An Efficient Algorithm for Haplotype Inference on Pedigrees with a Small Number of Recombinants (Extended Abstract)

Jing Xiao[1], Tiancheng Lou[2], and Tao Jiang[3]

[1] IBM China Research Lab, Beijing, China
`xiaojing82@gmail.com`
[2] Department of Computer Science and Technology, Tsinghua University, Beijing, China
`loutiancheng860214@gmail.com`
[3] Department of Computer Science and Engineering, University of California, Riverside, CA
`jiang@cs.ucr.edu`

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 recombinants 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 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 im-

A. Fiat and P. Sanders (Eds.): ESA 2009, LNCS 5757, pp. 325–336, 2009.

portant 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 [16,17,18,25] where the goal is to find a haplotype solution requiring the minimum number of recombinants. 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 [7,21]. 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 [5,16,20,22,23,28,30,31]. 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 [5,20,22] 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 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 k. 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 [5,20,22,30,31], 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 [5,20,22] 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 expectedly 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 PedPahse [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 [22]). 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 [5,20,22].

The rest of the paper is organized as follows. In Section 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 Section 4. Due to the page limit, we will omit all the technical proofs, figures, tables, and pseudocodes as well discussions on the implementation of the algorithm and our experimental results in this extended abstract and present them in the full paper which will soon be submitted to a journal.

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 $\boldsymbol{g}_j$ 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 $\boldsymbol{p}_j$ describes the paternal and maternal haplotypes of member j. Observe that the vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ represent a complete haplotype configuration of the pedigree. Also, for technical reasons, define a vector $\boldsymbol{w}_j$ 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 $\boldsymbol{d}_{j_1,j} = \boldsymbol{0}$ and $\boldsymbol{d}_{j_2,j} = \boldsymbol{w}_j$, the above equations can be unified into a single equation as:

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

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:

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

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

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 Equation 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. 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. $\mathcal{T}(G)$ 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 $|E^{\mathbb{X}}| \leq n+1$.

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 $\mathcal{T}(G_i)$. 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 $\boldsymbol{d}_{l,j} = \boldsymbol{d}_{j,l}$.

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*

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

Corollary 1. *[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 2. *[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 Equation 3. Note that since the edges broken 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 1. 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.*,

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

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 2 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.*,

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

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 2, 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 2. 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

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

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

Lemma 2. *[30,31]* $|C| = |C^{\mathbb{C}}| + |C^{\mathbb{P}}| + |C^{\mathbb{T}}| = O(mn)$.

Lemma 3. *None of the constraints in* $C^{\mathbb{P}} \cup C^{\mathbb{T}}$ *are defined on a path that begins or ends at a founder.*

As in [30,31], we can prove that 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 Equation 3. The proof is very similar to the corresponding proof in [30,31] and therefore omitted. The following lemma hence follows.

Lemma 4. *The k-RHC problem can be expressed as the following ILP:*

$$\sum_{\text{for all edges } (j_r,\, j)} \sum_{i=1}^{m-1} \left| h_{j_r,j}[i] - h_{j_r,j}[i+1] \right| \le k \qquad (5)$$

plus
all the linear equality constraints in C

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 Equation 5, and the k-RHC ILP is effectively reduced to a ZRHC instance which can be solved by the linear-time algorithm in [22]. 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 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 $P = j_0, \ldots, j_t$ at locus i and another constraint along the same path P at locus l ($l > i$). By Corollary 2, 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 adjacent 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 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. 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. 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 pre-determined around non-founder edges in a random k-RHC instance and thus we can determine the h-variable values on many non-founder edges.

Lemma 5. *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* $1/8$.

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 pre-determined for it has no parents. So, we cannot determine the h-variables on founder edges directly like in Lemma 5. However, we can determine the summation of any pair of h-variables concerning the same founder.

Lemma 6. *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* $1/8$.

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. 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 are equal, there is no recombinant between the loci of the h-variables. Otherwise, there exists at least one. Similarly, we can locate recombinants associated with the founder edges. Suppose that u is a founder and $v_1, \ldots, v_l$ 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 at least one. The detailed location algorithm, called Locate-Interior-Recombinants, will be given in the full paper.

Lemma 7. *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* $1 - k^2 \frac{6a \log n}{(m-1)n} - \frac{2nm}{a \log n} \left(\frac{7}{8}\right)^{a \log n}$.

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 7, each recombinant between loci i_t and i_b 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 $C = C^{\mathbb{C}} \cup C^{\mathbb{P}} \cup C^{\mathbb{T}}$.

Lemma 8. *For any constant b, the length of every constraint in the set C is less than* $b \log n$ *with probability* $1 - 2mn^2 \left(\frac{1}{2}\right)^{\frac{1}{4} b \log n}$.

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 C_{i-1} hold. Note that by Lemma 3, each constraint in C_{i-1} contains an even number of founder edges incident on the same founder. So, the validity of each constraint in 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 8, 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 C_{i-1}). By iterating this towards locus 1 and locus m separately, we can locate all boundary recombinants.

