J Comb Optim (2008) 16: 229–247
DOI 10.1007/s10878-007-9129-6

A new recombination lower bound and the minimum perfect phylogenetic forest problem

Yufeng Wu · Dan Gusfield

Published online: 3 January 2008

Abstract Understanding recombination is a central problem in population genetics. In this paper, we address an established computational problem in this area: compute lower bounds on the minimum number of historical recombinations for generating a set of sequences (Hudson and Kaplan in Genetics 111, 147–164, 1985; Myers and Griffiths in Genetics 163, 375–394, 2003; Gusfield et al. in Discrete Appl. Math. 155, 806–830, 2007; Bafna and Bansal in IEEE/ACM Trans. Comput. Biol. Bioinf. 1, 78–90, 2004 and in J. Comput. Biol. 13, 501–521, 2006; Song et al. in Bioinformatics 421, i413–i244, 2005). In particular, we propose a new recombination lower bound: the forest bound. We show that the forest bound can be formulated as the minimum perfect phylogenetic forest problem, a natural extension to the classic binary perfect phylogeny problem, which may be of interests on its own. We then show that the forest bound is provably higher than the optimal haplotype bound (Myers and Griffiths in Genetics 163, 375–394, 2003), a very good lower bound in practice (Song et al. in Bioinformatics 421, i413–i422, 2005). We prove that, like several other lower bounds (Bafna and Bansal in J. Comput. Biol. 13, 501–521, 2006), computing the forest bound is NP-hard. Finally, we describe an integer linear programming (ILP) formulation that computes the forest bound precisely for certain range of data. Sim-

A preliminary version of this paper appeared in the Proceedings of COCOON 2007, LNCS, vol. 4598, pp. 16–26.

The work was performed while Y. Wu was with UC Davis and supported by grants CCF-0515278 and IIS-0513910 from National Science Foundation.
D. Gusfield supported by grants CCF-0515278 and IIS-0513910 from National Science Foundation.

Y. Wu (✉)
Computer Science and Engineering Department, University of Connecticut, Storrs, CT 06269, USA
e-mail: ywu@engr.uconn.edu

D. Gusfield
Department of Computer Science, University of California, Davis, CA 95616, USA
e-mail: gusfield@cs.ucdavis.edu

ulation results show that the forest bound may be useful in computing lower bounds for low quality data.

Keywords Recombination · Lower bound on the minimum number of recombination · Ancestral recombination graph · Population genetics · Computational complexity

1 Introduction

Meiotic recombination is an important biological process which has a major effect on shaping the genetic diversity of a population. Recombination takes two equal length sequences and produces a third sequence of the same length consisting of some prefix of one sequence, followed by a suffix of the other sequence. Estimating the frequency or the location of recombination is central to modern-day genetics. Recombination also plays a crucial role in the ongoing efforts of association mapping. Association mapping is widely hoped to help locate genes that influence complex genetic diseases. The increasingly available population genetic variation data provides opportunities for better understanding of recombination.

In this paper, we assume the input data consists of *single nucleotide polymorphisms* (*SNPs*). A SNP is a nucleotide site where exactly two (of four) different nucleotides occur in a large percentage of the population. That is, a SNP has binary states (0 or 1). A haplotype is a binary vector, where each bit (called site) of this vector indicates the state of the SNP site for this sequence. Throughout this paper, the input to our computational problems is a set of (aligned) haplotypes (i.e. a binary matrix with n rows and m columns).

An established computational problem on recombination is to determine the *minimum* number of recombinations needed to generate a set of haplotypes from an ancestral sequence, using some specified model of the permitted site *mutations*. A mutation at a SNP site is a change of state from one nucleotide to the other nucleotides at that site. Throughout this paper, we assume that any SNP site can mutate at most once in the entire history of the sequences, which is supported by the standard *infinite sites model* in population genetics.

Given a set of binary sequences M, we let $R_{min}(M)$ denote the minimum number of recombinations needed to generate the sequences M from any ancestral sequence, allowing only one mutation per site over the entire history of the sequences. The problem of computing or estimating $R_{min}(M)$ has been studied in a number of papers, for example (Hudson and Kaplan 1985; Myers and Griffiths 2003; Gusfield et al. 2007; Bafna and Bansal 2004, 2006; Song et al. 2005). A variation to the problem occurs when a specific ancestral sequence is known in advance. No polynomial-time algorithm for either problem is known, and the second problem is known to be NP-hard (Wang et al. 2001; Bordewich and Semple 2004). Therefore, the problem of computing a good lower bound on the minimum number of recombinations has attracted much attention.

In this paper, we present a new lower bound on $R_{min}(M)$, which has a static and intuitive meaning. This lower bound (which we call the forest bound) is closely related to the minimum perfect phylogenetic forest problem, an extension of the classic binary perfect phylogeny problem. We then demonstrate that the forest bound is

provably higher than a well-known bound: the optimal haplotype bound (Myers and Griffiths 2003; Song et al. 2005).[1] We resolve the complexity of computing the forest bound in a negative way with a NP-hardness proof. Finally, we give an integer linear programming formulation whose solution computes the forest bound exactly. We show empirically that this formulation can be solved in practice for data with small number of sites.

2 Background

2.1 Recombination and ARGs

For haplotype data composed of (binary) SNPs, the simplest evolutionary history that derives these haplotypes is the classic binary perfect phylogeny (if we assume the infinite sites model of mutations). The perfect phylogeny problem is to build a (rooted) tree whose leaves are labeled by rows in M and edges labeled by columns in M, and a column can label at most one edge. For the binary perfect phylogeny problem, Gusfield (1991) developed a linear time algorithm. However, often the real biological data does not have a perfect phylogeny, which is partly due to recombination. In this case, we need a richer model of evolution: The evolutionary history of a set of haplotypes H, which evolve by site mutations, assuming one mutation per site, and recombination, is displayed on a directed acyclic graph called an "Ancestral Recombination Graph (ARG)" (Griffiths and Marjoram 1996), also called phylogenetic networks in some literature. An ARG N, generating n sequences of m sites each, is a directed acyclic graph containing exactly one node (the root) with no incoming edges, and exactly n leaves with one incoming edge each. Every other node has one or two incoming edges. A node with two incoming edges is called a "recombination" node (and the two incoming edges are called recombination edges). Each site (integer) from 1 to m is assigned to exactly one edge in N, and none is assigned to any edge entering a recombination node. The sequences labeling the leaves of N are the extant sequences, i.e., the input sequences. Figure 1 shows an example of ARG. See Gusfield et al. (2004) for a more detailed explanation.

