Improved Algorithms for Inferring the Minimum Mosaic of a Set of Recombinants

Yufeng Wu and Dan Gusfield

Department of Computer Science
University of California, Davis
Davis, CA 95616, U.S.A.
{wuyu,gusfield}@cs.ucdavis.edu

Abstract. Detecting historical recombination is an important computational problem which has received great attention recently. Due to recombination, input sequences form a mosaic, where each input sequence is composed of segments from founder sequences. In this paper, we present improved algorithms for the problem of finding the minimum mosaic (a mosaic containing the **fewest** ancestral segments) of a set of recombinant sequences. This problem was first formulated in [15], where an exponential-time algorithm was described. It is also known that a restricted version of this problem (assuming recombination occurs only at predefined block boundaries) can be solved in polynomial time [15,11]. We give a polynomial-time algorithm for a special case of the minimum (blockless) mosaic problem, and a practical algorithm for the general case. Experiments with our method show that it is practical in a range of data much larger than could be handled by the algorithm described in [15].

1 Introduction

A grand challenge for post-genomic era is dissecting the genetic basis of complex diseases. An important connection between the sequences (the genotypes) and the traits of interest (the phenotypes) is the evolutionary history (the genealogy) of the chosen individuals. Thus, inferring genealogy from sequences has received much attention recently. A major difficulty in inferring genealogy is meiotic recombination, one of the principal evolutionary forces responsible for shaping genetic variation within species. Efforts to deduce patterns of historical recombination or to estimate the frequency or the location of recombination are central to modern-day genetics.

A central genetic model used throughout this paper (and used before in [15]) is that the current population evolved from a *small* number of founder sequences. Over time, recombination broke down ancestral sequences and thus a current sequence is a concatenation of segments from the founder set. The set of input sequences then looks like a **mosaic** of segments from the founder sequences, when sequences are arranged as aligned rows. Thus, we refer the model as the *mosaic model*. See Figure 1 for an illustration. The biological literature contains

B. Ma and K. Zhang (Eds.): CPM 2007, LNCS 4580, pp. 150–161, 2007.

validations of this model. For example, it is stated in the Nature paper [14] that "The *Ferroplasma* type II genome seems to be a composite from three ancestral strains that have undergone homologous recombination to form a large population of mosaic genomes."

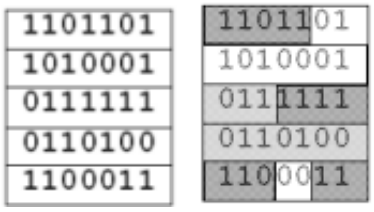
The mosaic pattern is potentially very informative in understanding the population evolution. The mosaic tells which sequences inherit their DNA from the same founder at a genomic site, and thus can be very useful in understanding the genetic basis of traits. In the context of inferring haplotypes from genotypes (i.e. the haplotype inference problem, also called phasing problem), Hidden Markov Model (HMM) based probabilistic approaches which exploit the mosaic patterns have been actively studied [3,9,10]. Therefore, understanding the genomic mosaic structure is an interesting problem, and better understanding of the mosaic pattern may be useful for population genetics problems.

Unfortunately, the mosaic boundaries (called *breakpoints*) are not readily seen from the sequences, and so we have the problem of inferring the true breakpoints (and the ancestral founder sequences) for the given input sequences. The breakpoints break the given sequences into segments of (possibly inexact) copies of ancestral materials that are inherited from some founder sequences of the population. The inexact copies of ancestral materials are often due to point mutations at nucleotide sites. In the context of recent human populations, however, the assumption is that the time period is short and the point mutation rates are low [15,11,1,2]. Hence, we assume throughout the paper that the input sequences inherit *exact* copies of ancestral material between two neighboring breakpoints. So every input sequence is a concatenation of segments of some founder sequences. Since there are a huge number of possible mosaic patterns for a set of input sequences, we need a biologically meaningful model to infer breakpoints and founders.

In 2002, Ukkonen [15] proposed a computational problem based on the mosaic model, given input of n binary sequences with m columns each. The model assume that the population evolves from a set of relatively small number of founders. The natural parsimonious objective is to construct a mosaic with fewest breakpoints. This motivates the following optimization problem.