The detailed algorithm for locating boundary recombinants, called Locate-Boundary-Recombinants, will be given in the full paper. 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.

Our main algorithm, Tree k-RHC, first calls a simple preprocessing procedure 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 C. Otherwise, it is *inactive*. Clearly, the values of inactive h-variables have no impact on the constraints.

Lemma 9. *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* $1-\frac{2nm}{a\log n}\left(\frac{7}{8}\right)^{a\log n}$.

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

Lemma 10. *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.*

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 1. *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* $O\Big(mn\log n\,(\max\{4a,b\}\log n)^k\Big)$ *with probability at least* $1-k^2\frac{6a\log n}{(m-1)n}-\frac{2nm}{a\log n}\left(\frac{7}{8}\right)^{a\log n}-2mn^2\left(\frac{1}{2}\right)^{\frac{1}{4}b\log n}-k(k-1)\frac{2ab\log^2 n}{(m-1)n}$.

Corollary 3. *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* $1-O(k^2\frac{\log^2 n}{mn}+\frac{1}{n^2})$.

Open Problem and Acknowledgement. It remains open to design an efficient algorithm for k-RHC on large general pedigrees with possibly mating loops. The research was supported in part by NIH grant LM008991, NSF grant IIS-0711129, NSFC grant 60553001, and National Key Project for Basic Research (973) grants 2002CB512801 and 2007CB807901.

References

1. Abecasis, G.R., et al.: Nat Genet, 30(1), 97–101 (2002)
2. Albers, C.A., et al.: Genetics 177, 1101–1116 (2007)
3. Axenovich, T.I., et al.: Human Heredity 65(2), 57–65 (2008)
4. Baruch, E., et al.: Genetics 172, 1757–1765 (2006)
5. Chan, M.Y., et al.: SIAM Journal on Computing 38(6), 2179–2197 (2009)
6. Chin, F., et al.: Proc. 5th ICCS, Atlanta, GA, pp. 985–993 (2005)
7. Doi, K., et al.: Minimum recombinant haplotype configuration on tree pedigrees. In: Benson, G., Page, R.D.M., et al. (eds.) WABI 2003. LNCS (LNBI), vol. 2812, pp. 339–353. Springer, Heidelberg (2003)
8. Downey, R., Fellows, M.: Parameterized Complexity. Springer, Heidelberg (1999)
9. Excoffier, L., Slatkin, M.: Mol. Biol. Evol. 12, 921–927 (1995)
10. Gabriel, S.B., et al.: Science 296(5576), 2225–2229
11. Griffiths, A., et al.: Modern Genetic Analysis: Integrating Genes and Genomes. W.H. Freeman and Company, New York (2002)
12. Gudbjartsson, D.F., et al.: Nat. Genet. 25(1), 12–13 (2000)
13. Haplotype Conference (May 2008), `http://www.soph.uab.edu/ssg/nhgri/haplotype2008`
14. Kruglyak, L., et al.: Am. J. Hum. Genet. 58, 1347–1363 (1996)
15. Lander, E.S., Green, P.: Proc. Natl. Acad. Sci. USA. 84, 2363–2367 (1987)
16. Li, J., Jiang, T.: Proc. 7th RECOMB, pp. 197–206 (2003)
17. Li, J., Jiang, T.: Proc. 8th RECOMB, pp. 20–29 (2004)
18. Li, J., Jiang, T.: J. Comput. Biol. 12(6), 719–739 (2005)
19. Li, J., Jiang, T.: J. Bioinformatics and Computational Biology 6(1), 241–259 (2008)
20. Li, X., Li, J.: Proc. 7th CSB, pp. 297–308 (2008)
21. Liu, L., et al.: Complexity and approximation of the minimum recombination haplotype configuration problem. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 370–379. Springer, Heidelberg (2005)
22. Liu, L., Jiang, T.: Proc. 18th GIW, pp. 95–106, Singapore (December 2007)
23. O'Connell, J.R.: Genet. Epidemiol. 19(suppl.1), S64–S70 (2000)
24. Piccolboni, A., Gusfield, D.: Journal of Computational Biololgy 10(5), 763–773 (2003)
25. Qian, D., Beckmann, L.: Am J Hum Genet, 70(6), 1434–1445 (2002)
26. Sobel, E., et al.: In: Speed, T., Waterman, M. (eds.) Genetic Mapping and DNA Sequencing, IMA Vol in Math. and its App., vol. 81, pp. 89–110 (1996)
27. The International HapMap Consortium. Nature 426, 789–796 (2003)
28. Wang, C., et al.: Journal of Chinese Science Bulletin 52(4), 471–476 (2007)
29. Wilson, I.J., Dawson, K.J.: Theor. Popul. Biol. 72(3), 436–458 (2007)
30. Xiao, J., et al.: Proc. 18th SODA, pp. 655–664 (2007)
31. Xiao, J., et al.: SIAM Journal on Computing 38(6), 2198–2219 (2009)