An ARG N is called a minARG if N uses exactly $R_{min}(M)$ recombinations. The ARG N may derive sequences that do not appear in M. These sequences are called Steiner sequences. Sequences in M are called input sequences.

A widely known technique for detecting recombination is the four gamete test. Two columns are said to be incompatible if the two columns contain all four patterns: 00, 01, 10 and 11. If we assume the site mutations fit the infinite sites model (i.e. at most one mutation per site), then there exists at least one recombination between a pair of incompatible sites.

2.2 Lower bounds on $R_{min}(M)$

There are a number of papers on *lower bounds* on $R_{min}(M)$ (Hudson and Kaplan 1985; Myers and Griffiths 2003; Gusfield et al. 2007; Bafna and Bansal 2006;

[1] Throughout this paper, when we say bound A is higher than bound B, we mean that bound A is guaranteed to never be lower than bound B, and that there are examples where it is strictly higher.

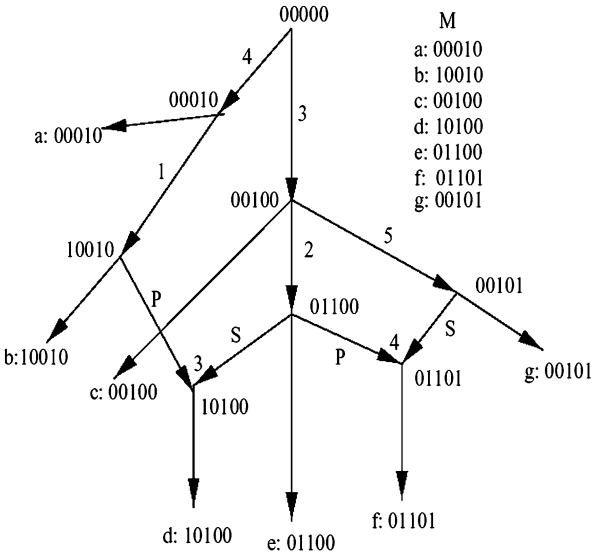

Fig. 1 A phylogenetic network (or ARG) with two recombination nodes. The sequences that this ARG derives are shown to the right. Here, internal nodes are also labeled by the given sequences. There is no Steiner sequences in this ARG. However, Steiner sequences are often needed in many ARGs. Note that if we remove sequences d and f, we remove all cycles in this ARG and we have a tree. This tree is a perfect phylogeny, where at most one mutation occurs at a site

Song et al. 2005). In Myers (2003), Myers and Griffiths (2003), Myers and Griffiths introduced the *haplotype lower bound*, which, when combined with additional ideas in Myers (2003), Myers and Griffiths (2003) significantly outperforms the previous lower bounds. The haplotype bound, $h(M)$, is simple and efficiently computable. Consider the set of sequences M arrayed in a matrix. Then $h(M)$ is the number of distinct rows of M, minus the number of distinct columns of M, minus one. It is easy to establish that this is a lower bound on $R_{min}(M)$. Simulations show that $h(M)$ by itself is a very poor bound, often a negative number. However, when used with a few more tricks, it leads to impressive lower bounds. One such trick is to compute the haplotype bound on a subset of sites from M that need not be contiguous. For a subset of sites S (not necessarily contiguous), let $M(S)$ be M restricted to the sites in S, and $h(S)$ be the haplotype bound computed on $M(S)$. It is easy to see that $h(S)$ is also a lower bound on the M. The *optimal haplotype bound* is the *highest* $h(S)$ over all choices of S. Since there are an exponential number of subsets, complete enumeration of subsets quickly becomes impractical, and the problem of computing optimal haplotype bound has also been shown to be NP-hard (Bafna and Bansal 2006). However, it was showed in Song et al. (2005) that integer linear programming (ILP) can be used to efficiently compute the optimal haplotype bound for the range of data of current biological interest. They also showed the optimal haplotype bound is often equal to $R_{min}(M)$ in certain biological datasets.

The history bound Myers and Griffiths (2003) also introduced the so-called "history bound". The history bound is provably higher than the haplotype bound (Bafna and Bansal 2004). In fact, the history bound is higher than all studied recombination lower bounds (about ten of them). However, the history bound is defined only by a computational procedure (described below), and there is no simple *static* and intuitive meaning provided for this bound in Myers and Griffiths (2003), independent

of the procedure to compute it. This makes it difficult to find alternative methods to compute the history bound, or to understand and improve it.

To compute the history bound for a set of binary sequences M, we initialize $R = 0$. A site c in M is called non-informative when entries in column c have only a single 0 or a single 1. A *cleanup step* is defined as the removal of any non-informative site in M, or the merging of two duplicate rows in M into one row. A *row removal step* arbitrarily picks one row in M for removal, provided that no cleanup step is possible. A history is defined by an execution of the following algorithm:

Repeat (a) and (b) until there is only one remaining sequence in M:

(a) Perform cleanup steps until no more cleanups is possible.
(b) Perform one row removal step by picking a row and then remove it; increment R by one.

The history lower bound is equal to the *minimum value* of R over *all* possible histories (i.e. the ways of choosing a row in the row removal step). The correctness of the history bound can be proved by induction (Myers and Griffiths 2003). Computing the history bound is NP-hard and a dynamic programming algorithm with $O(2^n m)$ running time is given by Bafna and Bansal (2006), which improves upon the original implementation by Myers and Griffiths (2003).

Interpretation of the history bound The history bound has a graphical interpretation. Consider a minimum ARG N. Suppose we want to remove the nodes in N one by one, until N becomes empty. We consider an ordering of node removal such that a node is removed *after* all its descendants have been removed. Type (a) operations correspond to removal of nodes not generated by recombination (i.e. has only a single parent node). Type (b) operations correspond to removal of recombination nodes. Thus, the number of type (b) operations is larger than the number of recombinations in N. Since N is unknown, the history bound is to find the smallest number of type (b) operations over all possible histories.

Our work on the forest bound comes out of an attempt to find a simple static definition of the history bound. A static definition is important because a definition of what is being computed, independent of how it is computed, is useful to understand and find alternative ways to compute or approximate. For example, with no static definition of the history bound, we do not know how to formulate an integer linear program to compute it.

3 The forest bound and the minimum perfect phylogenetic forest (MPPF) problem

3.1 Definition of the forest bound

The optimal haplotype bound is currently one of the best lower bounds that can be practically computed for medium-range data. In the following, we demonstrate a bound that can be proved to be higher than the optimal haplotype bound.