The Minimum Mosaic Problem . Given n input sequences (each with m columns) and a number K_f, find K_f founder sequences that *minimize* the total number of breakpoints needed to be put in the input sequences, which break the input sequences into segments from the founder sequences. See Figure 1 for an example. The Minimum Mosaic Problem has also been turned into a graphical game, called the Haplotye Threading Game, developed at the University of North Carolina [5].

It is important to emphasize that we require each segment to be derived from the corresponding aligned positions of a founder sequence, although the breakpoints do not need to be the same in each of the input sequences. Also note that once founder sequences are known, it is straightforward (using e.g. a method in [12]) to place breakpoints in the input sequences, so that the number of breakpoints is minimized for *each* sequence and thus also for all input sequences together.



(a) Haplotypes (b) Mosaic

Fig. 1. An example illustrating the minimum mosaic problem on binary sequences. Figure 1(a) shows the input sequences. Figure 1(b) shows one way to partition the sequences in Figure 1(a) into segments, such that each segment comes from one of *three* founders: 0110100, 1101111 and 1010001. Note that there are totally four breakpoints, which is the minimum over all possible solutions with three founders.

In [15] (and also [11]), efficient algorithms were developed for a related but different problem. In addition, Ukkonen [15] also described a dynamic programming algorithm for the minimum mosaic problem described above. But the algorithm given in [15] does not scale well when the number of founders or the size of input matrix grows. Another unaddressed question is how to deal with *genotypes* (to be defined later), since most current biological data comes in the form of genotypes.

Our contributions. This paper focuses on the combinatorial properties of the minimum mosaic problem, on which *little* progress has been made since the work of Ukkonen [15]. We report on two main results.

1. For the special case where there are *two* founders, we show the minimum mosaic problem can be solved in $O(mn)$ time [1]. We also give an efficient algorithm for finding the minimum breakpoints when the input sequences consist of *genotype* data (instead of haplotype data).
2. For the general minimum mosaic problem, we present an efficiently computable lower bound on the minimum number of breakpoints. We also develop an algorithm which solves the minimum mosaic problem exactly. Simulations show that this method is practical when the number of founders is small, and the numbers of rows and columns are moderate.

1.1 Additional Definitions

In diploid organisms (such as humans) there are two (not completely identical) "copies" of each chromosome, and hence of each region of interest. A description of the data from a single copy is called a *haplotype*, while a description of the conflated (mixed) data on the two copies is called a *genotype*. Today, the underlying data that forms a haplotype is usually a vector of values of m *single nucleotide*

[1] Note that the algorithm proposed by Ukkonen [15] is also implicitly polynomial-time when $K_f = 2$. The advantage of our method is that we establish an easily-verified condition to construct a minimum mosaic for two founder sequences.

polymorphisms (SNP's). A SNP is a single nucleotide site where exactly two (of four) different nucleotides occur in a large percentage of the population. Genotype data is represented as n n by m 0-1-2 (ternary) matrix G. Each row is a genotype. A pair of binary vectors of length m (haplotypes) *generate* a row i of G if for every position c both entries in the haplotypes are 0 (or 1) if and only if $G(i, c)$ is 0 (or 1) respectively, and exactly one entry is 1 and one is 0 if and only if $G(i, c) = 2$.

Given an input set of n genotype vectors (i.e. matrix) G of length m, the *Haplotype Inference (HI) Problem* is to find a set (or matrix) H of n pairs of binary vectors (with values 0 and 1), one pair for each genotype vector, such that each genotype vector in G is generated by the associated pair of haplotypes in H. H is called an "HI solution for G". Genotype data is also called "unphased data", and the decision on whether to expand a 2 entry in G to $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ or to $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ in H, is called a "phasing" of that entry. The way that all the 2's in a column (also called a site) are expanded is called the phasing of the column (*site*). Note that if a genotype is 02 at two sites, we know the two haplotypes in an HI solution will be 00 and 01 at these two positions. We sometimes call this case *trivial* for these two sites. If a genotype is 22 instead, the HI solutions are *ambiguous*: an HI solution may be either 00/11 or 01/10 at the two sites.

