(u, v_{j+1})$.
                  $k_1 = k_1 + 1$. Output this region $R_{k_1} = [u, v_j, v_{j+1}, i_1, i_2]$
              else
                  for $i_1 \le i \le i_2$ do $h_{u,v_j}[i] + h_{u,v_{j+1}}[i] = h_{u,v_j}[i_1] + h_{u,v_{j+1}}[i_1]$.
```

Fig. 5 The procedure for locating recombinants in the interior locus segments

Now we consider founder edges. Assume that founder u has l children $v_1, \ldots, v_l$. For any pair of children v_j and v_{j+1}, we can determine the summation $h_{u,v_j}[i] + h_{u,v_{j+1}}[i]$ at locus i with probability at least $1/8$ according to Lemma 8. Let us regard the segments of u, v_j, v_{j+1} concerning a same set of loci a *founder edge region* (see Fig. 4). With probability at least $1 - (\frac{7}{8})^{a \log n}$, there is at least one locus i in a founder edge region such that the summation $h_{u,v_j}[i] + h_{u,v_{j+1}}[i]$ is determined. Suppose that there are n_2 founder edges in the pedigree graph. There are at most $n_2 - 1$ pairs of children of the form (v_j, v_{j+1}) in the whole pedigree. Thus, we have at most $\frac{(n_2-1)m}{a \log n}$ founder edge regions. By union bound, we can determine at least one summation of two h-variables in each founder edge region with probability at least $1 - \frac{(n_2-1)m}{a \log n}(\frac{7}{8})^{a \log n}$. So, the number of loci between any two neighboring determined summation of h-variables is at most $2a \log n$. If there is at most one recombinant in any two adjacent founder edge regions concerning the same pair of children, there is at most one recombinant between two neighboring determined h-variables (with high probability). For any two neighboring determined summations of h-variables with different values, there should be a recombinant between them, and we can locate it in a region of size at most $4a \log n$ since the size of a founder edge region is $2a \log n$.

For any two specific recombinants r_1 and r_2, if r_1 is in a non-founder edge region, then there are three scenarios for them to be in two adjacent non-founder edge regions. One case is that r_2 is in the upper adjacent non-founder edge region, the second case is that r_2 is in the same non-founder edge region, and the third case is that r_2 is in the lower adjacent non-founder edge region. Since the size of a non-founder edge region is $a\log n$, the probability that r_2 is located in a non-founder edge adjacent (or identical) to r_1 is $\frac{3a\log n}{(m-1)n}$. When r_1 is on founder edge (u, v_j), there are six founder edge regions for r_2 to be located to be adjacent to r_1. Three of them are on edges (u, v_{j-1}) and (u, v_j), and the other three on edges (u, v_j) and (u, v_{j+1}). So, the probability that r_1 and r_2 are located in adjacent (or the same) founder edge regions is $\frac{9a\log n}{(m-1)n}$. In summary, the probability for two specific recombinants to be in two adjacent founder edge regions (or non-founder edge regions) is at most $\frac{9a\log n}{(m-1)n}$. By union bound, the probability for any two recombinants not to be in any two consecutive founder edge regions (or non-founder edge regions) is at least $1-k^2\frac{9a\log n}{(m-1)n}$. Thus, procedure LOCATE-INTERIOR-RECOMBINANTS can locate each recombinant in a small region of size at most $4a\log n$ with probability at least $1-k^2\frac{9a\log n}{(m-1)n}-\frac{(n_2-1)m}{a\log n}(\frac{7}{8})^{a\log n}-\frac{n_1 m}{a\log n}(\frac{7}{8})^{a\log n}$. Since the number of edges in the pedigree graph is at most $2n$, the probability is at least $1-k^2\frac{9a\log n}{(m-1)n}-\frac{2nm}{a\log n}(\frac{7}{8})^{a\log n}$. □

4.4 Locating Recombinants in the Boundary Locus Segments

For a non-founder (or founder) edge (u, v), suppose that i_s is the smallest locus such that $h_{u,v}[i_s]$ (or a summation containing $h_{u,v}[i_s]$) can be determined and i_t is the largest such locus. By Lemma 9, each recombinant between loci i_s and i_t on edge (u, v) is located in a small region of size at most $4a\log n$. But the lemma does not show how to decide if there exists a recombinant between loci 1 and i_s or one between loci i_t and m. We call these two regions, which are typically contained in the boundary segments, the *boundary regions* of edge (u, v). In this subsection, we will show how to locate recombinants from the boundary regions in small regions of size $O(\log n)$.

For convenience, define the *length* of a constraint as the number of h-variables in it. First, we give an upper bound on the maximum length of any constraint in the set $\mathcal{C} = C^{\mathbb{C}} \cup C^{\mathbb{P}} \cup C^{\mathbb{T}}$.

Lemma 10 *For any constant b, the length of every constraint in the set $\mathcal{C}$ is less than $b\log n$ with probability* $1-2mn^2(\frac{1}{2})^{\frac{1}{4}b\log n}$.

Proof For any two vertices u and v in the spanning tree $\mathcal{T}$, there exists a unique path between u and v on the spanning tree. We consider the vertices on this path and assign each vertex a label as follows. All the vertices whose parents are not on this path are labeled 1. For each remaining vertex, its label is 1 plus the larger label of its parents (if both are on the path).

Let L include the vertices that have no children on the path. We partition the rest of the vertices into disjoint sets according to their labels. Vertices with label t are put in set F_t. See Fig. 6. If this path belongs to a connected component of the locus forest $\mathcal{T}(G_i)$, all the vertices that are not in set L are heterozygous. If a vertex is heterozygous, it will pass alleles 1 and 2 to the next generation randomly with equal probability. Thus, each of its children is also heterozygous with probability $\frac{1}{2}$.

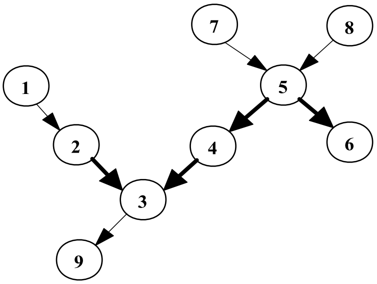


Fig. 6 An example path in a spanning forest. The *figure* illustrates a connected component of some spanning forest. The highlighted edges represent the path between vertices 2 and 6. For this path, $L = \{3, 6\}$, $F_1 = \{2, 5\}$, $F_2 = \{4\}$

Notice that any two random h-variables on different edges are independent from each other. When all the vertices in set F_{t-1} are heterozygous, all the vertices in set F_t are heterozygous with probability $\frac{1}{2}$ independently. From symmetry it is easy to see that a vertex in set F_1 is heterozygous with probability $\frac{1}{2}$. Since the pedigree is a tree, the vertices in F_1 do not have any common ancestors. In other words, they do not share any common q-variables. So, the vertices in F_1 are heterozygous with probability $\frac{1}{2}$ independently. Suppose that there are l vertices on the path and s vertices in the set L. It is easy to prove that $s \le (l+1)/2$. Thus, the probability that such a path between u and v exists in $\mathcal{T}(G_i)$ is

$$\begin{aligned}
&\Pr(\text{there exists a path of length } l)\\
&\quad = \Pr\big(g_{F_t}[i] = 2, 1 \le t \le f\big)\\
&\quad = \Pr\big(g_{F_1}[i] = 2\big)\Pr\big(g_{F_2}[i] = 2 \mid g_{F_1}[i] = 2\big)\cdots\\
&\qquad \Pr\big(g_{F_j}[i] = 2 \mid g_{F_t}[i] = 2, 1 \le t \le f-1\big)\\
&\quad = \left(\frac{1}{2}\right)^{l-s}\\
&\quad \le \left(\frac{1}{2}\right)^{(l-1)/2}.