Given an arbitrary ARG N, suppose we remove *all* recombination edges. N is then decomposed into connected components, each of which is a directed perfect

phylogeny (sometimes simply referred to as a directed tree). Some of the tree edges are labeled by site mutations in the original ARG. Note that a site appears exactly once in N. Thus, we have a forest $\mathcal{F}(N)$ of perfect phylogenies, created by removing all recombination edges. In what follows, we will consider each of these trees after ignoring the edge directions. An important property of these trees in $\mathcal{F}(N)$ is that there is no duplicate mutations at a site in two trees in $\mathcal{F}(N)$. In other words, if a site s labels an edge in tree $T_1 \in \mathcal{F}(N)$, another tree $T_2 \in \mathcal{F}(N)$ can not have a mutation at s. This implies that sequences in T_2 have a uniform value (either all-0 or all-1) at site s. Also note that $\mathcal{F}(N)$ partitions the rows in M, where each partition is a perfect phylogeny and each row in M appears as a label in one of the perfect phylogenies. We call such partitioning of M *perfect* partitioning. It is easy to see that perfect partitioning always exists: a trivial partitioning simply has n partitions, where each partition has a single row. Obviously, there exists a way of partitioning the rows of M such that the *number* of partitions is *minimized*. This motivates the following optimization problem.

The minimum perfect phylogenetic forest (MPPF) problem Given a binary matrix M, find a set of a *minimum* number of perfect phylogenies that derives M such that each row is derived by some perfect phylogeny and for any site s, mutations at s occur at most once in at most one tree. We denote the minimum number of perfect phylogenies $F_{min}(M)$.

There is a subtle issue about site mutations. When we construct an ARG for the entire data, we need to let each site mutate *exactly* once (assuming the site has both 0 and 1 values). Therefore, requiring each site to mutate at most once is equivalent to requiring each site mutate exactly once. This, however, may lead to potential problem when we construct a set of trees, rather than a single tree because there are multiple root sequences and the root sequences can introduce new states at sites without the need of site mutations. We now show that the two notions, "mutating at most once per site" and "mutating exactly once", are equivalent for the forest bound. To see this, consider a minimum forest under "mutating at most once per site" assumption. Then we can simply add a new *Steiner* sequence by utilizing the un-used mutations (if any). Here, we call a node (i.e. a sequence) Steiner node if this node or sequence does not appear in input data M. The resulting forest follows the "mutating exactly once" assumption and can not be smaller than the original due to the minimality assumption. Thus, the minimum forest under "mutating exactly once" assumption is not larger than that under "mutating at most once per site" assumption. On the other hand, "mutating at most once per site" contains "mutating exactly once", which implies that the minimum forest under "mutating exactly once" assumption is not smaller than that under "mutating at most once per site" assumption. Thus, the two assumptions are equivalent for the forest bound. In the following, we assume each site mutates *exactly* once in the forest.

Note that $F_{min}(M) = 1$ iff M has a perfect phylogeny. That is, there exists a single tree that derives all sequences in M iff M has a perfect phylogeny. On the other hand, when there is no perfect phylogeny for M, we need more than one tree to derive all the sequences in M. The MPPF problem asks to find the minimum number of trees in the forest. This problem is loosely related to the well-studied maximum parsimony

Table 1 One way to partition this data into two perfect phylogenies is to have rows r_1, r_2 and r_5 in one part and the rest in the other part. On the other hand, we can not put r_1, r_2, r_5 and r_6 in the same partition due to the four-gamete test

r_1	0	0	1	0	0
r_2	1	0	1	0	0
r_3	0	1	0	0	0
r_4	0	1	0	0	1
r_5	1	0	0	0	0
r_6	0	1	0	1	1
r_7	0	0	0	1	1

problem (i.e. the Steiner tree problem in phylogeny). In maximum parsimony, we construct a *single* tree (with back or recurrent mutations) which minimizes the number of site mutations. The MPPF problem asks for constructing the minimum number of trees, each of which is a perfect phylogeny, and each site can mutate once in at most one tree.

Now we define the forest bound.

Forest bound For a binary matrix M, the forest bound is equal to $F_{min}(M) - 1$.

We first illustrate the MPPF problem by the simple example in Table 1. Here, we want to split the matrix into partitions, each of which has a perfect phylogeny, i.e. does not violate four-gamete test. Also we require a site can mutate in at most one partition.

Lemma 3.1 shows that the forest bound is indeed a lower bound on the minimum recombination.

Lemma 3.1 *The forest bound is a valid lower bound on* $R_{min}(M)$.

Proof Suppose we trim a minARG N by removing all recombination edges in N and we have a forest with $k \geq F_{min}(M)$ trees. Note that N is connected and we need at least one recombination to connect a tree to the rest of trees. So $R_{min}(M) \geq k - 1 \geq F_{min}(M) - 1$. The reason that we subtract 1 is because we can start from a tree and this tree is the starting point of the network. □

Now we explain the reason for our interest in the forest bound. As mentioned above, the history bound lacks a static definition, unlike the forest bound. Below we show that the forest bound is higher than the haplotype bound but lower than the history bound, and hence, the forest bound is provably the highest lower bound that we know of which has a simple static definition. Recall that we call nodes (sequences) in a tree *Steiner* if the sequences do not appear in M.

Lemma 3.2 *For a perfect phylogenetic forest* $\mathcal{F}$ *with* n_s *Steiner nodes, the number of trees* k *is equal to* $n + n_s - m$.

Proof Suppose each tree $T_i \in \mathcal{F}$ contains n_i distinct sequences (nodes) for $i = 1 \ldots k$. Here, $\sum_{i=1}^{k} n_i = n + n_s$, where n_s is the number of Steiner nodes in the forest. We know for each tree T_i, there are $n_i - 1$ edges with mutations. Let m_i denote the number of mutations in tree T_i. This means $m_i = n_i - 1$. Here we assume there is

exactly one mutation per site in the forest. As described earlier, this assumption does not change the forest bound. So we have $m = \sum_{i=1}^{k} m_i = \sum_{i=1}^{k} (n_i - 1) = n + n_s - k$ mutations in the forest. So, $k = n + n_s - m$. □

The following shows an important property of a phylogenetic forest.