Each input row r inherits a state at a site s from a particular founder. We say this founder is ancestral to r at site s.

2 The Two-Founder Case

We consider the special case where there are only two founders. Note that for *any* haplotype data H, there exists two founders that can derive H in a mosaic [15]: a trivial set of two founders consists an all-0 sequence and an all-1 sequence. However, it is not immediately clear which pair of founder sequences leads to a minimum mosaic.

2.1 Solution for Haplotype Data Input

For a haplotype matrix H at two sites s_i and s_j, there are four possible states (called gametes): 00, 01, 10, 11. We use $n_{i,j,g}$ to denote the number of times that a gamete g appears in H for two sites s_i and s_j. As an example, for the first two sites (sites 1 and 2) of the data in Figure 1, $n_{1,2,00} = 0$, $n_{1,2,01} = 2$, $n_{1,2,10} = 1$ and $n_{1,2,11} = 2$.

It is easy to see that we can remove from input sequences any site that is uniform (i.e. either all 0 or all 1). This will not reduce the minimum number of breakpoints in a minimum mosaic. Hence we assume there are two *founder* states at any site, one is 0 and the other is 1. We can also remove any site i which is identical to site $i + 1$. A key observation is: at two neighboring sites s_i, s_{i+1}, we will have either $00, 11$ gametes or $01, 10$ gametes for the two *founder* sequences. We define the *distance* between a sites s_i and its neighboring site to

the right s_{i+1} in H as $d_i = MIN(n_{i,i+1,00} + n_{i,i+1,11}, n_{i,i+1,10} + n_{i,i+1,01})$. We have the following simple lemma.

Lemma 1. *The minimum number of breakpoints between two neighboring sites s_i, s_{i+1} is at least d_i.*

Proof. Since the two founders have either 00/11 or 01/10 gametes at sites s_i and s_{i+1}, either there is a breakpoint between s_i and s_{i+1} for every gamete 01 and 10 (if the founders have 00/11 states at sites s_i and s_{i+1}), or between s_i and s_{i+1} for every gamete 00 and 11 (if the founders have 01/10 states). □

The above lemma implies that the minimum number of breakpoints is at least $n_{tb} = \sum_{i=1}^{m-1} d_i$. On the other hand, the following algorithm finds two founders that derive the input sequences with exactly n_{tb} breakpoints.

Algorithm 1. Polynomial-time algorithm for finding two founders F_1, F_2 that gives the minimum number of breakpoints

1. Let $F_1[1] \leftarrow 0$, and $F_2[1] \leftarrow 1$. And set $i \leftarrow 1$.
2. while $i \leq m - 1$
2.1. If $n_{i,i+1,00} + n_{i,i+1,11} \geq n_{i,i+1,10} + n_{i,i+1,01}$, then $F_1[i+1] = F_1[i]$, and $F_2[i+1] = F_2[i]$.
2.2. Otherwise, $F_1[i+1] = 1 - F_1[i]$, and $F_2[i+1] = 1 - F_2[i]$.
2.3 $i \leftarrow i + 1$

It is easy to verify that the above algorithm produces two founder sequences using exactly n_{tb} breakpoints. Intuitively, Algorithm 1 gives the optimal solution to the minimum mosaic problem by constructing founders from left to right. At each position (other than the leftmost site, i.e. s_1) the algorithm is only constrained by the single site to its immediate left. This means it can *always* choose a state to introduce exactly d_i breakpoints for each s_i and s_{i+1}. Thus, the solution is optimal due to Lemma 1. The running time of the algorithm is $O(mn)$. Thus, we have:

Proposition 1. *When $K_f = 2$, the minimum mosaic problem can be solved for haplotype data H in $O(nm)$ time.*

2.2 Solution for Genotype Data Input

Now we consider genotypes (not haplotypes) as input. This problem is important because most currently available biological data is genotypic. With genotype data, the minimum problem can be formulated as follows.

The minimum mosaic problem with genotypes. Given a genotype matrix G and a number K_f, find an HI solution H and K_f founder sequences such that the number of breakpoints needed to derive H from the founders is minimized among *all* possible HI solutions and K_f founder sequences.