\end{aligned}$$

Here, f denotes the largest label assigned and $g_{F_t}[i] = 2$ denotes the event that all the vertices in set F_t are heterozygous at locus i. There are at most n^2 different paths in the spanning tree $\mathcal{T}$. By union bound, the length of a path between any two vertices in each connected component of $\mathcal{T}(G_i)$ is less than l with probability at least $1 - mn^2(\frac{1}{2})^{(l-1)/2}$. Thus, with probability $1 - mn^2(\frac{1}{2})^{(l-1)/2}$, the length of each tree constraint in set $C^{\mathbb{T}}$ is less than l, the length of each path constraint in set $C^{\mathbb{P}}$ is less than $2l+1$, and the length of each cycle constraint in set $C^{\mathbb{C}}$ is less than $l+1$. Setting $l = \frac{1}{2}b\log n - 1$ concludes the proof of the lemma. □

Now we give the basic idea of locating recombinants in the boundary regions. Let us consider two adjacent loci $i-1$ and i. Suppose that all the h-variables at locus i have already been determined. In other words, for the h-variables concerning non-founder edges, their values are known, and for the h-variables corresponding to founder edges, we know the summation of any pair of h-variables concerning edges incident on the same founder vertex. Note that for a founder u with children $v_1, \ldots, v_l$, the summation $h_{u,v_s}[i] + h_{u,v_t}[i]$ for any pair of children v_s, v_t $(s < t)$ can be calculated using $\sum_{j=s}^{t-1}(h_{u,v_j}[i] + h_{u,v_{j+1}}[i])$. If there is no recombinant between loci $i-1$ and i, all the h-variables at locus $i-1$ will be the same as those at locus i. So, we can set $h_{u,v}[i-1] = h_{u,v}[i]$ for each non-founder edge (u,v) and $h_{u,v_j}[i-1] + h_{x,v_{j+1}}[i-1] = h_{u,v_j}[i] + h_{u,v_{j+1}}[i]$ for each founder u with children $v_1, \ldots, v_l$, and then check if all the constraints in the set $\mathcal{C}_{i-1}$ (i.e. all the constraints in $\mathcal{C}$ generated in locus graph G_{i-1}) hold. Note that by Lemma 5, each constraint in $\mathcal{C}_{i-1}$ contains an even number of founder edges incident on the same founder. So, the validity of each constraint in $\mathcal{C}_{i-1}$ can be determined. If any constraint is unsatisfied, there is at least one recombinant on this constraint (path) between loci $i-1$ and i. Since each constraint contains fewer than $b \log n$ h-variables by Lemma 10, it can be regarded as a small region. Thus, each constraint contains no more than one recombinant with high probability. If all the constraints hold, there are no recombinants between these two loci. Otherwise, we can locate each recombinant in a region of size $b \log n$ (i.e. some unsatisfied constraint in $\mathcal{C}_{i-1}$). By iterating this towards locus 1 and locus m separately, we can locate all boundary recombinants.

A pseudocode for locating boundary recombinants, called LOCATE-BOUNDARY-RECOMBINANTS, is given in Fig. 7. It assumes that the procedure LOCATE-INTERIOR-RECOMBINANTS has been run to locate all recombinants in the interior regions. Once all the recombinants have been located, LOCATE-BOUNDARY-RECOMBINANTS in fact returns a feasible (final) solution in terms of the p-variables.

Figure 8 gives the details of our main algorithm TREE k-RHC. It first calls a simple preprocessing procedure as shown in Fig. 9 to set up the constraints and then the procedures LOCATE-BOUNDARY-RECOMBINANTS and LOCATE-INTERIOR-RECOMBINANTS to locate the recombinants and construct a feasible solution. Before we analyze the performance of algorithm TREE k-RHC, we prove two lemmas. An h-variable is called *active* if it appears in some constraints in $\mathcal{C}$. Otherwise, it is *inactive*. Clearly, the values of inactive h-variables have no impact on the constraints.

Lemma 11 *For any edge (u,v) and set of $2a \log n$ consecutive loci $i+1$, $i+2, \ldots, i+2a\log n$, at least one of $h_{u,v}[i+1], \ldots, h_{u,v}[i+2a\log n]$ is active with probability at least $1 - \frac{2nm}{a \log n}(\frac{7}{8})^{a \log n}$.*

Proof It follows from the proof of Lemma 9 that each non-founder edge region has at least one determined h-variable and each founder edge region has at least one determined summation of two h-variables with probability at least $1 - \frac{2nm}{a \log n}(\frac{7}{8})^{a \log n}$. Each such determined $h_{u,v}[i]$ or $h_{u,v}[i] + h_{u,v'}[i]$ suggests an active h-variable at locus i. □

The next lemma shows that we can focus on active h-variables when trying to locate the recombinants.

Lemma 12 *For each edge (u,v), if $h_{u,v}[i_1] \neq h_{u,v}[i_2]$ and all the h-variables $h_{u,v}[i]$ $(i_1 < i < i_2)$ are inactive, then there is a recombinant between loci i_1 and i_2 on edge (u,v). Moreover, any two consecutive loci in this interval would be a feasible location for this recombinant.*

```
Procedure: LOCATE-BOUNDARY-RECOMBINANTS($i_1, i_2, r_1, \ldots, r_{k_1}$)
Input:     Two loci $i_1 < i_2$ such that all the recombinants between loci $i_1 + 1$
           and $i_2 - 1$ have been located and the $h$-variables determined;
           the determined recombinant locations $r_1, \ldots, r_{k_1}$.
Output:    A feasible assignment of all recombinant locations and the $p$-variables.
$l_1 = 0, l_2 = 0$.
for each non-founder edge $(u, v)$
    $h_{u,v}[i_1] = h_{u,v}[i_1 + 1]; h_{u,v}[i_2] = h_{u,v}[i_2 - 1]$.
for each founder $u$ with children $v_1, \ldots, v_s$ and $j = 1, \ldots, s - 1$
    $h_{u,v_j}[i_1] + h_{u,v_{j+1}}[i_1] = h_{u,v_j}[i_1 + 1] + h_{u,v_{j+1}}[i_1 + 1]$.
    $h_{u,v_j}[i_2] + h_{u,v_{j+1}}[i_2] = h_{u,v_j}[i_2 - 1] + h_{u,v_{j+1}}[i_2 - 1]$.
for each constraint $c$ in set $\mathcal{C}_{i_1}$
    If $c$ is unsatisfied then
        $l_1 = l_1 + 1$.
        There is a recombinant between loci $i_1$ and $i_1 + 1$ on $c$.
        Denote this region as $R_{k_1+l_1}$.
for each constraint $c$ in set $\mathcal{C}_{i_2}$
    If constraint $c$ is unsatisfied then
        $l_2 = l_2 + 1$.
        There is a recombinant between loci $i_2$ and $i_2 - 1$ on $c$.
        Denote this region as $R_{k_1+l_1+l_2}$.
for each possible assignment of recombinant
      locations $(r_{k_1+1}, \ldots, r_{k_1+l_1+l_2}) \in (R_{k_1+1}, \ldots, R_{k_1+l_1+l_2})$
    if the number of distinct recombinants is larger than $k$ then return.
    if any two recombinants are derived from the same constraint then continue.
    for $1 \le j \le l_1$
        If $r_{k_1+j}$ is identical to any $r_i$ $(k_1 < i < k_1 + j)$ then continue
        If $r_{k_1+j}$ is on a non-founder edge $(u, v)$ then
            $h_{u,v}[i_1] = h_{u,v}[i_1 + 1] + 1$.
