Incremental Haplotype Inference, Phylogeny, and Almost Bipartite Graphs*

Peter Damaschke
Department of Computer Science and Engineering
Chalmers University, 41296 Göteborg, Sweden
ptr@chalmers.se

Abstract

We address the combinatorial problem of inferring haplotypes in a population that forms a perfect phylogeny (PP) given a sample of genotypes. The problem is relevant because, in DNA sequencing, genotypes are easier to obtain than haplotyping by DNA sequencing. Since PP's appear naturally and frequently on DNA sequences of restricted length, PP haplotyping is a favourable approach to facilitate reliable haplotype inference. Since Gusfield's seminal paper from 2002, a number of different algorithms have been proposed. Here we give an algorithm that identifies haplotypes incrementally (along the sequence). Under the random mating assumption, all sufficiently frequent haplotypes are inferred from a random genotype sample of asymptotically optimal size. By its extreme simplicity, the idea of the algorithm easily extends to more general population structures. This can be beneficial because the strict PP assumption is easily violated in reality. Missing data can also be recovered by incremental haplotyping, if they are not too prevalent. In a more graph-theoretic part of this work we solve a problem we call almost-2-coloring of graphs, which arises in an enhanced version of our haplotyping algorithm. We show that the solution space of this graph problem can be computed in linear time.

1 Introduction

Somatic cells of diploid organisms such as higher animals and plants contain two copies of genetic material, in pairs of homologous chromosomes. The genetic

*An early version has been presented at the *2nd RECOMB Satellite Workshop on Computational Methods for SNPs and Haplotypes*, Pittsburgh 2004.

data on a fixed part of a single chromosome is called a *haplotype*. Formally we may describe a haplotype as a vector $(a_1, \ldots, a_m)$ where m is the number of sites considered, and a_i is the genetic data at site i. Here the term *site* (or synonymously, *locus*) can refer to a gene, a short subsequence, or even a single nucleotid. The a_i are called *alleles*. The vector of *unordered* pairs $(\{a_1, b_1\}, \ldots, \{a_m, b_m\})$ resulting from haplotypes $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$ on homologous chromosomes is called a *genotype*.

Usual sequencing methods yield only genotypes but not the pairs of haplotypes they are built from, the so-called phase information. Haplotyping techniques exist, but they are more expensive. On the other hand, haplotype data is needed for analyzing the background of hereditary dispositions and prediction of traits: A hereditary trait often originates from mutations on a chromosome that has been transmitted over generations, with further silent mutations (without effect) supervened. This way the trait is associated with a certain subset of haplotypes, even if the observed loci do not contain the trait gene. Hence one can observe correlations between the presence of certain haplotypes and the trait. Especially when a trait appears or disappears several times in the evolutionary tree, it is not associated with a single mutation at the observed loci but with combinations of them. If only genotypes are known, it is more difficult to see correlations. Once the frequent haplotypes in a population are determined, it is a straightforward procedure to find the haplotype pair that constitutes a given genotype. Recent methods for association analysis [1, 2, 3] use haplotypes. Other applications include questions from population dynamics.

Generally speaking, haplotypes are much more informative than genotypes, since they change only slowly over generations, whereas genotypes are just the results of random pairings of haplotypes. Therefore it is important to reconstruct haplotypes from observed genotypes. A genotype with $i > 0$ ambigous sites can be explained by 2^{i-1} distinct haplotype pairs, and reconstruction is impossible as long as isolated genotypes are considered. However, with a large enough genotype sample and a proper assumption about the structure of the population it is sampled from, one can often infer the haplotypes with high accuracy.

One such assumption is that every new mutation affects a new site never changed before, and pointwise mutations are the only way haplotypes are modified, which is very likely to be true for nearby loci on DNA from relatively young populations. In this case, at most two alleles appear at every site. The two possible values of every allele are usually denoted by 0 and 1 (where the naming is arbitrary). Moreover, and more substantially, the haplotypes are in an obvious sense vertices of a tree:

Definition 1 *A* perfect phylogeny (PP) *is a tree whose vertices are labeled by bit vectors of length m such that every bit changes at most once in the tree, along one edge. In other words, every bit a_i either has the same value in the* *whole tree, or there is exactly one edge e_i such that $a_i = 0$ on one side of e_i, and $a_i = 1$ on the other side.*