We give a polynomial-time algorithm for the special case of $K_f = 2$. We begin with a lemma which extends Lemma 1 to genotypes. Note that for two genotypic sites i and j, the possible states are: $00, 01, 10, 11, 02, 20, 12, 21$, and 22. Similar to the haplotype case, we denote the number of times that gamete g appears at two sites i and j as $n_{i,j,g}$. We define the *distance* between two genotypic sites s_i and its right neighbor s_{i+1} as $d_i^g = MIN(2n_{i,i+1,00} + n_{i,i+1,02} + n_{i,i+1,20} + 2n_{i,i+1,11} + n_{i,i+1,12} + n_{i,i+1,21}, 2n_{i,i+1,01} + n_{i,i+1,02} + n_{i,i+1,21} + 2n_{i,i+1,10} + n_{i,i+1,20} + n_{i,i+1,12})$.

Lemma 2. *The minimum number of breakpoints between two neighboring genotypic sites s_i, s_{i+1} is at least d_i^g.*

Proof. Note that d_i^g represents the *minimum* number of gametes 00/11 and gametes 01/10 for *all* possible ways of phasing these two sites. Note that one can always phase a "22" gamete at sites s_i and s_{i+1} to *agree* exactly with the two founders at s_i, s_{i+1}. Therefore, following the same idea in the proof of Lemma 1, it is easy to see d_i^g is a lower bound on the number of breakpoints between sites i and $i + 1$. □

Proposition 2. *When $K_f = 2$, the minimum mosaic problem with genotypes can be solved in $O(nm)$ time.*

Proof. Using a similar idea as in Proposition 1, the two-founder minimum mosaic problem can also be solved efficiently even when the input is genotypic. The number of minimum breakpoints is equal to $\sum_{i=1}^{m-1} d_i^g$. This can be done as follows.

We can construct two founders using a procedure similar to Algorithm 1. A small difference is that here we use the smaller term in d_i^g (rather than d_i) to decide whether to let founders have gametes 00/11 or 01/10. Now that we have constructed two founders, we derive an HI solution H as follows. We start from the leftmost site and move to the right by one site each time. We pick any feasible phasing for the leftmost site. Now we consider site s_{i+1} by assuming s_i has been properly phased. Note that the only ambiguous rows at sites s_i and s_{i+1} are those containing 22. We phase 22 so that the phased gametes agree with the founder states at sites s_i and s_{i+1}.

The only subtle issue left is whether there are will ever be an inconsistency during the process. That is, will we be prohibited from phasing s_{i+1} in the way described, due to the phasing of site s_i. But inconsistency will not occur. For one row r of G, suppose the above procedure dictates the two 2's in s_i, s_{i+1} are phased to 01 and 10. If column s_i (for row r) has been phased as $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ we phase s_{i+1} (for row r) as $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$. Otherwise, we phase s_{i+1} as $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$. In either case, we will produce the needed binary pairs in sites s_i, s_{i+1} for row r. □

2.3 The Minimum Mosaic Problem with Unknown Site Order

We consider here a variation of the two-founder minimum mosaic problem, where the linear order of the m sites is *unknown*. Formally, we define an optimization problem as follows.

The two-founder minimum mosaic with permutation problem. Given a matrix M, we want to find a permutation Π of the sites, and two founder sequences, such that with those founders and by ordering the sites according to Π, the number of breakpoints used is the minimum over all possible site permutations and all possible pair of founders.

A Biological motivation. One biological motivation for allowing site permutation is the *linkage mapping* problem, which is to find the true ordering of multiple loci on a chromosome. Linkage maps remain important for species which have not yet been sequenced. See the recent paper [13] for a discussion of current interest in linkage mapping and a detailed explanation of computational issues involved in linkage mapping. In order to infer the true site ordering, a natural approach is to find the ordering of sites, and a small set of founders, so that the number of needed breakpoints is minimized with that number of founders.

We establish an interesting connection to the metric traveling salesman problem, which implies a 1.5-approximation solution to the minimum mosaic with permutation problem. Details are omitted due to lack of space.