        If $r_{k_1+j}$ is on a founder edge $(u, v_j)$,
            where $u$ is a founder with children $v_1, \ldots, v_s$, then
            $h_{u,v_{j-1}}[i_1] + h_{u,v_j}[i_1] = h_{u,v_{j-1}}[i_1 + 1] + h_{u,v_j}[i_1 + 1] + 1$.
            $h_{u,v_j}[i_1] + h_{u,v_{j+1}}[i_1] = h_{u,v_j}[i_1 + 1] + h_{u,v_{j+1}}[i_1 + 1] + 1$.
    for $1 \le j \le l_2$
        If $r_{k_1+j}$ is identical to any $r_i$ $(k_1 < i < k_1 + j)$ then continue
        If $r_{k_1+l_1+j}$ is on a non-founder edge $(u, v)$ then
            $h_{u,v}[i_2] = h_{u,v}[i_2 - 1] + 1$.
        If $r_{k_1+l_1+j}$ is on a founder edge $(u, v_j)$,
            where $u$ is a founder with children $v_1, \ldots, v_s$, then
            $h_{u,v_{j-1}}[i_2] + h_{u,v_j}[i_2] = h_{u,v_{j-1}}[i_2 - 1] + h_{u,v_j}[i_2 - 1] + 1$.
            $h_{u,v_j}[i_2] + h_{u,v_{j+1}}[i_2] = h_{u,v_j}[i_2 - 1] + h_{u,v_{j+1}}[i_2 - 1] + 1$.
    if $i_1 \le 1$ and $i_2 \ge m$ then
        Translate the ILP in (5) into a tree ZRHC instance taking
        into account the recombinants $(r_1, \ldots, r_{k_1+l_1+l_2})$.
        Solve the instance using the linear-time algorithm in [21].
        Output the recombinants $(r_1, \ldots, r_{k_1+l_1+l_2})$ and the $p$-variables. stop.
    elseif $i_1 \le 1$ and $i_2 < m$ then
        LOCATE-BOUNDARY-RECOMBINANTS($0, i_2 + 1, r_1, \ldots, r_{k_1+l_1+l_2}$).
    elseif $i_1 > 1$ and $i_2 \ge m$ then
        LOCATE-BOUNDARY-RECOMBINANTS($i_1 - 1, m + 1, r_1, \ldots, r_{k_1+l_1+l_2}$).
    else
        LOCATE-BOUNDARY-RECOMBINANTS($i_1 - 1, i_2 + 1, r_1, \ldots, r_{k_1+l_1+l_2}$).
```

Fig. 7 The procedure for locating recombinants in the boundary regions. For convenience, we define $\mathcal{C}_0 = \mathcal{C}_{m+1} = \emptyset$. For the simplicity of presentation, here we enumerate all new recombinants together. But in our real implementation, we actually enumerate a recombinant at a time and then update the corresponding h-variables' values. This guarantees that the same recombinant will not be enumerated twice

```
Algorithm: TREE $k$-RHC
Input:  A tree pedigree $G = (V, E)$ with genotype information $g_j[i]$; parameter $k$.
Output: Recombinant locations and haplotype information $p_j[i]$.
GENERATE-CONSTRAINTS.
LOCATE-INTERIOR-RECOMBINANTS.
for each edge $(u, v)$
    Find the smallest locus $i$ with $h_{u,v}[i]$
    (or some summation $h_{u,v}[i] + h_{u,v'}[i]$) determined. Let $s_{u,v} = i$.
    Find the largest locus $i$ with $h_{u,v}[i]$
    (or some summation $h_{u,v}[i] + h_{u,v'}[i]$) determined. Let $t_{u,v} = i$.
Let $s = \max s_{u,v}$, $t = \min t_{u,v}$.
if $k_1 = 0$ then
    LOCATE-BOUNDARY-RECOMBINANTS($s - 1, t + 1$).
else
    for each assignment of recombinant locations $(r_1, r_2, \dots, r_{k_1}) \in (R_1, \dots, R_{k_1})$
        for $1 \le j \le k_1$
            if $R_j = [u, v, i_1, i_3]$ where $(u, v)$ is a non-founder edge,
                and $r_j$ is between loci $i_2$ and $i_2 + 1$ then
                $h_{u,v}[i] = h_{u,v}[i_1]$ for each $1 \le i \le i_2$.
                $h_{u,v}[i] = h_{u,v}[i_3]$ for each $i_2 + 1 \le i \le i_3$.
            if $R_j = [u, v_1, v_2, i_1, i_3]$ where $(u, v_1)$ and $(u, v_2)$ are two adjacent founder edges,
               and $r_j$ is between loci $i_2$ and $i_2 + 1$ then
                $h_{u,v_1}[i] + h_{u,v_2}[i] = h_{u,v_1}[i_1] + h_{u,v_2}[i_1]$ for each $i_1 \le i \le i_2$.
                $h_{u,v_1}[i] + h_{u,v_2}[i] = h_{u,v_1}[i_3] + h_{u,v_2}[i_3]$ for each $i_2 + 1 \le i \le i_3$.
        for $s \le i \le t$
            Check all the constraints in set $\mathcal{C}_i$.
            if any constraint is unsatisfied then break the loop.
        LOCATE-BOUNDARY-RECOMBINANTS($s - 1, t + 1, R_1, \dots, R_{k_1}$).
```

Fig. 8 The main algorithm for solving k-RHC on tree pedigrees

```
Procedure: GENERATE-CONSTRAINTS
Input:  A tree pedigree graph $G = (V, E)$ with genotype information $g_j[i]$.
Output: $\mathcal{T}(G)$, $G_i$, $\mathcal{T}(G_i)$, and the constraint set $\mathcal{C}$.
Construct an arbitrary spanning tree $\mathcal{T}(G)$ on G.
for each locus $i$
    Generate the locus graph $G_i$.
    Generate the locus spanning forest $\mathcal{T}(G_i)$.
    Identify the pre-determined $p$-variables at locus $i$.
Generate the constraints $\mathcal{C} = C^{\mathbb{C}} \cup C^{\mathbb{P}} \cup C^{\mathbb{T}}$.
```

Fig. 9 The preprocessing procedure

Proof Since the variables $h_{u,v}[i]$ ($i_1 < i < i_2$) are inactive, their values do not affect the validity of any constraint. In other words, they are totally “free”. Hence, a recombinant can be located anywhere among them without affecting feasibility by Lemma 6. Because $h_{u,v}[i_1] \neq h_{u,v}[i_2]$, there exist an odd number of recombinants between loci i_1 and i_2. Since we are generally interested in solutions with the fewest recombinants, we may assume that only one recombinant exists. □

To prove that the algorithm TREE k-RHC finds a feasible solution in $O(mn \log^{k+1} n)$ time with high probability, we need only show that all the recombinants can be located in the correct regions with high probability.