The PP haplotyping problem is: Given a set of n vectors of m pairs $\{0, 0\}$, $\{1, 1\}$, $\{0, 1\}$, explain each of them by two haplotypes, so that all these $2n$ (not necessarily distinct) haplotypes fit in a PP. The first published result by Gusfield [4] was an almost linear-time algorithm, however this algorithm relies on deep results from matroid theory and is by far too complicated for implementation. Alternative algorithms [5, 6] are slower but elementary and practical. Independently, we had proposed in [7] an algorithm that differed from the others in both the approach and the goal. Whereas the former algorithms are purely combinatorial and generate a description of all possible haplotyping results for any set of genotypes, we concentrated on the haplotypes that can be inferred safely (perhaps the most important information). We obtained a better expected running time under the natural assumption of random mating, and we estimated the sample size being sufficient for inferring all haplotypes above some desired frequency threshold. The problem has also been considered under probabilistic assumptions in [8]. An almost linear-time algorithm has been given there, based on a result of Pe'er et al. [9]. It can also cope with missing information, and it works under the *rich data hypothesis* which demands, roughly speaking, that enough combinations of alleles be present in the data matrix, which is true with high probability in a large enough sample under the random mating assumption. All algorithms mentioned above are not very simple.

The method presented here achieves the goals of our earlier work in a more natural way and has several other advantages: The basic algorithm is extremely simple. The very idea is to solve the problem incrementally on increasing subsequences of k sites ($k = 1, 2, 3, \ldots$) and to extend the haplotyping results, so-called k-haplotypes. This extension step works with a few local rules in a suitably defined sample graph. The rules can be applied to the known data even if some small fraction of data items is missing. The algorithm itself is deterministic and runs in time linear in the size of the data. Given a random sample of genotypes, it identifies with high probability all haplotypes that appear with a frequency above some user-defined threshold. We analyze its resolution performance under the random mating assumption. A very mild extra assumption is that no k-haplotype makes up for the majority of the population, therefore we start at some site where the number of 0's and 1's is as balanced as possible. The necessary size of the random sample is close to a provable lower bound. Actually, the algorithm makes use of an assumption that is properly weaker than PP, hence it works accurately for more general populations violating the very strict PP assumption. The algorithm in its basic form does not even explicitly use any tree structure of the population. Next, one can easily add data in both directions, i.e., new genotypes and new sites of already considered genotypes, because the very idea is to add new sites and to extend the haplotyping results incrementally. The basic scheme can be smoothly enhanced if the random sample is not large enough to make the local rules work. In the extreme case, the local rules may be replaced with some global computation. (We will characterize and completely solve the arising graph coloring problem.)

More recently, the PP haplotyping problem for complete data has been solved by a practical and deterministic linear-time algorithm [10]. (We also refer to that paper for pointers to other recent developments.) Still, the extreme simplicity, extendability, and tolerance of missing data seems to render our algorithm an interesting alternative. Coping with missing data is important for practicality [11].

Organization of the paper: In Section 2 we introduce the overall idea. Section 3 outlines a "parameterized" probabilistic analysis of the algorithm on a random sample of genotypes under the random mating assumption. In the more informal Section 4 we discuss how our scheme can cope practically with incomplete data. In Section 5 we turn to pure combinatorics and develop a linear-time algorithm for the aforementioned graph coloring problem that may be used instead of the simple local rules if the necessity arises. Section 6 is devoted to extensions to not so perfect phylogenies. Section 7 reports an implementation, and Section 8 concludes the paper.

2 The Basic Algorithm

The following well-known characterization of PP has been discovered several times (cf. [4, 5]):

Lemma 1 *A population of haplotypes has a PP if and only if, for no pair of sites, all four combinations 00, 01, 10, 11 of alleles appear in the population.* □

Let H be a population of haplotypes of length m that build a PP. For any integer k, $0 \leq k \leq m$, denote by H_k the set of prefixes of length k of the haplotypes in H. Note that, since H has a PP, every H_k and every subset of H_k forms a PP as well. This follows trivially from Lemma 1. For $k = 0$, by definition H_k contains only the empty string.

Suppose that H is unknown to us, but we are given a (random) sample S of genotypes of length m, each formed by two haplotypes from H. In the following we develop an algorithm that takes this sample and outputs haplotypes from H. We will show that this algorithm is *correct* in the sense that all returned haplotypes really exist in H. Since the input is a random sample S of genotypes, in general not all haplotypes in H will be found and not all genotypes in S will be resolved, but in Section 3 we show that all haplotypes above some frequency threshold are found with high probability, given a large enough S. (Note that *no* haplotyping algorithm can guarantee to reconstruct the whole of H from a random sample S, already for the trivial reason that some rare haplotypes might not even appear in S.)