Lemma 3.3 *The forest bound applied to all of M is at least as large as the forest bound applied to a subset of sites in M.*

Proof For a given data M, suppose we have a minimum phylogenetic forest $\mathcal{F}$ for M with $F_{min}(M)$ trees. Now we consider $\mathcal{F}(S)$ when we restrict our attention to S, a subset of sites. To derive $\mathcal{F}(S)$ from $\mathcal{F}$, we remove all mutation sites in $\mathcal{F}$ that are not in S and cleanup the forest by removing edges with no mutations, and collapsing identical sequences. It is important to note that $\mathcal{F}(S)$ has *at most* $F_{min}(M)$ trees. This is because when we remove sites not in S, we may need to link up two previously disjoint trees (and thus make the number of trees smaller), but we can never *increase* the number of trees. Thus, we know $F_{min}(M(S))$ can not be higher than $F_{min}(M)$. □

Lemma 3.4 *This forest bound is always higher than the haplotype bound, but lower than the history bound.*

Proof We first show that the forest bound is provably higher than the haplotype lower bound. Given a matrix M, we first consider the submatrix consisting of n unique rows and m unique columns of M. Suppose a minimum forest has $k = F_{min}(M)$ trees. From Lemma 3.2, $k = n + n_s - m \geq n - m$. So $k - 1 \geq n - m - 1$, which is the haplotype bound. Furthermore, due to Lemma 3.3, the forest bound applied the entire M is at least as large as the forest bound applied to the submatrix. Therefore, the forest bound is always higher than the haplotype bound on matrix M.

Now we show that the history bound is higher than the forest bound. From the algorithm to compute the history bound, it can be seen that the method produces a phylogenetic forest. However, in contrast to the definition of a phylogenetic forest given above, the forest produced by the history bound has additional time-order constraints: the trees in the forest can be time-ordered such that if site s mutates in a tree T_i, the states at s for sequences in earlier trees must be ancestral states (i.e. not the derived states). But since the forest produced by the history bound is a valid phylogenetic forest, the number of trees in that forest produced by the history bound cannot be smaller than that from the definition of the forest bound. □

We now relate the forest bound to the optimal haplotype bound.

Theorem 3.5 *The forest bound is higher than the optimal haplotype bound.*

Proof By Lemma 3.4 we know that the forest bound applied to any subset of sites is higher than the haplotype bound applied to the same subset of sites. In particular, if S^* is the subset of sites of M (called *optimal subset*) that gives the optimal haplotype bound, then the forest bound applied to S^* is higher. By Lemma 3.3, we know that

Table 2 Example where the optimal haplotype bound is smaller than the forest bound

1	0	0	0	1
0	0	0	1	0
0	0	1	0	0
1	1	0	1	1
0	1	1	0	1

the forest bound applied to all of M is at least as large as the forest bound applied to a subset of sites in M, which in turn is at least as large as the optimal haplotype bound. □

Theorem 3.5 and Lemma 3.4 say that the forest bound is higher than the optimal haplotype bound but lower than the history bound. Hence we conclude,

Corollary 3.6 *The optimal haplotype bound cannot be higher than the history bound.*[2]

Experiments show that the forest bound can be strictly higher than the optimal haplotype bound and improve the overall recombination lower bound. For example, consider the following matrix shown in Table 2. The optimal haplotype bound for this data is 1 by taking e.g. the first and the second sites, while it is not hard to see that a perfect phylogenetic forest contains at least three components. For example, suppose the first two rows belong to one tree T_1 and the rest three rows belong to the other tree T_2. Then, T_1 mutates at sites $1, 4, 5$, while T_2 mutates at $1, 3, 4, 5$. This is not a legal partition since site $1, 4, 5$ mutates two times (once in T_1 and once in T_2). Thus, the forest bound for this data is 2.

As mentioned earlier, it is known that the optimal haplotype bound and the history bound are both NP-hard to compute (Bafna and Bansal 2006). However, if the forest bound could be computed efficiently, we would not need to compute the optimal haplotype bound, but could instead use the forest bound. Unfortunately, the forest bound is also NP-hard to compute, which we now show.

3.2 The complexity of the forest bound

Theorem 3.7 *The MPPF problem is NP-hard.*

Proof The high-level construction of our proof is inspired by Foulds and Graham's NP-completeness proof of Steiner tree in phylogeny problem (Foulds and Graham 1982).

As in Foulds and Graham (1982), we reduce from the known NP-complete problem of Exact Cover by 3-sets (X3C) (Garey and Johnson 1979). Recall that the gen-

[2]Myers (2003) asserted (with no proof) that the history bound is higher than the optimal haplotype bound. Here we have furnished the proof to this claim.

eral form of X3C is as the following:

$$\mathcal{S} = \{S_1, S_2, \ldots, S_n\}, \quad \text{where each } |S_i| = 3 \text{ and}$$
$$S_i \subseteq \{1, 2, \ldots, 3m\} = I_{3m}, \text{ for } 1 \leq i \leq n.$$

Does $\mathcal{S}$ contain (non-overlapping) m sets $S_{i_1}, \ldots, S_{i_m}$ whose union is I_{3m}?

High-level idea We construct a binary matrix M for $\mathcal{S}$ (the collection of sets), such that for each set S_i, the set of corresponding sequences in M can be generated on a perfect phylogeny. Thus, if there is a solution for X3C, we have m perfect phylogenies that use up all site mutations, and a collection of isolated sequences (also trivially perfect phylogenies) and the total number of trees is $F_{min}(M)$. If there is no solution for X3C, the number of trees in any perfect phylogenetic forests is more than $F_{min}(M)$. To enforce this property, two sequences corresponding to the same set S_i will have a small Hamming distance. For two sequences corresponding to different sets, their Hamming distance will be large. So, if two far apart sequences are placed into the same tree, there will be too many Steiner sequences needed to connect them, and thus by Lemma 3.2 and proper manipulation of the construction, we will need more than $F_{min}(M)$ trees in such forest.

WLOG we assume there is no duplicate sets in $\mathcal{S}$. Given an instance of X3C, we construct a binary matrix M as follows. We let $K = m + 1$. Note that $2K - 3 > m$, when $m > 1$. For each S_i, we construct a set of $3K$ sequences of length $3mK$. We have K sequences corresponding to each of the three elements in S_i. Each of these sequences is composed of $3m$ blocks of K sites. Each block is arranged sequentially in the increasing order for each integer in S_i. The sequences are constructed as follows. Suppose we are constructing the j_{th} sequence ($j \in \{1 \ldots K\}$) for an element $p \in S_i$. For block (of number q) $B_{i,p,j,q}$ in this sequence, if the corresponding integer $q \notin S_i$, then block $B_{i,p,j,q}$ contains all 1. If $q \in S_i$ and $q \neq p$, then we set $B_{i,p,j,q}$ to be all 0. If $q \in S_i$ and $q = p$, we set the j_{th} bit in $B_{i,p,j,q}$ to 1 and 0 for all other bits. Note that for a given row associated to a set S_i, all bits corresponding to elements not in this set are 1. Also note that for the K sequences corresponding to an integer $q \in S_i$, the K blocks $B_{i,p,j,q}$ form a diagonal matrix with all 1 on the main diagonal. One example is shown in Table 3 for the simple case when $m = 2$.