3 The Case of Three or More Founders

When the number of founders is at least three, we do not have a polynomial-time algorithm for the minimum mosaic problem, although we conjecture that there is one. In this section, we first describe an efficiently computable lower bound on the minimum number of breakpoints for any fixed K_f with haplotype data. We also develop an algorithm that solves the minimum mosaic problem exactly. In our testing, the method is practical for many problem instances when the number of founders is three or four and the size of input matrix is moderate (e.g. with 50 sequences and 50 sites).

3.1 Lower Bound on the Number of Breakpoints for Haplotype Data

We now describe a simple lower bound on the minimum mosaic problem, inspired by the "composite haplotype bound", a lower bound developed for a different recombination model [7]. Consider a binary matrix H. We collect the set S of distinct rows together with their multiplicities (denoted as $(s_i, n_i) \in S$). Here, n_i records the number of times s_i appears in the input. We order S so that $\{n_i\}$ is *non-increasing*. If $|S| \leq K_f$, then the lower bound (denoted B_m) on the minimum number of breakpoints is simply 0. Otherwise, $\sum_{i=K_f+1}^{|S|} n_i$ is a lower bound on the minimum number of breakpoints.

This is a trivial bound. But Myers and Griffiths [7] introduced a general method (called the *composite method*) to *amplify* weaker bounds, to get much higher overall lower bounds. We apply the composite method to the minimum mosaic problem. Instead of computing a lower bound on the whole matrix, we compute lower bound B_m for each of the $\binom{m}{2}$ intervals, each with the above idea. Then we combine these bounds to form a *composite* bound as detailed in [7,6]. This improves the lower bounds, demonstrated by our empirical studies in Section 4. One of our major results in this paper is the demonstration of the effectiveness of the composite lower bound for the minimum mosaic problem.

3.2 Exact Method for the Minimum Mosaic Problem When $K_f \geq 3$

When $K_f \geq 3$, no polynomial-time algorithm is known for the minimum mosaic problem for either haplotypes or genotypes. Here we develop a method that solves the minimum mosaic problem exactly, and give heuristics that make it practical for a range of data of current biological interest. Simulation shows that our method works well for a large range of problem instances when $K_f = 3$, and for medium-size data (say 50 by 50 matrix) when $K_f = 4$. We describe our method for haplotypes, but remark that the method can be modified to handle genotype data. Practical performance of the method in [15] was not demonstrated there, but the method was implemented in program *haplovisual* (http://www.cs.helsinki.fi/u/prastas/haplovisual). Direct implementation of Ukkonen's method is expected to be prohibitive when n and m increase, even for a small number of founders. Our initial experiments with program *haplovisual* suggest it is not practical for 20 or more rows and three or more founders.

We start by developing some notation and terminology. The choice of binary states for each of the K_f founders at a site i is called the "founder setting at site i", and denoted $f(i)$. There are $2^{K_f} - 2$ possible founder settings at a site, assuming each site contains both 0's and 1's. A combined founder setting at each of the sites from 1 to i is denoted $F(i)$ and called a "founder setting up to i"; a founder setting up to m is denoted F and is called a "full founder setting". The founder setting at site i together with a legal mapping of input sequences to the K_f founders is called the "configuration at site i". A mapping is legal for site i if the state of each input sequence equals the state, at site i, of the founder it is mapped to. Clearly, given configurations at sites i and $i+1$, the number of breakpoints that occur between these two sites is the number of input sequences mapped to different founders at sites i and $i+1$, which is at most n. A combined configurations at each of the sites from 1 to i is denoted $C(i)$ and called a "configuration up to i", and the founder setting up to m is called the "full configuration".

Given a founder setting F, the problem of finding a full configuration that minimizes the number of breakpoints is called the "CF problem". Given F and i, the problem of finding a configuration $C(i)$ up to i to minimize the number of breakpoints in $F(i)$ is called the $CF(i)$ problem. Problem CF can be solved by a simple greedy algorithm [12] that is run *independently* for each input sequence s as follows. To start, set a variable p_s to 1 and find the longest match, starting

at site p_s, between s and any of the founder sequences. If the longest match extends to site i and occurs between s and founder q, then map each site from 1 to i in s to founder q. If there are ties for longest match, q can be set to be any one of the tied founders. Next, set p_s to $i+1$ and iterate. Continue until the end of s is reached.