Theorem 13 *For any $a > 0, b > 0$ and $m > 2a\log n$, the algorithm* Tree k-RHC *solves the probabilistic k-RHC problem on tree pedigrees in time $O(mn\log n(\max\{4a, b\}\log n)^k)$ with probability at least $1 - k^2\frac{9a\log n}{(m-1)n} - \frac{2nm}{a\log n}(\frac{7}{8})^{a\log n} - 2mn^2(\frac{1}{2})^{\frac{1}{4}b\log n} - k(k-1)\frac{2ab\log^2 n}{(m-1)n}$.*

Proof By Lemma 9, all recombinants in the interior segments are located correctly by procedure Locate-Interior-Recombinants with high probability. So, we need only prove that procedure Locate-Boundary-Recombinants can locate all the remaining recombinants correctly. By symmetry, we will consider only the upper boundary segments below. Note that the top locus is locus 1. As in algorithm Tree k-RHC, for each non-founder (or founder) edge (u, v), the smallest locus i with $h_{u,v}[i]$ (or some summation containing $h_{u,v}[i]$, respectively) determined is denoted as $s_{u,v}$. The largest locus i with $h_{u,v}[i]$ (or some summation containing $h_{u,v}[i]$, respectively) determined is denoted as $t_{u,v}$. Let $s = \max s_{u,v}$ and $t = \min t_{u,v}$. Recall that a segment contains $a\log n$ loci. When $m > 2a\log n$, the loci of each member is divided into at least two segments. So it is easy to see that $s \le a\log n < t$. Since all the recombinants on edge (u, v) between loci $s_{u,v}$ and $t_{u,v}$ have been correctly located by procedure Locate-Interior-Recombinants, we may assume that all the h-variables associated with the edge (u, v) between loci $s_{u,v}$ and $t_{u,v}$ have been correctly determined.

Consider a recombinant r_f between loci l and $l + 1$ on edge (u, v), where $l < s_{u,v}$. Suppose that i is the largest locus above l (i.e. $i \le l$) such that $h_{u,v}[i]$ is active. Similarly, let $h_{u,v}[j]$ ($j \ge l + 1$) be the nearest active h-variable below r_f. Thus, $h_{u,v}[i] \ne h_{u,v}[j]$, and we will prove that this recombinant will be located correctly by our algorithm by induction on i. Let us call the above $h_{u,v}[i]$ an *affected variable*. We also say that the recombinant r_f is *observed* at the above locus i and *observable on* any constraint that contains $h_{u,v}[i]$. Clearly, it follows from these definitions that each recombinant affects a unique variable and is observed at a unique locus. Assume that all the recombinants observed between loci $i + 1$ and s have been located correctly in the entire pedigree. We want to show that the recombinant r_f is correctly located.

Suppose that $h_{u,v}[i]$ is contained in some constraint c that contains only one affected variable. In other words, $h_{u,v}[i]$ is the unique affected variable in c. Because all the recombinants observed between loci $i + 1$ and s are located correctly, there is a feasible solution consistent with the h-variable values and summations determined up to locus $i + 1$ with the following properties: (i) the h-variables that appear in constraint c have the same values (or give rise to the same summations) as their counterparts at locus $i + 1$ except $h_{u,v}[i]$ and (ii) the constraint c is satisfied. Note that although there may still exist unidentified recombinants below locus $i + 1$ at this point, they should cause no problem to the existence of such a feasible solution because they do not affect any (active) h-variables below locus i. Since the procedure Locate-Boundary-Recombinants first copies the h-variable values (or summations) at locus $i + 1$ to locus i, it is easy to see that the constraint c is not satisfied by these copied values and summations, and r_f would be located correctly. We denote as A_f the event that r_f does not have any affected variable $h_{u,v}[i]$ which is the unique affected variable in any constraint, when all the recombinants between loci $i + 1$ and s are located correctly. In other words, A_f represents the event that we do not observe recombinant r_f when all the recombinants between loci $i+1$ and s are located correctly. So, the probability that LOCATE-BOUNDARY-RECOMBINANTS works correctly is $P = 1 - \Pr(A_1 \cup A_2 \cup \cdots \cup A_{k-k_1})$. By symmetry and union bound, $P \geq 1 - (k-k_1)\Pr(A_f) \geq 1 - k\Pr(A_f)$.

Because $h_{u,v}[i]$ is active, it is contained in at least one constraint c. If c does not reveal r_f, there must be other recombinants that are also observable on c. Thus, $\Pr(A_f) \leq (k-1)\Pr(r_{f'} \text{ is observable on } c)$, where $r_{f'}$ denotes any recombinant different from r_f. Let Q_i denote the set of all founder allele q-variables at locus i and H_i the set of all h-variables at locus i. Then,

$$\begin{aligned}
&\Pr(r_{f'} \text{ is observable on constraint } c) \\
&\quad = \sum_{H_i, Q_i} \Pr(H_i, Q_i) \cdot \Pr(r_{f'} \text{ is observable on constraint } c \mid H_i, Q_i).
\end{aligned}$$

When H_i and Q_i are fixed, constraint c is also fixed, but the recombinant $r_{f'}$ has equal probability to be at any location in the pedigree. By Lemma 10, constraint c has at most $b\log n$ h-variables. By Lemmas 11 and 12, each recombinant must be at most $2a\log n$ loci away from a constraint where it is observable. So, there are at most $2ab\log^2 n$ possible locations where the recombinant $r_{f'}$ can be located and remain observable on constraint c. Thus, $\Pr(r_{f'} \text{ is observable on constraint } c \mid H_i, Q_i) \leq \frac{2ab\log^2 n}{(m-1)n}$, and

$$\begin{aligned}
P &\geq 1 - k\Pr(A_f) \\
&\geq 1 - k(k-1)\sum_{H_i, Q_i} \Pr(H_i, Q_i)\frac{2ab\log^2 n}{(m-1)n} \\
&= 1 - k(k-1)\frac{2ab\log^2 n}{(m-1)n}.
\end{aligned}$$

Combining Lemmas 9 and 10, algorithm TREE k-RHC correctly locates the recombinants and returns a feasible solution with probability at least $1 - k^2\frac{9a\log n}{(m-1)n} - \frac{2nm}{a\log n}(\frac{7}{8})^{a\log n} - 2mn^2(\frac{1}{2})^{\frac{1}{4}b\log n} - k(k-1)\frac{2ab\log^2 n}{(m-1)n}$.

Now we analyze the time complexity of the algorithm. Each recombinant is first located in a region of size at most $\max\{4a, b\}\log n$ in the procedures LOCATE-INTERIOR-RECOMBINANTS and LOCATE-BOUNDARY-RECOMBINANTS. The exact locations of these recombinants are then exhaustively enumerated in procedure LOCATE-BOUNDARY-RECOMBINANTS and TREE k-RHC, which generate altogether $(\max\{4a, b\}\log n)^k$ combinations. By Lemma 4, there are $O(mn)$ constraints. Since each constraint has at most $b\log n$ h-variables, we can check if each of them is satisfied in $O(mn\log n)$ time. For a founder vertex u with children $v_1, \ldots, v_l$, we only keep track of summations of the form $h_{u,v_j}[i] + h_{u,v_{j+1}}[i]$ at a locus i in procedure LOCATE-BOUNDARY-RECOMBINANTS. But when we move on to locus $i-1$, we may need summations of the form $h_{u,v_j}[i] + h_{u,v_l}[i]$ (it could be useful in step 19 of TREE k-RHC). We can quickly obtain such summations when needed by setting temporarily $h_{u,v_1}[i] = 0$ and calculating all the other h-variable

$h_{u,v_j}[i]$ according to the summations $h_{u,v_j}[i] + h_{u,v_{j+1}}[i]$. Then we can obtain any summation $h_{u,v_j}[i] + h_{u,v_l}[i]$ in constant time. It is easy to see that such computation takes $O(n)$ time for each locus in the whole pedigree and thus $O(mn)$ time for all loci. Since the algorithm in [21] finds a feasible solution of ZRHC on tree pedigrees in $O(mn)$ time, the total running time of algorithm TREE k-RHC is $O(mn\log n(\max\{4a, b\}\log n)^k)$. □

Corollary 14 *When* $90\log n < m < n^3$, TREE k-RHC *solves the probabilistic k-RHC problem on tree pedigrees in time* $O(mn\log^{k+1} n)$ *with probability* $1 - O(k^2\frac{\log^2 n}{mn} + \frac{1}{n^2})$.