The following facts (proof omitted) about M are important.

P1. There are *no* two identical sequences in M.
P2. The $3K$ sequences corresponding to a single set S_i have a star-shaped perfect phylogeny, with the center sequence as the only Steiner sequence.
P3. For two sequences s_1, s_2 coming from the same set S_i, the Hamming distance between s_1, s_2 is 2. For two sequences s_1, s_2 coming from different sets S_i, S_j, the Hamming distance is at least $2K - 2$.

Now we claim that *X3C* problem has a solution (i.e. union of $S_{i_1}, \ldots, S_{i_m}$ is equal to I_{3m}) if and only if there is a phylogenetic forest for M with exactly $3nK - 3mK + m$ perfect phylogenies.

We first show that given a solution (i.e. $S_{i_1}, \ldots, S_{i_m}$) of X3C, we can build a forest with $3nK - 3mK + m$ trees. From property P2, we construct m perfect phylogenies,

Table 3 Example of the constructed matrix when $m = 2$ (i.e. there are 6 elements in I_{3m}), and thus $K = 3$. The table lists the constructed rows for a set $\{1, 2, 4\}$

	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6
r_1	1	0	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1
r_2	0	1	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1
r_3	0	0	1	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1
r_4	0	0	0	1	0	0	1	1	1	0	0	0	1	1	1	1	1	1
r_5	0	0	0	0	1	0	1	1	1	0	0	0	1	1	1	1	1	1
r_6	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	1	1
r_7	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1
r_8	0	0	0	0	0	0	1	1	1	0	1	0	1	1	1	1	1	1
r_9	0	0	0	0	0	0	1	1	1	0	0	1	1	1	1	1	1	1

each from $3K$ sequences corresponding to each of S_{i_j}. Then, we treat the remaining $3nK - 3mK$ as isolated sequences (trivially perfect phylogenies). So the total number of perfect phylogeny is $3nK - 3mK + m$.

Now we show the other direction: if there is a phylogenetic forest for M that has $3nK - 3mK + m$ perfect phylogenies, then there is a solution for problem X3C. We first argue that no two sequences from different sets S_i, S_j can appear together in a same perfect phylogeny. For contradiction, suppose s_1, s_2 coming from different sets are together in one perfect phylogeny. Consider the path from s_1 to s_2 in the perfect phylogeny. From Property P3, the Hamming distance between s_1 and s_2 is at least $2K - 2$. This means there are at least $2K - 3$ intermediate nodes on that path whose states changes from s_1 to s_2 on these $2K - 2$ sites. It is also easy to see that none of these $2K - 3$ nodes can be part of M, since each sequence in M has either exactly a single 1 or all 1s within a block and intermediate nodes between s_1, s_2 must contain from 2 to $K - 1$ 1s for the block corresponding to the element not shared by S_i, S_j. That is, we know that the phylogenetic forest contains at least $2K - 3$ Steiner nodes. But from Lemma 3.2, we will have at least $3nk - 3mK + (2K - 3)$ perfect phylogenies, which is larger than $3nK - 3mK + m$ since $2K - 3 > m$. That is a contradiction, and thus each phylogeny can only have sequences derived from the same set S_i.

Now it is easy to see that within the forest there can be at most m non-degenerated trees. This is because each non-degenerated tree contains at least one Steiner node (see Property P3). Also note that we can not have fewer than m non-degenerated trees. To see this, suppose for contradiction, that we have at most $m - 1$ non-degenerate perfect phylogenies. Since each such tree comes from a single set S_i, if there are at most $m - 1$ trees, there are at most $3(m - 1)K$ nodes in the trees. Then there are at least $3nK - 3(m - 1)K = 3nK - 3mK + 3K$ degenerated trees. Since $K = m + 1$, we know we will have more than $3nK - 3mK + m$ (isolated) trees. That is a contradiction.

Therefore, we know we will have exactly m non-degenerated trees. Now we need to show that these m non-degenerated trees correspond to a solution for *X3C*. Note

Table 4 Example for the 2rMPPF problem. One way to partition this data into two perfect phylogenies is to have the upper four rows in one part and the rest in the other part. Note that different from the example given in Table 1, site 1 (for example) mutates once in the upper partition and once in the lower partition. This is allowed in the 2rMPPF problem

r_1	0	0	0
r_2	0	1	0
r_3	1	0	1
r_4	1	0	0
r_5	1	1	1
r_6	1	1	0
r_7	0	1	1
r_7	0	0	1

that each of such trees does correspond to a set in $\mathcal{S}$. What we need to show is that every element in I_{3m} is covered, and no element is covered more than once. Suppose a tree has a block whose corresponding integer is not covered by other picked sets, then we can easily enlarge the tree by adding the sequences of that block. Now we want to argue that no two sets picked by this phylogenetic forest can overlap. For contradiction, suppose there is an overlap between S_i, S_j when we select all $3K$ sequences corresponding to S_i, S_j. Then for the corresponding two trees, there must be mutations in *both* trees for the overlapped sites. This contradicts the assumption that the set of trees are perfect phylogenies with no duplicate mutations (sites). Therefore, the given phylogenetic forest leads to a valid X3C solution. □

Corollary 3.8 *Computing the forest bound is NP-hard.*

3.3 A variant of the MPPF problem

Note that the MPPF problem requires that if a mutation occurs at a site in one partition, then this site does not mutate in the other partitions. Now, suppose we relax the phylogenetic forest problem by allowing a site to mutate once in more than one perfect phylogenies (but still at most once in the perfect phylogeny of each partition). We call this problem relaxed minimum Perfect Phylogenetic Forest problem (or rMPPF problem), which might be of interests on its own. Here, we consider a special case of this problem, where we want to find a bi-partition of M such that each partition has a perfect phylogeny and the same site can mutate once in both partitions. We denote this problem as 2rMPPF problem. See the example in Table 4 for an illustration. The relaxed version of the phylogenetic forest problem is shown in the following example. One way to view the 2rMPPF problem is that in 2rMPPF, the two partitions are independent. That is, we only require each partition has a perfect phylogeny and the mutations used in one partition do not affect the other partition.