Now suppose that a full founder setting F, and the input sequences, are only given to the greedy algorithm one site at a time, in increasing order. Then the greedy algorithm doesn't know the full length of any match between a founder sequence and an input sequence. However, the CF problem can be solved with a "locally greedy algorithm" that implicitly records all the possible actions of the greedy algorithm on each $F(i)$. Since the greedy algorithm considers each input sequence separately, we describe the locally-greedy algorithm for one input sequence s. At each site i, the locally-greedy algorithm records a subset $SF_s(i)$ of founders, and a number $BF_s(i)$. To begin, let x denote the state of sequence s at site 1. The original greedy algorithm would map s to one of the founders that has state x at site 1, so in the locally-greedy algorithm we let $SF_s(1)$ be the set of all founders which have state x at site 1, and set $BF_s(1)$ to zero. For $i > 1$, let $A_s(i)$ be the subset of founders whose state at site i agrees with the state of s at site i. Then $SF_s(i) = SF_s(i-1) \cap A_s(i)$, and $BF_s(i) = BF_s(i-1)$, if the intersection is non-empty; otherwise, $SF_s(i) = A_s(i)$, and $BF_s(i) = BF_s(i-1)+1$. $BF_s(m)$ is the number of breakpoints in the optimal solution to problem CF, given the full founder setting F. It is also easy to reconstruct the optimal configuration by a backwards trace from m to 1. Note that at each i, the SF sets compactly and implicitly encode all the optimal configurations for the $CF(i)$ problem that the greedy algorithm could find. Note also that the locally-greedy algorithm not only solves the CF problem, given F, but also solves each of the $CF(i)$ problems implied by each $F(i)$.

We now describe our method to solve the minimum mosaic problem; the method must find both an optimal F and a solution to the implied CF problem. At the high level, before optimizations to significantly speed it up, the method enumerates all possible founder settings $F(i)$, for i from 1 to m, dovetailing the execution of the locally-greedy algorithm on each growing $F(i)$. In more detail, the algorithm builds a branching tree T where the root is at level 0 and each node v at level i represents one possible founder setting at site i, denoted $f_v(i)$. The path from the root to v specifies a distinct founder setting up to i, denoted $F^v(i)$. Let w denote the predecessor of v in T; the path in T to w specifies a founder setting denoted $F^w(i-1)$. Suppose that the execution of the locally-greedy algorithm along the path to w has computed the subset of founders $SF_s^w(i-1)$ and the number of breakpoints $BF_s^w(i-1)$ (based on $F^w(i-1)$), for each input sequence s. Then, given $f_v(i)$, one step of the locally-greedy algorithm can easily compute the next set $SF_s^v(i)$ and the number $BF_s^v(i)$ for each input sequence s. Note that the algorithm at level i only needs information from level $i-1$, which allows significant space savings. The node at level m with smallest $\sum_s BF_s(m)$ identifies an optimal solution to the minimum mosaic problem. The correctness of this method follows from the correctness of the locally-greedy algorithm on any

fully specified F, and the fact that all possible $F(i)$ are enumerated. However, without further speedups the method is only practical for very small data sizes.

The obvious speedup is to implement a branch-and-bound strategy, using a lower bound $L(i+1)$ on the number of breakpoints needed for the sites $i+1$ to m. If at node v, $\sum_s BF_s^v(i) + L(i+1)$ is greater or equal to the number of breakpoints needed in some known full configuration, then no expansion from node v is needed. We have implemented this strategy using the lower bound described earlier, but we have found the following speedup to be more effective.