Proof Let $a = \frac{6}{\log 8/7}$ and $b = \frac{28}{\log 2}$, where the log function is to the base e. Then $1 - k^2\frac{9a\log n}{(m-1)n} - \frac{2nm}{a\log n}(\frac{7}{8})^{a\log n} - 2mn^2(\frac{1}{2})^{\frac{1}{4}b\log n} - k(k-1)\frac{2ab(\log n)^2}{(m-1)n} = 1 - O(k^2\frac{\log^2 n}{mn} + \frac{1}{n^2})$, and $O(mn\log n(\max\{4a, b\}\log n)^k) = O(mn(\log n)^{k+1})$. Since we also need $m > 2a\log n$ in the proof of Theorem 13, $m > 90\log n$. □

5 Some Implementation Issues

The above algorithm TREE k-RHC is originally based on the simple probabilistic model where the founder haplotypes, haplotype transmission and recombinants are all assumed to follow the uniform distribution. Moreover, it assumes that the markers are bi-allelic and contain no missing alleles. To deal with real data that may not satisfy these assumptions, we need carefully resolve the following issues.

Missing Data In practice, some alleles may be missing due to the imperfectness of genotyping technologies. Here, we simply omit the constraints described in Sect. 3.2 that involve missing alleles and proceed to infer the h-variables. Some missing alleles can be imputed uniquely using the solved h-variables and the constraints, while the remaining ones can be enumerated. Once the missing alleles of a member are enumerated, more missing alleles of other members could be determined using the constraints. In our experiments, we only had to enumerate the alleles at a few loci (about 4 to 8 loci) of a very small number of members (about 2 to 4 members) in the input pedigree.

Multi-Allelic Markers We can extend the construction of the linear equations in Sect. 2 to work for multi-allelic data. Suppose that member j_r is a parent of member j. For each locus i, we define $p_j[i] = 0$ if the allele at locus i of j with the larger ID is located on j's paternal haplotype, and $p_j[i] = 1$ otherwise. It is easy to prove that there always exists a linear equation of the form $c_1 \cdot p_{j_r}[i] + c_2 \cdot h_{j_r,j}[i] + c_3 \cdot p_j[i] + c_4 = 0$ over the field $F(2)$ to describe the inheritance from j_r to j at locus i. Here the definition of p-variable is a little bit different from that in Sect. 2. We can enumerate all 16 possible cases of (c_1, c_2, c_3, c_4) and find one that agrees with the inheritance on all 8 possible combinations of $p_{j_r}[i]$, $p_j[i]$ and $h_{j_r,j}[i]$. The same strategy can be used to construct the constraints in Sect. 3.2.

Bounded Breadth-First-Search A straightforward implementation of the algorithm TREE k-RHC using (for example) breadth-first-search (BFS) may not be very efficient when the number of the recombinants becomes very large, because the product of the areas of all regions containing recombinants increases exponentially. So, when k becomes large (e.g. 10 in our experiments), we could switch to a *bounded BFS* strategy to speed up the search. The basic idea is that we keep only B nodes during BFS that incur the smallest number of recombinants before exploring the next level of the search, for some appropriately chosen B. Clearly, a smaller B would allow the program to run faster, but perhaps with less accuracy. In our experiments on real data, we found that setting $B = 20$ worked very well.

6 Preliminary Experimental Results

The above implemented C++ program will simply be named TREE-k-RHC, which is available to the public upon request to either of the authors. To evaluate the performance of TREE-k-RHC, we compare it with both the ILP-based exact algorithm (called ILP) and fast heuristic algorithm (called BE) in PedPhase [16–18] on simulated genotype data in terms of accuracy and efficiency using two different pedigree structures. ILP is known to be the best combinatorial program for haplotype inference on pedigrees in the literature [19]. The comparison shows that TREE-k-RHC is comparable to ILP in terms of accuracy but much faster than ILP on large data. On the other hand, TREE-k-RHC is comparable to BE in terms of speed but much more accurate. We further compare the performance of TREE-k-RHC, ILP and an EM algorithm from [10] on a real dataset that consists of 12 multi-generation pedigrees studied in [10]. The results show that TREE-k-RHC is able to discover almost all the common haplotypes (i.e. haplotypes with frequencies at least 5%) that were inferred by the EM algorithm and ILP. These results are discussed in more detail below.

6.1 Comparing TREE-k-RHC, BE and ILP on Simulated Data

Two real tree human pedigrees from the literature [20] are considered. One has 29 members as shown in Fig. 10(a) and the other 52 members as shown in Fig. 10(b).

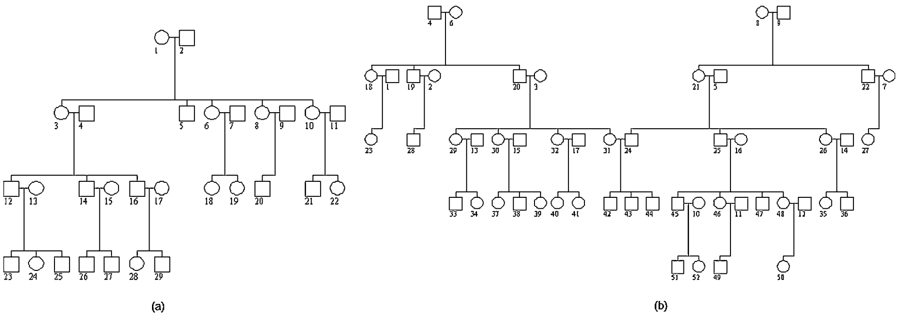


Fig. 10 The pedigree structures used in the simulation

Table 1 Speeds and numbers of recombinants of Tree-k-RHC, BE and ILP on simulated data. The first column indicates the combination of parameters: the size of the pedigree, the number of loci in each member and the number of recombinants used to generate the genotype data, respectively. The time was measured on a desktop PC with P4 3 GHz CPU and 512 M memory running Windows operating system

Parameters	Missing	Tree-k-RHC		BE		ILP	
	rate	Time (s)	# Recombinants	Time (s)	# Recombinants	Time(s)	# Recombinants
(29,60,10)	0.00	0.307	9.6	0.117	48.5	3.722	9.6
	0.05	0.302	9.7	0.125	119.0	4.772	9.6
	0.10	0.293	9.6	0.125	154.0	5.595	9.6
	0.15	0.355	9.9	0.125	197.5	6.616	9.7
	0.20	0.478	9.9	0.125	212.5	11.639	9.7
(29,90,20)	0.00	0.444	18.5	0.133	80.0	11.115	18.3
	0.05	0.429	18.3	0.148	175.0	14.572	18.3
	0.10	0.425	18.5	0.141	250.0	16.741	18.3
	0.15	0.453	18.7	0.148	314.0	17.706	18.3
	0.20	0.678	19.2	0.179	359.5	23.293	18.5
(29,90,10)	0.00	0.431	9.7	0.125	99.5	8.009	9.7
	0.05	0.412	9.7	0.133	173.0	10.027	9.7
	0.10	0.400	9.9	0.133	236.5	10.993	9.7
	0.15	0.398	9.8	0.148	311.0	12.828	9.7
	0.20	0.406	9.9	0.133	362.0	14.364	9.7
(29,60,2)	0.00	0.302	2.0	0.117	9.5	3.203	2.0
	0.05	0.292	2.0	0.117	84.0	4.044	2.0
	0.10	0.280	2.0	0.117	123.5	4.544	2.0
	0.15	0.277	2.0	0.117	169.0	5.213	2.1
	0.20	0.348	2.1	0.117	226.5	5.750	2.1
(52,60,10)	0.00	1.070	9.1	0.133	12.0	9.643	9.1
	0.05	1.036	9.1	0.141	117.5	12.797	9.1
	0.10	1.042	9.1	0.125	213.5	14.626	9.5
	0.15	1.181	9.3	0.125	312.5	16.230	9.5