A *k-haplotype* is any binary sequence of length k. Our algorithm will, for every $k = 0, \ldots, m$, return a set of k-haplotypes and keep a certain subset $S_k \subseteq S$ for the remaining steps.

More specifically, we will establish the following inductive hypothesis: *All returned k-haplotypes exist in H_k, and all genotypes $g \in S_k$ are resolved until site k.* The latter expression means that the two k-haplotypes in H_k that build the prefix of length k of g are correctly reconstructed. From the inductive hypothesis it follows, with $k = m$, that all returned haplotypes exist in H, as claimed.

We begin specifying our algorithm. For $k = 0$ we output only the empty string, and let $S_0 = S$. This makes the inductive hypothesis vacuously true for $k = 0$, as H_0 contains the empty string.

For the main loop of the algorithm, suppose that we have a set of k-haplotypes that exist in H_k, and a set $S_k \subseteq S$ of genotypes resolved until site k (inductive hypothesis). In the following, letters x, y, z, etc. denote k-haplotypes, and a letter with Boolean value 0 or 1 attached denotes a $(k+1)$-haplotype in the obvious sense. Now the algorithm takes this known subset of H_k and constructs a sample graph:

Definition 2 *We define the* sample graph *G_k as follows. The vertices of G_k are the known k-haplotypes. For every genotype in S_k, the two vertices (k-haplotypes) contained therein are joined by an edge. There is one edge for every genotype (hence loops and multiple edges can occur in G_k). Every edge is labeled by an unordered pair, namely one of 00, 11, and 01, corresponding to the Boolean values at site $k+1$ of the genotype.*

By the inductive hypothesis, every vertex of G_k represents some k-haplotype that exists in H_k. Now consider any edge uv in G_k (possibly with $u = v$). If uv is a 00-edge then, obviously, the $(k+1)$-haplotypes $u0$ and $v0$ must exist in H_{k+1}. Similarly, if uv is a 11-edge, then the $(k+1)$-haplotypes $u1$ and $v1$ must exist in H_{k+1}. Finally, if the edge uv is a loop ($u = v$) labeled with 01, then clearly the $(k+1)$-haplotypes $u0$ and $v1$ must exist in H_{k+1}. (These trivial implications hold for arbitrary populations, not only for PP.) The only problem is with 01-edges uv that are not loops. In this case we know that either $u0, v1$ or $u1, v0$ must exist in H_{k+1}, but from the edge alone we cannot decide which option is true.

Here the PP assumption comes into play. The following is an immediate consequence of Lemma 1:

Lemma 2 *For at most one vertex v of G_k, both $v0$ and $v1$ can exist in H_{k+1}.*

Proof. Suppose that $u0, u1, v0, v1$ are in H_{k+1}, for two distinct vertices u and v of G_k. Since, by the inductive hypothesis, u and v represent different k-haplotypes in H_k, there must be a site $i \leq k$ where u has 1 and v has 0, or vice versa. But now all pairs 00, 01, 10, 11 appear at sites $i, k+1$, so that H_{k+1} is not a PP. This contradicts the assumption that the whole population is PP. □

We refer to a vertex v such that both $v0, v1 \in H_{k+1}$ as a *split vertex*.

Now we formulate two rules for inferring certain $(k+1)$-haplotypes in H_{k+1}. Let x, y, z be k-haplotypes that are vertices in G_k (and hence belong to H_k, by the inductive hypothesis).

Rule 1: If x is incident to some 00-edge or 01-loop in G_k, then $x0 \in H_{k+1}$. Similarly, If x is incident to some 11-edge or 01-loop in G_k, then $x1 \in H_{k+1}$.
Rule 2: Suppose that x is joined to distinct vertices y and z by 01-edges. If $y0, z0 \in H_{k+1}$ then also $x1 \in H_{k+1}$. Similarly, if $y1, z1 \in H_{k+1}$ then also $x0 \in H_{k+1}$.