The following theorem shows the rMPPF problem is NP-complete even when we just want to partition the matrix into two perfect phylogenies.

Theorem 3.9 *The 2rMPPF problem is NP-complete.*

Proof In the following, we present a proof of NP-completeness of 2rPPF problem. It is easy to see 2rPPF is in NP. That is, given two sub-populations, we can check

Table 5 Choice gadget for x_i. The property of this gadget is that there is exactly one way (as shown above) to achieve perfect phylogeny for both sub-population, assuming gamete 00 is included for any two columns in each partition

1	0	0	1	1	1
0	1	0	1	1	1
0	0	1	1	1	1
1	1	1	1	0	0
1	1	1	0	1	0
1	1	1	0	0	1

whether each of the two sub-populations has perfect phylogeny in linear time (Gusfield 1991). Then we need to show a reduction from an established problem to 2rPPF problem to complete the NP-completeness proof. We will reduce from the well-known NP-complete problem: NAESAT (not-all-equal SAT).

Recall that in NAESAT, we are given m clauses with three literals each. We shall construct a haplotype matrix M such that there is a way to partition M into two sub-populations (each having a perfect phylogeny) iff there is a truth assignment that makes at least one literal true and at least one literal false in each clause. We would call such partitioning a *perfect* partition. Suppose that the clauses are $C_1, C_2, \ldots, C_m$ and the variables appearing in them are $x_1, x_2, \ldots, x_n$. Also, in the following, when we say splitting or partitioning, we refer to the valid partitioning into two sub-populations, each with perfect phylogeny.

Before diving into the details, here is the high-level idea. We want to construct a set of (partial) haplotype rows H_{x_i} for variable x_i. H_{x_i} is constructed in such a way that the rows in this set must be separated into two sub-populations evenly to make the sub-population having perfect phylogeny. It is also important to ensure that there is only one way to partition H_{x_i} to achieve perfect phylogenies. Intuitively, the way of dividing H_{x_i} decides the boolean value for x_i. We then extend the constructed haplotypes by adding more *sites*, which correspond to the clauses. These new sites are constructed in a way that we create specific patterns for rows corresponding to the three variables in the clauses. And to achieve perfect phylogeny, these haplotypes corresponding to the three literals can not be located inside a single sub-population. This helps to ensure the not-all-equal condition in the NAESAT problem.

We begin with the (simplified) choice gadget for variable x_i. See Table 5 for illustration on the gadget H_{x_i}. Note that the generalized choice gadget is composed of four partitions: the upper left and the lower right are simply the 3 by 3 identity matrices, and the lower left and upper right are matrices with 3 by 3 all-1. Note also that we can construct generalized choice gadgets of any size.

It is easy to see the haplotypes in H_{x_i} do not have perfect phylogeny, assuming gamete 00 is included in each partition. Thus, in any partitioning, H_{x_i} can not reside inside a single partition. How many ways of partitioning H_{x_i} are there (so that both partitions allow perfect phylogenies)? We claim there is exactly a single way (three upper rows in one partition and three lower rows in the other partition, as shown in Table 5), assuming that gamete 00 is always present for any pair of sites (like a root-known perfect phylogeny case). Why? We call the three rows in the upper partitions as top rows, and the other three rows as bottom rows. If the rows are not divided as shown in the table, then there must be three rows in one partition that are not all top (resp. bottom) rows. But then it is easy to see there must be incompatibility (recall

Table 6 Example of constructing a choice gadget for a x_i by replicating five times. Here, I_3 stands for a 3 by 3 identity matrix. 1_3 is a 3 by 3 matrix filled with all 1. 0 stands for an all-zero 3 by 3 matrix. The leftmost column provides index for the combo rows of triple rows

1	I_3	0	0	0	0	0	0	0	0	1_3
2	0	I_3	0	0	0	1_3	0	0	0	0
3	0	0	I_3	0	0	0	1_3	0	0	0
4	0	0	0	I_3	0	0	0	1_3	0	0
5	0	0	0	0	I_3	0	0	0	1_3	0
6	1_3	0	0	0	0	I_3	0	0	0	0
7	0	1_3	0	0	0	0	I_3	0	0	0
8	0	0	1_3	0	0	0	0	I_3	0	0
9	0	0	0	1_3	0	0	0	0	I_3	0
10	0	0	0	0	1_3	0	0	0	0	I_3

that we always have gamete 00) in that sub-population. Also note that there are no incompatibility in the shown partitioning.

But we are not done with choice gadgets yet. To avoid complexities caused by the same variable appearing in several clauses, we use a trick of *replicating*. We will replicate each variable the number of times that the variable appears in all these clauses. The goal is the decouple between the clause gadgets (see below) such that there would be no incompatibility between two clause gadgets. Note that the constructed matrix size after replicating is still polynomial size regarding to the input.

An example of replicating a variable 5 times is shown in Table 6.

Note that we define combo rows are three neighboring rows in the data. As shown in Table 6, we can essentially create copies of H_{x_i} such that each copy needs to make synchronized choices. The purpose of these replicated choice gadgets is that we now can use different segments of choice gadgets in different clause gadgets. By doing this, we can now safely decouple two clause gadgets because we ensure different clause gadgets never share choice gadget segments.

Here, we must show that this generalized choice gadget is still a choice gadget. That is, we need to prove that there is exactly one way of partitioning the generalized choice gadget (i.e. the upper half and lower half) s.t. the two partitions are perfect phylogeny. For better presentation, we only prove this for the example in Table 6. We comment that the proof can be trivially extended to generalized choice gadgets of any size.

Lemma 3.10 *There is a unique way of perfect bi-partitioning the data in Table 6. That is, partitioning by the upper half and lower half.*

Proof The claimed partition is trivially a solution. We now need to show that there is no other solution for perfect bi-partitioning.

We first prove the claim for the case where there is no separation of rows in any of the 3×3 submatrices (i.e. the three rows in such submatrices belong to a single partition). WLOG assume combo row 1 are in partition A. Since combo row 6 has all-1 at first three sites (where combo row 1 has 01, 10 gametes). This immediately implies combo rows 6 are in partition B. Then, from combo rows 6, combo row 2 must be in partition A, and so on. So, combo rows 1, 2, 3, 4, 5 are in partition A while the other rows are in partition B. But this is exactly the claimed unique partition.