	0.20	1.356	9.0	0.125	384.5	18.086	9.6

On each of these pedigrees, bi-allelic genotype data are generated randomly following the simple probabilistic model with various numbers of loci, missing rates and number of recombinants. For each configuration, 100 random replicates are used and the average speed and performance of each program on these replicates are assessed. The results in Table 1 demonstrate that Tree-k-RHC is more than 20 times faster than ILP and about 10 times slower than BE. Moreover, it yields solutions with much fewer recombinants than BE and similar numbers of recombinants as ILP. (Although ILP is supposed to be an exact algorithm, its actual implementation uses heuristics to speed up as well.) Note that the speed up over ILP should grow quickly

Table 1 (*Continued*)

Parameters	Missing rate	Tree-k-RHC		BE		ILP	
		Time (s)	# Recombinants	Time (s)	# Recombinants	Time(s)	# Recombinants
(52,90,20)	0.00	1.566	18.5	0.157	107.0	21.581	18.2
	0.05	1.530	18.6	0.344	262.0	27.822	18.3
	0.10	1.510	18.3	0.148	365.0	32.136	18.6
	0.15	1.647	18.6	0.141	492.0	35.598	18.5
	0.20	1.946	18.5	0.164	608.0	40.470	18.3
(52,50,10)	0.00	0.925	8.9	0.133	23.0	6.922	8.9
	0.05	0.927	8.9	0.133	108.5	9.362	8.9
	0.10	1.041	8.9	0.125	194.5	10.526	8.9
	0.15	1.641	9.1	0.125	239.0	11.878	9.1
	0.20	3.554	9.1	0.125	329.0	13.336	8.9
(52,80,15)	0.00	1.390	13.3	0.149	60.0	16.529	13.3
	0.05	1.345	13.3	0.141	187.0	21.293	13.3
	0.10	1.333	13.5	0.141	352.5	25.148	13.2
	0.15	1.350	13.5	0.171	462.5	27.467	13.3
	0.20	1.760	13.6	0.156	561.5	30.141	13.3
(52,80,20)	0.00	1.404	17.5	0.148	102.0	17.999	17.5
	0.05	1.353	17.6	0.148	260.0	23.138	17.7
	0.10	1.363	17.8	0.141	390.0	26.812	17.7
	0.15	1.494	17.9	0.164	478.0	29.471	17.7
	0.20	1.721	18.5	0.164	550.5	37.973	17.5
(52,95,15)	0.00	1.649	13.5	0.157	83.5	23.880	13.5
	0.05	1.582	13.7	0.148	228.5	31.519	13.8
	0.10	1.517	13.7	0.156	403.0	36.617	13.8
	0.15	1.480	13.9	0.156	544.5	44.405	14.2
	0.20	1.548	14.0	0.164	619.5	52.325	14.6

with m and n as the worst-case time complexity of the ILP-based algorithm is at least $(mn)^k$.

For accuracy, we calculate the percentage of loci whose phases are correctly inferred and the percentage of missing alleles correctly imputed by the programs. As shown in Table 2, Tree-k-RHC and ILP have comparable accuracies. Both programs are able to infer phases correctly in more than 96% of the cases and impute missing alleles correctly in more than 90% of the cases. Interestingly, although BE requires a lot of extra recombinants on these data, it can somehow infer the phase and missing allele information correctly in more than 86% and 70%, respectively, of the cases.

Table 2 The percentage of loci with correctly inferred phases and the percentage of missing alleles correctly imputed by Tree-k-RHC, BE and ILP on simulated data. Again, the first *column* indicates the pedigree size, the number of loci and the number of recombinants

Parameters	Missing	Tree-k-RHC		BE		ILP	
	rate	Phase	Missing allele	Phase	Missing allele	Phase	Missing allele
(29,60,10)	0.00	0.994	NA	0.974	NA	0.994	NA
	0.05	0.992	0.956	0.944	0.754	0.993	0.958
	0.10	0.989	0.949	0.926	0.790	0.988	0.944
	0.15	0.983	0.935	0.903	0.756	0.983	0.937
	0.20	0.977	0.924	0.893	0.748	0.978	0.929
(29,90,20)	0.00	0.992	NA	0.967	NA	0.990	NA
	0.05	0.990	0.947	0.944	0.758	0.987	0.947
	0.10	0.986	0.931	0.917	0.738	0.982	0.942
	0.15	0.980	0.928	0.888	0.730	0.977	0.934
	0.20	0.974	0.914	0.874	0.732	0.973	0.925
(29,90,10)	0.00	0.992	NA	0.972	NA	0.995	NA
	0.05	0.989	0.947	0.942	0.754	0.989	0.950
	0.10	0.985	0.941	0.910	0.738	0.986	0.947
	0.15	0.980	0.931	0.892	0.730	0.984	0.943
	0.20	0.975	0.924	0.865	0.712	0.978	0.934
(29,60,2)	0.00	0.999	NA	0.998	NA	0.999	NA
	0.05	0.997	0.965	0.966	0.766	0.998	0.967
	0.10	0.995	0.958	0.960	0.780	0.995	0.956
	0.15	0.990	0.944	0.915	0.754	0.991	0.949
	0.20	0.984	0.936	0.874	0.708	0.986	0.937
(52,60,10)	0.00	0.995	NA	0.990	NA	0.996	NA
	0.05	0.991	0.938	0.962	0.769	0.992	0.938
	0.10	0.986	0.925	0.939	0.768	0.988	0.933
	0.15	0.978	0.915	0.899	0.742	0.983	0.926
	0.20	0.972	0.907	0.899	0.742	0.983	0.926
(52,90,20)	0.00	0.990	NA	0.980	NA	0.991	NA
	0.05	0.986	0.931	0.944	0.754	0.987	0.932
	0.10	0.981	0.923	0.921	0.754	0.983	0.927
	0.15	0.974	0.910	0.896	0.740	0.977	0.919
	0.20	0.968	0.903	0.873	0.722	0.972	0.914
(52,50,10)	0.00	0.995	NA	0.988	NA	0.996	NA
	0.05	0.990	0.939	0.965	0.771	0.990	0.938
	0.10	0.985	0.926	0.928	0.750	0.986	0.930
	0.15	0.979	0.918	0.903	0.724	0.981	0.923
	0.20	0.973	0.910	0.878	0.720	0.975	0.914

Table 2 (*Continued*)

Parameters	Missing rate	TREE-k-RHC Phase	TREE-k-RHC Missing allele	BE Phase	BE Missing allele	ILP Phase	ILP Missing allele
(52,80,15)	0.00	0.993	NA	0.984	NA	0.993	NA
	0.05	0.989	0.936	0.965	0.792	0.989	0.937
	0.10	0.985	0.929	0.922	0.762	0.984	0.932
	0.15	0.979	0.918	0.897	0.740	0.979	0.926
	0.20	0.971	0.909	0.875	0.727	0.973	0.917
(52,80,20)	0.00	0.990	NA	0.977	NA	0.991	NA
	0.05	0.985	0.923	0.937	0.765	0.986	0.933
	0.10	0.981	0.920	0.912	0.742	0.983	0.928
	0.15	0.974	0.908	0.892	0.740	0.978	0.923
	0.20	0.963	0.967	0.900	0.873	0.971	0.913
(52,95,15)	0.00	0.992	NA	0.982	NA	0.994	NA
	0.05	0.988	0.927	0.948	0.782	0.989	0.933
	0.10	0.982	0.919	0.923	0.764	0.984	0.928
	0.15	0.977	0.914	0.897	0.752	0.979	0.922
	0.20	0.969	0.904	0.879	0.740	0.974	0.915

6.2 Comparing TREE-k-RHC, ILP and an EM Algorithm on a Real Dataset

Gabriel et al. [10] 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 250 kbps, spanning a total of 13.4 Mbps (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 designed by Excoffier and Slatkin [9]. One of the populations (the European population) contains pedigrees and was used in our study. There are totally 93 members in the European population, separated into 12 multi-generation pedigrees (each with 7 to 8 members). The genotyped regions are distributed among all the 22 autosomes and each autosome contains 1 to 10 regions. We downloaded the SNP genotype data and pedigree structures from Whitehead/MIT Center for Genome Research website (http://www-genome.wi.mit.edu/mpg/hapmap/hapstruc.html), and obtained the results of the EM algorithm and ILP concerning common haplotypes and their frequencies in the European population as given in [17, 18].