Lemma 3 *Rule 1 and 2 are sound.*

Proof. As we have already seen above, Rule 1 is obviously correct, even without the PP assumption. Correctness of Rule 2 follows from Lemma 2 (we show only the first part, as the second part is symmetric): Since $y0, z0 \in H_{k+1}$, at most one of $y1, z1$ can be in H_{k+1} as well. Without loss of generality, assume $y1 \notin H_{k+1}$. But since xy is a 01-edge, it must be the case that $x1 \in H_{k+1}$. □

In each extension step (going from H_k to H_{k+1}) we first apply Rule 1 wherever possible, and then we apply Rule 2 to vertices x that have two neighbors y, z such that either $y0, z0 \in H_{k+1}$ or $y1, z1 \in H_{k+1}$ has been already inferred by Rule 1.

Now we have determined a set of $(k+1)$-haplotypes which, by Lemma 3, surely belong to H_{k+1}. Finally we define $S_{k+1} \subseteq S_k$ and resolve the genotypes in S_{k+1} until site $k+1$. In the language of G_k that means, we explicitly assign the two Boolean labels of edge uv to its vertices u and v, for edges uv that we want to keep in S_{k+1}. (The problem is that not necessarily all edges of G_k have been involved when we applied the rules.)

For 00-edges and 11-edges this step is trivial, and we keep all the corresponding genotypes in S_{k+1}. Consider any 01-edge uv where $u0, v1$ are returned and thus $u0, v1 \in H_{k+1}$ (or the symmetric case with 0 and 1 switched). Then it is clear that the genotype until site $k+1$ consists of $u0, v1$. The other option $u1, v0$ is impossible because then we also have $u1, v0 \in H_{k+1}$, which contradicts Lemma 2. Thus, we keep such genotypes in S_{k+1} as well. In all other cases we simply throw out the edge uv and do not include the corresponding genotype in S_{k+1}.

We conclude with a high-level description of the algorithm. Note that the algorithm is computationally very simple. Using basic data structures for manipulating graphs, it runs in $O(nm)$ time.

Algorithm "Incremental Haplotyping"
Background: a PP population H of unknown haplotypes of length m.
Input: a sample S of n genotypes, each formed by two haplotypes from H.
Output: a subset of H.

Summary of the Algorithm:
1. Arrange the sites in any linear order, but start with a site where the maximum frequency of 0s and 1s is minimal. Trivially, the empty string is known to be a 0-haplotype in H_0. Let $S_0 := S$. Do the following steps for $k = 0, 1, 2, \ldots, m-1$.
2. Construct graph G_k from S_k and the known k-haplotypes in H_k. 3. Apply Rule 1 to G_k. The inferred $(k+1)$-haplotypes are in H_{k+1}.
4. Apply Rule 2 to G_k and the already known haplotypes in H_{k+1}. The inferred $(k+1)$-haplotypes are in H_{k+1}.
5. Put all genotypes in S_{k+1} which are represented by 00-edges and 11-edges where both vertices represent inferred $(k+1)$-haplotypes.
6. Put all genotypes in S_{k+1} which are represented by 01-edges uv where either $u0, v1$ or $u1, v0$ are inferred $(k+1)$-haplotypes.

From Lemma 3 and the subsequent discussion it follows:

Theorem 1 *Provided that the given population is PP, Incremental Haplotyping outputs only correct haplotypes that exist in the population.*

On the other hand, the analysis in Section 3 we will show that the simple rules of Incremental Haplotyping are sufficient to infer all *frequent* k-haplotypes (above some frequency threshold) with high probability, from a random genotype sample S whose size is close to a trivial lower bound. To explain a detail in the initial step: The analysis needs that on the starting site there is no vast majority of 0s or 1s. This is a very mild and easy-to-check additional assumption. In real populations it should even hold for many sites, so that we are not in trouble getting a starting point.

Another remark is that we applied Rule 2 only to those $(k+1)$-haplotypes inferred before by Rule 1. Of course, one could choose to iterate Rule 2 as long as it is applicable, and possibly infer even more $(k+1)$-haplotypes. However, our analysis will not take advantage of this cascading effect, therefore we have described the simplest variant.

Example: As an illustration we give an example with six genotypes (first column) built from the population of haplotypes 00000, 10000, 01000, 10100, 10010, 01001; it is not hard to see that they form a PP. This sample is large enough to reconstruct the haplotypes: The other columns are the six edges, i.e., pairs of vertices, of the sample graph in each step. In Step 3, genotypes b and e are 01-edges adjacent to vertex 100. Both 101 and 000 are already extended to 1010 and 0000 elsewhere, hence Rule 2 implies that 100 must be extended to 1001 in b and e. Note that 1000 appears as well (in a and d), thus 100 is a split vertex. In step 4, c is a 01-loop causing the split vertex 0100, and f is resolved afterwards by Rule 2, as edges c and f that meet in 0100 are now incident to two different vertices (0100 and 0000) already extended by 0 elsewhere.