If $\sum_s BF_s^v(i) - \sum_s BF_s^u(i) \geq n$, then no expansion from v is needed. To see this, note that if v were expanded, any configuration at a child of v (at level $i+1$) can be created from any one of the implicitly described configurations at u, using at most n breakpoints between sites i and $i+1$. Therefore, any path to level m from v requiring b breakpoints will require at most $b+n$ breakpoints from u. This idea can be greatly sharpened as follows. Suppose for some input sequence s, the set $SF_s^v(i) \subseteq SF_s^u(i)$, and consider a configuration c at a child of v at level $i+1$. If configuration c maps s to a founder in $SF_s^v(i)$, then configuration c can be created from at least one of the implicitly described configurations at u, with no breakpoints in s between sites i and $i+1$. If c maps s to a founder not in $SF_s^v(i)$ then there is one breakpoint used (for s) on that path out of v, and so one breakpoint can also be used on a path out of u to create the same mapping of s. Continuing with this reasoning, let n' be the number of sequences s where $SF_s^v(i) \subseteq SF_s^u(i)$. Then if $\sum_s BF_s^v(i) - \sum_s BF_s^u(i) \geq n - n'$, node u is as good or better than v, and v can be pruned. To fully implement this idea, we examine pairs of nodes at level i to find any node that is "beaten" by another node, and therefore can be pruned. While that is a relatively expensive step, without any pruning the size of T grows exponentially with i, and so it is worthwhile for the algorithm to spend time finding significant pruning. We have seen empirically that this approach is very effective in efficiently solving the minimum mosaic problem for a small number of founders (in the range 3 to 5) and a number of input sequences and sites which is generally larger than many biological applications today.

There is another speedup that can be introduced if the number of founders becomes large. As described above, tree T cannot contain two founder settings up to i, $F^u(i)$ and $F^v(i)$ at nodes u and v, where the *ordered* rows of $F^u(i)$ and $F^v(i)$ are identical. However, the rows of $F^u(i)$ can be the same as the rows of $F^v(i)$, but in a permuted order. We call such a pair of nodes "isomorphic", and in any isomorphic pair only one of the two nodes needs to be expanded; the other node and the subtree extending from it can be deleted. Redundant computation caused by isomorphism only becomes a significant problem when the number of founders is large, but isomorphism can be easily handled or avoided. One simple rule to handle it is to only expand a node u if the rows of $F^u(i)$ are in lexicographic sorted order (say lexicographically non-decreasing); any node whose rows are not in lexicographic sorted order can be pruned. That leads to the idea of only generating founder settings whose rows are in lexicographic order, avoiding isomorphic pairs entirely. Suppose inductively that at level $i-1$,

every generated founder setting up to $i-1$, $F^w(i-1)$, has rows that are in lexicographic sorted order. Of course, all identical rows in $F^w(i-1)$ will be contiguous. Then for any set of k identical rows in $F^w(i-1)$, if a potential founder setting $f_v(i)$ for a child v of w, would set k' of those k rows to 0, and $k-k'$ to 1, at site i, place the zeros in the first k' of those k rows. In that way, the rows of $F^v(i)$ will be in lexicographic sorted order, and no isomorphism will be created at level i.

4 Simulation Results and Open Problems

We implemented the general method (without speedups to avoid isomorphism) in a C++ program, and ran our program on biological datasets on a standard 2.0GHz Pentium PC.

The first data is Kreitman's classic data [4]. After appropriate data reduction [12], there are 9 haplotypes and 16 sites left. The simulation results, including lower bound and exact minimum number of breakpoints, are shown in Table 1 for different K_f. As expected, as the number of founders increases, the minimum number of breakpoints decreases. Note that the *composite* lower bound using the composite method can be higher than the simple lower bound B_m on the entire data. For example, when $K_f = 3$, $B_m = 9 - 3 = 6$, while the composite bound reported in Table 1 is equal to 10.

Table 1. Solutions for minimum mosaic problem for Kreitman's data. The data is reduced from the original 11 haplotypes and 43 binary sites. After data reduction, there are 9 rows and 16 sites left. Both lower bound (LB) and exact minimum number of breakpoints (EMB) are shown. Running time (Time) is also displayed.