We focus on chromosome 3 as in [17, 18]. There are four regions in the chromosome 3 data and each region is partitioned into 1 to 4 blocks as in [10]. The physical length and partitioned block information of each region from [10] are summarized in Table 3. Since TREE-k-RHC does not work well with a very small number of loci (this can also been seen from the condition in Corollary 14), we run it on each of the

Table 3 The regions and blocks on chromosome 3

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

regions instead of blocks. Once haplotypes are inferred for the members of all pedigrees, haplotype frequencies (in the population) are estimated by simple counting. The common haplotypes and their frequencies in each block, estimated by TREE-k-RHC, ILP and the EM algorithm are summarized in Table 4. Since region 16a-1 has only 14 loci, which are too few for TREE-k-RHC, we omit it from the comparison. The majority of the common haplotypes identified by TREE-k-RHC and ILP for all blocks are the same as those found by the EM algorithm. Furthermore, for the common haplotypes found by all three programs, the programs estimated very similar frequencies.

We also compare the numbers of recombinations required in the solutions found by ILP and TREE-k-RHC, and the similarity between the phases inferred by them. We randomly select four regions from the entire genome and report their results in Table 5. Again, TREE-k-RHC is run on each region while ILP is run on each block. We observe that the number of the recombinants that TREE-k-RHC found in each block are the same as ILP's numbers in most of the blocks, except for one (i.e. block 17a-3). In this block, ILP requires no recombinant but TREE-k-RHC requires one. After a further analysis, we find that this extra recombinant required by TREE-k-RHC is buried in a sequence of consecutive homozygous loci across the boundary between this block and a neighboring block, and can thus be shifted to the boundary without affecting the haplotype solution. In other words, this recombinant does not have to be in this block, and thus we can treat the results of ILP and TREE-k-RHC on this block as identical. The table also shows that for more than 99.5% of the loci, both program inferred the same phases. Note that, here ILP takes advantage of the available haplotype block structure while TREE-k-RHC is able to produce a similar haplotype solution without using the block information. In fact, TREE-k-RHC puts most of the recombinants required in a region at the boundary between haplotype blocks.

The minimum number of loci required by TREE-k-RHC depend on many factors including the size of the pedigree, the structure of the pedigree, and the distribution of the alleles. Our rough empirical estimation shows that TREE-k-RHC (for a reason-

Table 4 Common haplotypes and their frequencies obtained by TREE-k-RHC, ILP and the EM method. In haplotypes, the alleles are encoded as $1 = \text{A}$, $2 = \text{C}$, $3 = \text{G}$, and $4 = \text{T}$

Block	Common haplotype	EM	ILP	TREE-k-RHC	Block	Common haplotype	EM	ILP	TREE-k-RHC
17a-1	3 1 3 4 4 4	0.340	0.292	0.315	18a-1	1444231214144132	0.269	0.240	0.222
	1 3 3 2 4 2	0.302	0.250	0.282		1444111214144132	0.240	0.208	0.277
	3 3 2 4 2 4	0.135	0.093	0.118		1444131214144132	0.189	0.198	0.231
	3 3 3 4 4 4	0.102	0.135	0.131		4222133313412211	0.125		
	3 3 2 4 4 4	0.068	0.073	0.065		1444231234144132	0.083	0.073	0.092
	1 3 3 2 4 4	0.052				4444133214144132		0.052	0.064
Sum		1.000	0.844	0.911	Sum		0.906	0.771	0.886
17a-2	2 3 2 4 2	0.354	0.323	0.337	18a-2	3 1 2 4 4 2	0.497	0.427	0.501
	3 3 4 2 4	0.333	0.313	0.337		1 3 2 4 3 4	0.260	0.167	0.272
	3 3 4 4 2	0.146	0.156	0.139		3 1 2 2 4 2	0.127	0.094	0.098
	3 4 4 4 4	0.125	0.125	0.156		1 3 4 4 4 4	0.094	0.073	0.075
Sum		0.958	0.917	0.969		1 3 2 4 3 2		0.063	
17a-3	4 4 3 1	0.413	0.417	0.447		3 1 2 4 4 4		0.052	
	3 1 1 2	0.281	0.229	0.208	Sum		0.978	0.892	0.946
	4 1 3 1	0.236	0.219	0.201	18a-3	2 2 1 1	0.419	0.385	0.417
	4 1 3 2	0.070	0.073	0.082		4 3 3 3	0.219	0.219	0.217
Sum		1.000	0.930	0.938		2 3 1 1	0.206	0.240	0.194
17a-4	3 4 4 1 2 4	0.385	0.344	0.401		4 3 1 3	0.125	0.115	0.147
	2 3 2 4 3 2	0.333	0.302	0.307	Sum		0.969	0.958	0.975
	3 4 2 4 2 4	0.250	0.219	0.235	16b-2	4 1 2 2	0.541	0.510	0.516
Sum		0.969	0.865	0.943		2 3 3 4	0.281	0.250	0.269
16b-1	3 2 4 1 1 2	0.801	0.781	0.797		2 3 3 2	0.156	0.156	0.148
	1 3 2 3 3 4	0.083	0.083	0.062	Sum		0.978	0.917	0.933
Sum		0.885	0.865	0.859					

ably large k) requires about 40 loci when $n \le 20$ and about 60 loci when $20 < n < 50$. The number does not increase much once n passes 50.

7 Concluding remarks

This paper presents an efficient algorithm for k-RHC on tree pedigrees without mating loops. The algorithm is capable of handling large tree pedigrees. On the other hand, the k-RHC problem on general pedigrees with mating loops seems much harder to deal with. By extending the techniques in this paper, we have only been able to obtain a nontrivial algorithm that runs in time $O(mn^2 + n^3 \log n (n \log n)^k)$ with success probability $1 - O(k^2 \frac{\log n}{m} + \frac{1}{n^2})$. This (latter) algorithm is not efficient for large general pedigrees, and thus further research is needed.