a	20000	0	00	000	0000	00000
		1	01	100	1000	10000
b	10220	1	10	100	1001	10010
		1	10	101	1010	10100
c	01002	0	01	010	0100	01000
		0	01	010	0100	01001
d	10200	1	10	100	1000	10000
		1	10	101	1010	10100
e	20020	0	00	000	0000	00000
		1	10	100	1001	10010
f	02002	0	00	000	0000	00000
		0	01	010	0100	01001

3 Analysis of Incremental Haplotyping

To study the power of Incremental Haplotyping we adopt the *random mating assumption:* Every haplotype appears with some fixed frequency in the population, and for any ordered pair of haplotypes (x, y), the probability to pick a genotype composed of x and y is simply the product of their frequencies (and hence twice as much for the unordered pair $\{x, y\}$, if $x \neq y$). It is equivalent to say that haplotypes are sampled according to their frequencies in the population, independently and with repetition, and paired up to genotypes. Although the random mating assumption is barely perfectly satisfied in real populations (e.g. due to mating preferences), it provides a reasonable theoretical model for analyzing the performance one can expect of a haplotyping method.

Very rare haplotypes will probably not show up in a random sample. On the other hand, they are also of minor interest. Therefore the following definition is sensible:

Definition 3 *We fix some parameter h and call a haplotype* frequent (rare) *if it appears with frequency at least (less than) $1/h$. Furthermore, let r be the sum of frequencies of rare haplotypes.*

We aim at discovering all frequent haplotypes. Recall that n denotes the size of the random genotype sample S. The goal of this section is to estimate a sample size n (depending on h and r) that is *sufficient* to infer, with some prescribed probability $1 - \epsilon$, all frequent haplotypes.

The following technical trick will simplify our analysis. We assume that some imaginary, omniscient "helper" that knows the haplotype frequencies preprocesses S and gives us a modified sample $S' \subseteq S$ instead of S. We analyze how our algorithm would behave on S' (which will be simpler), and then we drop the assumption and conclude the real behaviour on S. Speficically, set S' provided by the helper is the set of genotypes from S composed of two frequent haplotypes. Since a randomly picked haplotype is some of the rare haplotypes with probability r, the random mating assumption yields $|S'| = (1-r)^2|S|$ in expectation.

Equivalently we can say that S' has been sampled from the given population without the rare haplotypes. Note that the frequency of every haplotype in this "purged" population is at least $1/h(1-r)$, by the definition of h and r. Based on these relationships we proceed as follows. First we analyze the special case that all haplotypes in the population are frequent. (That is, we assume $r = 0$ and further denote the minimum frequency by $1/h$). In the resulting expression for sample size n, we finally replace h with $h(1-r)$, and multiply the expression with $1/(1-r)^2$. This yields the result for arbitrary $r > 0$. (Note that rare haplotypes that are present in the real sample S but ignored by the analysis can only increase the rate of recognized frequent haplotypes.)

We suppose that, from some small k on, each k-haplotype appears with frequency at most $1/2$, that is, no single k-haplotype has the majority. In the special case that a majority haplotype x exists, it will be quickly detected by Rule 1 (in simpler words: at least $n/4$ genotypes in the sample are just twice x) and can be removed from S, similarly as rare haplotypes.

Now we begin with the actual analysis. The *degree* of a vertex in a graph is the number of edges incident with this vertex, where a loop counts twice. By the definition of G_k, the degree of any vertex in G_k is the number of occurences of the corresponding k-haplotype in S_k. Without loss of generality, let 0 be the more frequent allele at every site (otherwise we switch the names 0 and 1 at the sites where the majority was 1). By Rule 1 and 2, any $v0$ is inferred if v is incident to a 00-edge, and any $u1$ is inferred if u is incident to two 01-edges ending in distinct 0-vertices that are already inferred.

Define $Z = 2n/h$. Consider any fixed haplotype $\hat{x}$. Trivially, the prefix x of length k of $\hat{x}$ has a frequency at least $1/h$, since $\hat{x}$ has. From random mating it follows that the expected degree of the corresponding vertex x in G_k is at least Z. For the moment consider vertices whose actual degree is at least Z.

In the following, a *0-vertex* in G_k is a vertex representing a k-haplotype that is prefix of a $(k+1)$-haplotype in H_{k+1} that has a 0 at site $k+1$. A *1-vertex* is defined similarly. Note that a split vertex vertex (see Lemma 2) is both a 0-vertex and a 1-vertex.