Table 7 How to arrange the constructed choice gadgets. Here, H_{x_i} is the generalized choice gadget for variable x_i

$x_1, \overline{x_1}$	H_{x_1}	0	0
$x_2, \overline{x_2}$	0	H_{x_2}	0
$x_3, \overline{x_3}$	0	0	H_{x_3}

Table 8 Gadgets for clauses. Here is an example for clause $(x_i, x_j, \overline{x_k})$. We use 0 to indicate a 3-cells as all 0. $1_{3,1}$ stands for a 3×1 submatrix filled with all-1. It is important to note that these 3×1 submatrices are aligned to the 3×3 submatrices in the choice gadgets

	For x_i	For x_j	For $\overline{x_k}$
x_i	0	$1_{3,1}$	$1_{3,1}$
$\overline{x_i}$	0	0	0
x_j	$1_{3,1}$	0	$1_{3,1}$
$\overline{x_j}$	0	0	0
x_k	0	0	0
$\overline{x_k}$	$1_{3,1}$	$1_{3,1}$	0

Now we consider the situation when some comb row (say combo row 1 WLOG) is divided into partition A and B. WLOG we further assume two rows in combo row 1 belongs to partition A. This implies *whole* combo row 6 is in partition B. This in turn implies whole combo row 2 is in partition A, and so on. Finally, we know whole combo row 10 is in partition B. However, there is a single row of combo row 1 in partition B, and this causes incompatibility among sites in the I_3. Contradiction.

Thus, there is only a single way of perfect partitioning of the generalized choice gadget. □

Since we have enforced the unique perfect bi-partitioning of H_{x_i} for a *single* variable x_i, we let the top partition correspond to literal x_i, and bottom partition for literal $\overline{x_i}$. Now we show how we arrange these H_{x_i} for *all* variables. This is shown in Table 7.

As shown in Table 7, we arrange the gadgets in a diagonal way. This can avoid interference between choice gadgets for different variables. Also note that by filling all 0 bits in, we achieve the previously assumed 00 gamete, because both partition must have at least one 00 gamete since H_{x_i} (on the diagonal) is not perfect phylogeny. An important thing to note is that there is no incompatibility between two sites coming from different choice gadget. This is due to the isolation caused by diagonal arrangement: gamete 11 is missing for all such pairs.

With these choice gadgets in place, we are ready to handle the clauses. From now on, we are not going to add new rows. We will only add columns (sites) to the existing matrix for the clauses. For a clause, say $C_i = (x_i, x_j, \overline{x_k})$. We will create a block of new sites for each of these literals. We show this by Table 8. Since there are other rows corresponding to variables not appearing in c_i, we fill in all 0 again. Note that this means in any partitioning, we have gamete 00 in both partitions.

Let us look closely at the clause gadget. We already know each set of rows corresponding to a literal must belong to a single partition. Therefore, we can treat the 3×1 submatrices of all-1 (resp. 0) as a single letter 1 (resp. 0) since they must belong to a single partition. Since we always have gamete 00 for pair of sites in this clause gadget, it is easy to see the three sites in this clause gadget are pairwise incompatible.

Thus, any partitioning must not have the three non-zero combo rows together in one sub-population. Conversely, any way of separating the non-zero combo rows leads to no-incompatibility within this clause gadget. Therefore, we know that we can achieve no-incompatibility within the clause gadget iff the three literals are not equal in the truth assignment for the clause.

It can also be shown that there are no incompatibility between sites in any partitioning that enforces the choices gadgets. That is, there is not extra constraints imposed by our construction. It is easy to see that there is no incompatibility between two clause gadgets. Also there is no incompatibility between sites in generalized choice gadgets for any variable assignments. What remains to show is to ensure that there is no incompatibility between sites in a choice gadget and clause gadget. It is easy to see that any site in a clause gadget is compatible with sites in a choice gadget for x_i if x_i (or $\overline{x_i}$) does not appear in the clause (due to missing of gamete 11). Now suppose x_i does appear in the clause. The key observation is that for any site s in a choice gadget, it either contains a single 1 or a block of three 1. The first case leads trivially to compatibility. The second case also works because we assign *all* three values to 1 in the clause gadget. Thus, we can never have gametes 11 and 10 between s and a site s_c in a clause gadget at the same time. So we conclude that there is a way of partitioning perfectly iff there is a not-all-equal truth assignment.

To summarize, our construction of matrix M ensures that there is a way of partitioning M into perfect phylogenies iff there is a not-all-equal truth assignment for the clauses. □

4 Practical computation of the forest bound

Now we use integer programming to compute the *exact* forest bound for data within certain range.

4.1 Computing the forest bound precisely using integer linear programming

Consider an input matrix M with n rows and m sites. Our goal is to compute the minimum forest that derives the input sequences. There are 2^m possible sequences (which form a hypercube) that could be part of the minimum forest. Of course, the n input sequences must appear in this forest. From Lemma 3.2, in order to compute the forest bound we need to *minimize* the number of Steiner nodes. Thus, we create a variable v_i for each sequence s_i in the set of 2^m possible sequences at Steiner nodes, where $v_i = 1$ means sequence s_i appears in the forest. Next, we create a variable $e_{i,j}$ for two sequences s_i, s_j that differ at exactly one column. We create constraints to ensure $e_{i,j} = 1$ *implies* $v_i = 1$ and $v_j = 1$. We define a set E_c as the set of $e_{i,j}$ where s_i, s_j differ exactly at the single site c. The infinite sites mutation model requires that exactly one $e \in E_c$ has value 1.

Optimization goal Minimize $(\sum_{i=1}^{2^m} v_i) - m - 1$
Subject to
$v_i = 1$, for each row $s_i \in M$.
$e_{i,j} \leq v_i$, and $e_{i,j} \leq v_j$, for each edge (s_i, s_j).

$\sum_{(s_i,s_j)\in E_c} e_{i,j} = 1$, for each site c

Binary Variables

v_i for each sequence s_i with m binary characters

$e_{i,j}$ for each pair of sequences s_i, s_j such that $d(s_i, s_j) = 1$.

The formulation can also be extended easily to handle the situation where there are missing values in the input data, which is important for handling real biological data. To handle missing data in the ILP formulation, for a sequence s_i with missing values, we change the constraint $v_i = 1$ to $\sum_j v_j \geq 1$, for each sequence s_j that matches the values of s_i at all non-missing positions. Our experience shows that the formulation can be solved reasonably fast for data with up to 8 sites (by a powerful ILP package CPLEX). Note that handling missing values is the advantage of the ILP formulation. The history bound can be computed in time $O(m2^n)$ but can not be easily extended to situations where there are missing values in the input data M. Although the number of variables in the ILP is $O(2^n)$, our experience shows that it is practical for certain range of data even when the data contains some missing values. See Sect. 4.2 for practical results on this issue.