	$K_f = 2$	$K_f = 3$	$K_f = 4$	$K_f = 5$	$K_f = 6$
LB	27	10	7	4	3
EMB	37	15	8	6	4
Time (s)	< 1	< 1	1	12	1245

For a larger example, we use the full Jackson region of the LPL data [8] (with 40 haplotypes and 49 sites). After data reduction, it contains 37 haplotypes and 43 sites. When $K_f = 3$, it takes 27 seconds to find 241 as the minimum number of breakpoints. When $K_f = 4$, it takes a little over an hour to find 181 as the minimum number of breakpoints. The program, taking about 50 minutes, was also used to show that 53 breakpoints are the minimum needed in a dataset with 20 haplotypes and 36 sites and $K_f = 5$, posted at http://www.unc.edu/courses/2007spring/comp/790/087/. A heuristic greedy algorithm discussed there previously found a solution with 54 breakpoints.

Our program, called *RecBlock*, is available for download at the web page: http://wwwcsif.cs.ucdavis.edu/~wuyu/.

Open problems. A major open problem is to determine the complexity of the minimum mosaic problem. Another interesting problem is to develop a (possibly parametrized) polynomial time algorithm when K_f is a small constant larger than two.

Acknowledgments. Work supported by grants CCF-0515278 and IIS-0513910 from National Science Foundation. We thank Leonard McMillan at UNC for providing the dataset discussed above. We also thank Yun S. Song for bringing the linkage mapping problem to our attention.

References

1. El-Mabrouk, N.: Deriving haplotypes through recombination and gene conversion, J. of Bioinformatics and Computational Biology 2, 241–256 (2004)
2. El-Mabrouk, N., Labuda, D.: Haplotype histories as pathways of recombinations, Bioinformatics 20, 1836–1841 (2004)
3. Kimmel, G., Shamir, R.: A block-free hidden markov model for genotypes and its application to disease association, J. of Comp. Bio. 12, 1243–1260 (2005)
4. Kreitman, M.: Nucleotide polymorphism at the alcohol dehydrogenase locus of drosophila melanogaster, Nature 304, 412–417 (1983)
5. McMillan, L., Moore, K.: `http://www.unc.edu/courses/2007spring/comp/790/087/?p=11`
6. Myers, S.: The detection of recombination events using DNA sequence data, PhD dissertation, Dept. of Statistics, University of Oxford, Oxford, England (2003)
7. Myers, S.R., Griffiths, R.C.: Bounds on the minimum number of recombination events in a sample history. Genetics 163, 375–394 (2003)
8. Nickerson, D., Taylor, S., Weiss, K., Clark, A., et al.: DNA Sequence Diversity in a 9.7-kb region of the human lipoprotein lipase gene. Nature Genetics 19, 233–240 (1998)
9. Rastas, P., Koivisto, M., Mannila, H., Ukkonen, E.: A Hidden Markov Technique for Haplotype Reconstruction. In: Proceedings of Workshop on Algorithm of Bioinformatics (WABI 2005) pp. 140–151 (2005)
10. Scheet, P., Stephens, M.: A fast and flexible statistical model for large-scale population genotype data: applications to inferring missing genotypes and haplotypic phase. Am. J. Human Genetics 78, 629–644 (2006)
11. Schwartz, R., Clark, A., Istrail, S.: Methods for Inferring Block-Wise Ancestral History from Haploid Sequences. In: Proceedings of Workshop on Algorithm of Bioinformatics (WABI'02), vol. 2452, pp. 44–59 (2002)
12. Song, Y.S., Wu, Y., Gusfield, D.: Efficient computation of close lower and upper bounds on the minimum number of needed recombinations in the evolution of biological sequences. Bioinformatics, vol. 421, pp. i413–i422. In: Proceedings of ISMB (2005)
13. Tan, Y., Fu, Y.: A Novel Method for Estimating Linkage Maps, Genetics 173, 2383–2390 (2006)
14. Tyson, G.W., Chapman, J., Hugenholtz, P., Allen, E., Ram, R., Richardson, P., Solovyev, V., Rubin, E., Rokhsar, D., Banfield, J.: Community structure and metabolism through reconstruction of microbial genomes from the environment, Nature 428, 37–43 (2004)
15. Ukkonen, E.: Finding Founder Sequences from a Set of Recombinants. In: Proceedings of Workshop on Algorithm of Bioinformatics (WABI'02), vol. 2452, pp. 277–286 (2002)