Suppose that $\hat{x}$ has a 0 at site $k+1$, that is, x is a 0-vertex. Since x has degree at least Z, the probability that no edge incident with x ends in another 0-vertex is bounded by $(1/2)^Z$. That is, we infer the $(k+1)$-haplotype $x0$ with probability $1-(1/2)^Z$.

Suppose (the other case) that $\hat{x}$ has a 1 at site $k+1$, that is, x is a 1-vertex. Then we infer $x1$ already if some edge incident with x ends at another 1-vertex, or two edges incident with x end at different 0-vertices. The only bad case is that *all* edges incident with x end at a unique 0-vertex. Let b denote the highest frequency of a k-haplotype in the population. Then the bad case happens with probability at most b^Z. That is, we infer $x1$ with probability $1-b^Z$.

Since $b < 1/2$ (see above), the probability to miss any specific $(k+1)$-haplotype is at most $(1/2)^Z$ in either case. Moreover, we do not lose edges between vertices, once we have inferred the $(k+1)$-haplotypes there (see steps 5-6 of the algorithm).

Since we have considered m sites, each with at most h different k-haplotypes, the union bound yields: The probability that some of the k-haplotypes ($k = 1,\ldots,m$) is not inferred is at most $hm/2^Z$, provided that all vertices in the G_k have minimum degree Z. Recall that Z was only a lower bound for the *expected* degree of any vertex. However, for arbitrarily small $\delta > 0$, the probability of degrees smaller than some $(1-\delta)Z$ quickly goes to 0 for growing n (due to Chernoff bounds). It could simply be added to the failure probability, but since this term becomes arbitrarily small and does not affect the asymptotic result below, we skip this technicality.

Let ϵ be the user-defined failure probability. Now $n = hZ/2$ and $Z = \log_2(hm/\epsilon)$ gives the sample size $n = \frac{1}{2}hZ = \frac{h}{2}\log_2(hm/\epsilon)$ for the case $r = 0$. As we discussed above, for general $r > 0$ we have to replace h with $h(1-r)$, and to multiply the result eventually by $1/(1-r)^2$. Altogether we conclude:

Theorem 2 *For a parameter $h > 1$, let r be the total frequency of rare haplotypes in a given population, i.e., those haplotypes with frequencies below $1/h$. Suppose that, from some fixed k on, no k-haplotype has the majority, and genotypes are formed by random mating. Then Incremental Haplotyping recognizes with probability at least $1-\epsilon$ all haplotypes of frequency at least $1/h$, using a sample of*

$$n = \frac{h}{2(1-r)}\log_2((1-r)hm/\epsilon)$$

genotypes, subject to a factor that tends to 1 as $1/\epsilon$ grows. □

For fixed r, ϵ and $h = \Omega(m)$ this sample size is $O(h\log h)$. Remarkably, this bound cannot be improved, because $\Theta(h\log h)$ random genotypes are already needed to hit all haplotypes of frequency at least $1/h$, by the well-known “coupon collector” result from basic probability theory. By adding more complicated local rules besides Rule 1 and 2, one could only reduce the constants hidden in $O(h\log h)$, but not the asymptotics.

4 Missing Data

In real data, a fraction of data items in the genotype sample may be missing or unreadable, that is, we read symbol "?" instead of 0,1 or 2. For the kth step in Incremental Haplotyping this means that we may not know the labels of some edges in G_k; let us call them ?-edges. But Rule 1 and 2 are still sound and can be applied to the available 00-, 01- and 11-edges, and we can also resolve the corresponding genotypes until site $k+1$ exactly as before.

Moreover, we can reconstruct the missing labels of certain ?-edges. Remember that Lemma 2 guarantees the existence of at most one split vertex. If the split vertex has been recognized, that is, the $(k+1)$-haplotypes $v0$ and $v1$ have been inferred for some v, then we also know the label of any ?-edge not incident to v, from the uniquely determined alleles at site $k+1$ of their endvertices. However, ?-edges incident to the split vertex v remain unresolvable and are excluded from S_{k+1}.