Acknowledgements We are very grateful to the annonymous referees for their many constructive suggestions and comments. The research was supported in part by NIH grant 2R01LM008991, NSF

Table 5 Comparison of the number of recombinations in the solutions found by ILP and TREE-k-RHC. The phase similarity column means the percentage of loci with the same phases inferred by these two programs

Region	Block	# Recombinants (TREE-k-RHC)	# Recombinants (ILP)	Phase similarity
11a		0		0.996
	11a-1	0	0	
17a		14		0.996
	17a-1	0	0	
	17a-2	0	0	
	17a-3	1	0	
	17a-4	0	0	
18a		4		0.995
	18a-1	0	0	
	18a-2	0	0	
	18a-3	2	2	
19a		8		0.996
	19a-1	3	3	
	19a-2	1	1	

grant IIS-0711129, NSFC grant 60553001, and National Key Project for Basic Research (973) grants 2002CB512801 and 2007CB807901.

References

1. Abecasis, G.R., Cherny, S.S., Cookson, W.O., Cardon, L.R.: Merlin–rapid analysis of dense genetic maps using sparse gene flow trees. Nat. Genet. **30**(1), 97–101 (2002)
2. Albers, C.A., Heskes, T., Kappen, H.J.: Haplotype inference in general pedigrees using the cluster variation method. Genetics **177**, 1101–1116 (2007)
3. Axenovich, T.I., Zorkoltseva, I.V., Liu, F., Kirichenko, A.V., Aulchenko, Y.S.: Breaking loops in large complex pedigrees. Hum. Hered. **65**(2), 57–65 (2008)
4. Baruch, E., Weller, J.I., Cohen-Zinder, M., Ron, M., Seroussi, E.: Efficient inference of haplotypes from genotypes on a large animal pedigree. Genetics **172**, 1757–1765 (2006)
5. Chan, M.Y., Chan, W.T., Chin, F., Fung, S., Kao, M.Y.: Linear-time haplotype inference on pedigrees without recombinations. In: Proc. WABI'2006, pp. 56–67 (2006). Also in SIAM J. Comput. **38**(6), 2179–21797 (2009)
6. Chin, F., Zhang, Q., Shen, H.: k-recombination haplotype inference in pedigrees. In: Proc. 5th International Conference on Computational Science (ICCS), Atlanta, GA, pp. 985–993 (2005)
7. Doi, K., Li, J., Jiang, T.: Minimum recombinant haplotype configuration on pedigrees without mating loops. In: Proc. Workshop on Algorithms in Bioinformatics (WABI), Budapest, Hungary, pp. 339–353 (2003)
8. Downey, R., Fellows, M.: Parameterized Complexity. Springer, Berlin (1999)
9. Excoffier, L., Slatkin, M.: Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. Mol. Biol. Evol. **12**, 921–927 (1995)
10. Gabriel, S.B., Schaffner, S.F., Nguyen, H., Moore, J.M., Roy, J., Blumenstiel, B., Higgins, J., DeFelice, M., Lochner, A., Faggart, M., Liu-Cordero, S.N., Rotimi, C., Adeyemo, A., Cooper, R., Ward, R., Lander, E.S., Daly, M.J., Altshuler, D.: The structure of haplotype blocks in the human genome. Science **296**(5576), 2225–2229

11. Griffiths, A., Gelbart, W., Lewontin, R., Miller, J.: Modern Genetic Analysis: Integrating Genes and Genomes. Freeman, New York (2002)
12. Gudbjartsson, D.F., Jonasson, K., Frigge, M.L., Kong, A.: Allegro, a new computer program for multipoint linkage analysis. Nat. Genet. **25**(1), 12–13 (2000)
13. Haplotype Conference, Birmingham, AL, May, 2008. www.soph.uab.edu/ssg/nhgri/haplotype2008
14. Kruglyak, L., Daly, M.J., Reeve-Daly, M.P., Lander, E.S.: Parametric and nonparametric linkage analysis: a unified multipoint approach. Am. J. Hum. Genet. **58**, 1347–1363 (1996)
15. Lander, E.S., Green, P.: Construction of multilocus genetic linkage maps in humans. Proc. Natl. Acad. Sci. USA **84**, 2363–2367 (1987)
16. Li, J., Jiang, T.: Efficient rule-based haplotyping algorithm for pedigree data. In: Proc. 7th Annual Conference on Research in Computational Molecular Biology (RECOMB'03), pp. 197–206 (2003)
17. Li, J., Jiang, T.: An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming. In: Proc. RECOMB'04, pp. 20–29 (2004)
18. Li, J., Jiang, T.: Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by integer linear programming. J. Comput. Biol. **12**(6), 719–739 (2005)
19. Li, J., Jiang, T.: A survey on haplotyping algorithms for tightly linked markers. J. Bioinform. Comput. Biol. **6**(1), 241–259 (2008)
20. Li, X., Li, J.: Efficient haplotype inference from pedigrees with missing data using linear systems with disjoint-set data structures. In: Proc. 7th Annual International Conference on Computational Systems Bioinformatics (CSB2008) (2008)
21. Liu, L., Jiang, T.: Linear-time reconstruction of zero-recombinant Mendelian inheritance on pedigrees without mating loops. In: Proc. 18th International Conference on Genome Informatics (GIW), Singapore, Dec, pp. 95–106 (2007)
22. Liu, L., Chen, X., Xiao, J., Jiang, T.: Complexity and approximation of the minimum recombination haplotype configuration problem. In: Proc. 16th International Symposium on Algorithms and Computation (ISAAC'05), pp. 370–379 (2005)
23. O'Connell, J.R.: Zero-recombinant haplotyping: Applications to fine mapping using SNPs. Genet. Epidemiol. **19**(Suppl. 1), S64–S70 (2000)
24. Piccolboni, A., Gusfield, D.: On the complexity of fundamental computational problems in pedigree analysis. J. Comput. Biol. **10**(5), 763–773 (2003)
25. Qian, D., Beckmann, L.: Minimum-recombinant haplotyping in pedigrees. Am. J. Hum. Genet. **70**(6), 1434–1445 (2002)
26. Sobel, E., Lange, K., O'Connell, J., Weeks, D.: Haplotyping algorithms. In: Speed, T., Waterman, M. (eds.) Genetic Mapping and DNA Sequencing. IMA Vol in Math and Its App, vol. 81, pp. 89–110. Springer, Berlin (1996)
27. The International HapMap Consortium: International HapMap Project. Nature **426**, 789–796 (2003)
28. Wang, C., Wang, Z., Qiu, X., Zhang, Q.: A method for haplotype inference in general pedigrees without recombination. J. Chin. Sci. Bull. **52**(4), 471–476 (2007)
29. Wilson, I.J., Dawson, K.J.: A Markov chain Monte Carlo strategy for sampling from the joint posterior distribution of pedigrees and population parameters under a Fisher-Wright model with partial selfing. Theor. Popul. Biol. **72**(3), 436–58 (2007)
30. Xiao, J., Liu, L., Xia, L., Jiang, T.: Fast elimination of redundant linear equations and reconstruction of recombination-free Mendelian inheritance on a pedigree. In: Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'07), pp. 655–664. (2007)
31. Xiao, J., Liu, L., Xia, L., Jiang, T.: Efficient algorithms for reconstructing zero-recombinant haplotypes on a pedigree based on fast elimination of redundant linear equations. SIAM J. Comput. **38**(6), 2198–2219 (2009)