4.2 Simulations of data with missing data

Now we describe computations of the forest bound and how they compare to the haplotype bound on simulated data. In this simulation study, to make comparison easier, we do not use the composite method (Myers and Griffiths 2003), which often gives higher lower bound. Initial experience shows that when composite bounds (instead of a single bound over the entire interval) are compared, the forest bound tends to agree more with the haplotype bound. This supports the general view that the optimal haplotype bound is a very good bound. It remains an interesting question on finding effective ways to give bounds higher than the composite bound from the optimal haplotype bounds on small intervals. We show here that the forest bound can be effectively computed for certain range of data with missing values.

We generated 100 datasets with Hudson's program MS (Hudson 2002) for each parameter setting. We fix the number of sites in these data to 7 or 8. We want to compare the forest bound with the optimal haplotype bound when the data contains various level of missing data. Currently, the only known method computing optimal haplotype bound with missing data can only work with very small data. So instead, we compare with a weaker haplotype bound method, which is implemented in program HapBound (Song et al. 2005). HapBound can handle missing data but not always give the optimal bound (Song et al. 2006) when there is missing data. Missing values are added to the datasets by setting an entry to be missing with a fixed probability P_{mv}.

Table 9 shows that the forest bound can outperform HapBound in some cases. Our results show that the haplotype bound method used by the program HapBound appears to be quite good for the range of data we tested. In particular, our tests show that when the missing value level is low or moderate, the program HapBound performs quite well for the range of data we generated.

We also simulated randomly generated data to see how the three lower bounds perform. Our results show that the forest bound is more likely to give higher bounds than the optimal haplotype bound on random data. For example, for randomly generated

Table 9 Comparing the forest bound with haplotype bound. We simulate 100 datasets. Then we introduce various level of missing values to these datasets. We report the percentage of datasets where the forest bound is strictly higher than the haplotype bound. That is, each value is the percentage of datasets where the forest bound is higher than the optimal haplotype bound for the specific setting (i.e. number of rows and columns, and percentage of the missing values). Note that the forest bound is always as high as the haplotype bound

% Missing value	0%	10%	20%	30%
20 rows, 7 sites	0%	0%	0%	3%
20 rows, 8 sites	0%	1%	0%	0%
30 rows, 7 sites	0%	1%	0%	8%
30 rows, 8 sites	0%	0%	0%	7%

data with 15 rows and 7 sites, the forest bound gives higher bound than the optimal haplotype bound in 12% of the random data, while the history bound gives higher bound than the optimal haplotype bound in 18% of the data. This suggests that as the sequences become less correlated, the forest bound tends to perform better than the optimal haplotype bound.

5 Discussion

In this paper, we present our understanding of a good lower bound on $R_{min}(M)$ (i.e. the minimum number of recombinations needed to derive a set of sequences M), namely the forest bound. We demonstrate the relationship between the forest bound and the other two known lower bounds: the haplotype bound and the history bound. Our work provides a unified view on all three lower bounds. This may help future research on the recombination lower bounds.

There are many unsolved problem related to computing good lower bound on $R_{min}(M)$. The ILP formulation can solve data with no more than 8 sites. This seems to make the forest bound impractical because the real haplotype data can be quite large, with hundreds of sites, and thus is well over the limit that the ILP formulation can solve. However, this method can still be practical because a common way of computing lower bounds is to compute local lower bounds on short intervals and then use a composite method to compute the overall bound (Myers and Griffiths 2003). Thus, the ILP formulation may be useful in computing local bounds for short intervals and thus improve the overall bound. Still, an interesting research question is how to compute the forest bound for data with larger number of sites, which would allow us to find out how the forest bound compares to the haplotype bound in those cases. Another possibility is to explore some good approximation of the forest bound, which can be easily computed.

Another interesting problem is to develop practical methods to compute the history bound when the data contains missing values. The current dynamic programming method to compute the history bound does not easily generalize to handle missing values. As mentioned earlier, one attractive feature of the ILP formulation for the forest bound is its ability to handle missing values. It will be interesting to see how we can address the missing value issue in history bound methods.

References

Bafna V, Bansal V (2004) The number of recombination events in a sample history: conflict graph and lower bounds. IEEE/ACM Trans Comput Biol Bioinf 1:78–90

Bafna V, Bansal V (2006) Inference about recombination from haplotype data: lower bounds and recombination hotspots. J Comput Biol 13:501–521

Bordewich M, Semple C (2004) On the computational complexity of the rooted subtree prune and regraft distance. Ann Comb 8:409–423

Foulds LR, Graham RL (1982) The Steiner tree in phylogeny is NP-complete. Adv Appl Math 3

Garey M, Johnson D (1979) Computers and intractability. Freeman, San Francisco

Griffiths RC, Marjoram P (1996) Ancestral inference from samples of DNA sequences with recombination. J Comput Biol 3:479–502

Gusfield D (1991) Efficient algorithms for inferring evolutionary history. Networks 21:19–28

Gusfield D, Eddhu S, Langley C (2004) Optimal, efficient reconstruction of phylogenetic networks with constrained recombination. J Bioinf Comput Biol 2:173–213

Gusfield D, Hickerson D, Eddhu S (2007) An efficiently-computed lower bound on the number of recombinations in phylogenetic networks: theory and empirical study. Discrete Appl Math 155:806–830

Hudson R (2002) Generating samples under the Wright-Fisher neutral model of genetic variation. Bioinformatics 18(2):337–338

Hudson R, Kaplan N (1985) Statistical properties of the number of recombination events in the history of a sample of DNA sequences. Genetics 111:147–164

Myers S (2003) The detection of recombination events using DNA sequence data. PhD dissertation, Dept of Statistics, University of Oxford, Oxford, England

Myers SR, Griffiths RC (2003) Bounds on the minimum number of recombination events in a sample history. Genetics 163:375–394

Song YS, Ding Z, Gusfield D, Langley C, Wu Y (2006) Algorithms to distinguish the role of gene-conversion from single-crossover recombination in the derivations of SNP sequences in populations. In: Proceedings of RECOMB 2006. LNBI, vol 3909

Song YS, Wu Y, Gusfield D (2005) Efficient computation of close lower and upper bounds on the minimum number of needed recombinations in the evolution of biological sequences. Bioinformatics 421:i413–i422. Proceedings of ISMB 2005

Wang L, Zhang K, Zhang L (2001) Perfect phylogenetic networks with recombination. J Comput Biol 8:69–78