The more cumbersome case is that no split vertex has been recognized. Note that there may still exist a hidden split vertex v, but we may have failed to infer $v0$ or $v1$. In order to avoid the loss of all ?-edges, we restrict the split vertex candidates as follows. The PP tree of the known k-haplotypes can easily be reconstructed incrementally [12], and this can be done synchronously with the haplotyping. Once the tree is known, the only split vertex candidates are the two vertices of the tree edge where the $(k+1)$st bit changes. Thus, we exclude only those ?-edges from S_{k+1} which are incident to these two candidate vertices.

Still, discarding unresolvable ?-edges can lead to substantial loss in the sample, if the (candidate) split vertices always represent haplotypes with high frequencies and the rate of missing data items is high. But this loss seems to be unavoidable in the worst case. (We remark that an algorithm of Kimmel and Shamir [11] needs $n = \omega(m^2)$ genotypes in the random mating model, in order to guarantee a complete solution with high probability in $\tilde{O}(m^2 n)$ expected time, if missing entries appear independently and with constant probability.) However, we observed in simulations that typically a considerable fraction of genotypes is kept in the sample.

5 All Colorings of Almost Bipartite Graphs

The step from k-haplotypes to $(k+1)$-haplotypes in Incremental Haplotyping gives rise to a certain graph coloring problem. Remember that our inferred k-haplotypes are represented as vertices of a graph G_k. Now let us interpret the alleles at site $k+1$ as two *colors* that we still denote 0 and 1. Every vertex has one color, except a possible split vertex having both colors. We attempt to infer the vertex colors from the edge labels. We call a vertex *precolored* if it is incident to a 00- or 11-edge and *must* therefore receive the corresponding color, see Rule 1. Then the following nontrivial problem remains, due to Lemma 2.

Almost 2-Coloring:
Given a graph with uncolored and precolored vertices (colored either 0 or 1 or both 0 and 1) and 01-edges, assign colors 0 and 1 to the vertices of every 01-edge, in such a way that at most one split vertex exists in the whole graph.

The precise connection to Incremental Haplotyping is as follows. Due to Lemma 2, we can infer a $(k+1)$-haplotype $v0$ if vertex v gets color 0 in *all* solutions to Almost 2-Coloring, and similarly with color 1. What we have shown is that vertices representing frequent haplotypes fulfill this condition with high probability in a large enough sample and, moreover, the simple Rules 1 and 2 are then sufficient to actually determine the color. However, if only a smaller sample is available, many vertex colors may still be identical in all almost 2-colorings, but Rules 1 and 2 become too weak. In this situation we may compute all solutions to Almost 2-Coloring in the graph and finally determine the unambiguously colored vertices. The question is how difficult this graph problem is.

This is a possible enhancement of Incremental Haplotyping, but no longer simple. However, we show in this section that all solutions to Almost 2-Coloring can still be computed and represented in linear time. This result might also find independent interest, as recognition of graphs that "almost" have certain properties is an established field of research, also in connection with other computational biology problems. More specifically, Almost 2-Coloring is related to bipartization problems [13, 14].

We first solve the special case without precolored vertices. After that, it is not hard take also precolored vertices into account. Let us call a graph G *almost bipartite* if it is not bipartite but becomes bipartite (2-colorable) after deletion of some vertex and all its incident edges. Almost bipartite graphs can be recognized in linear time as shown by Cai and Schieber [15]. Later we have somewhat simplified the algorithm of Cai and Schieber in [16]. Note that a graph is bipartite if and only if it has no odd cycles. We identify a cycle with its vertex set. The results from the aforementioned papers relevant for our purpose are summarized in:

Theorem 3 *Given a graph G, one can recognize that G is bipartite, and otherwise compute the intersection X of all odd cycles in G and a particular odd cycle C, everything in linear time. The vertices in X are exactly those which can take the role of the split vertex. X is nonempty if and only if G is almost bipartite.*

For the other prerequisites used below, we refer to standard graph-algorithmic results as provided in many textbooks.

If G is bipartite then every 2-coloring (computed independently in the connected components of G) is a solution to Almost 2-Coloring. In addition to these solutions, any vertex may be chosen as a split vertex. If the split vertex is an articulation point, we get further colorings in an obvious way. Since articulation points and the block-tree structure can be computed in linear time, we also get a concise description of all possible 2-colorings with at most one split vertex in linear time.

Now we treat the more tricky case: If G is almost bipartite, exactly one connected component G' is almost bipartite, and all other components must be bipartite. Clearly, the split vertex must be in G', thus it suffices to study this component from now on.

Compute X and some C in G', according to Theorem 3. Note that $\emptyset \neq X \subseteq C$. Removal of any vertex $x \in X$ induces two possible colorings of the path $C - x$, which differ only by swapping the colors. The vertices of X partition C in an obvious sense in edge-disjoint paths which we call the *segments* of C, with the understanding that every vertex of X belongs to both incident segments. Now let D be any connected component of $G' - C$. The *contacts* of D are those vertices of C adjacent to some vertex of D. We say that a pair of vertices has *parity* 0 (1) if both have the same color (different colors). With these denotations we formulate and prove:

Lemma 4 *All contacts of a component D of $G' - C$ are in one of the segments of C, possibly including one or both ends of this segment in X. In particular, D may have exactly one contact $x \in X$, in which case x is an articulation point.*

Proof. We claim that two vertices $x_1, x_2 \in X$ and two contacts y_1, y_2 of D can never occur in cyclic ordering (x_1, y_1, x_2, y_2) on C.

To prove the claim, note that y_1 and y_2 are connected by an even path and an odd path on C. If we delete once x_1 and once x_2 then y_1, y_2 have the same color in one rest graph and different colors in the other rest graph. But since y_1, y_2 are also connected by some path through D, only one case can be true, a contradiction.

The claim immediately implies that all contacts of D in $C - X$ must belong to one segment. Moreover, D cannot have four contacts in X.

Assume D has three contacts in X. If we delete any of them, say x, then the parity of the two others is determined by their fixed distance in D, but at the same time by their distance on path $C - x$. It is easy to check that these three parities contradict each other, since C is an odd cycle. Hence D can have at most two contacts in X.

The claim also imposes restrictions on the positions of contacts of D in X and $C - X$: Two contacts in X must be ends of the same segment, and the other contacts must be within this segment (or in the same segment if $|X|$=2). Similarly, if D has exactly one contact x in X then the other contacts are necessarily in one of the segments incident to x. From these observations the lemma follows. $\Box$

The next lemma describing all 2-colorings of $G' - x$ for all $x \in X$ in terms of parities is almost an immediate consequence. Despite its lengthy formulation, it provides a comprehensive overview of the solution space.

Lemma 5 *Given a connected, almost bipartite graph G' with a set X of k vertices $x_0, \dots, x_{k-1}$, appearing in this cyclic ordering on some odd cycle C, the vertex set of $G' - X$ can be partitioned into disjoint sets P_i, $i = 0, \dots, k-1$, and Q_{ij}, $j = 1, \dots, j(i)$ so that the statements below hold for this partition. Moreover, the partition and the cyclic ordering, which we call the* cyclic structure *of G', is computable in linear time. The P_i, Q_{ij}, and $\{x_i\}$ are called* parts*, and arithmetic is understood modulo k.*
(1) The parity of any pair of vertices inside one part is the same in all $G' - x_i$.
(2) The parity of any pair of vertices, one from P_a and one from P_b, is the same in all $G' - x_i$, $i = a+1, \dots, b$, and the same in all $G' - x_i$, $i = b+1, \dots, a$, and different in both cases.
(3) Property (2) holds similarly for any pair of vertices, one from $\{x_a\} \cup \bigcup_j Q_{aj}$ and one from P_b, excluding case $i = a$.
(4) Property (2) holds similarly for any pair of vertices, one from $\{x_a\} \cup \bigcup_j Q_{aj}$ and one from $\{x_b\} \cup \bigcup_j Q_{bj}$, excluding cases $i = a$ and $i = b$.
(5) In $G' - x_i$, all the Q_{ij} and the rest of $G' - x_i$ can be 2-colored independently.

Proof. Every connected component of $G' - C$ with x_i as the only contact becomes one of the Q_{ij}, where j is just a serial number. We define P_i to be the union of all components D of $G' - C$ having their contacts on the path on C from x_i to x_{i+1} in the cyclic ordering (except the components Q_{ij} and $Q_{i,j+1}$), plus the vertices of this path. Properties (1)-(5) follow easily from the connectivity relationships and the fact that C is odd. The complexity follows from Theorem 1 and the linear-time computability of connected components. $\square$

Finally, Lemma 5 also makes it easy to describe how precolored vertices in the almost bipartite graph G' further restrict the possible 2-colorings: Any precolored vertex v determines the coloring of each $G' - x_i - \bigcup_j Q_{ij}$ completely, with the exception $G' - x_i$ if $v = x_i$ or $v \in Q_{ij}$. Moreover, the colorings of $G' - x$ and $G' - x'$ for $x, x' \in X$ differ exactly on the parts assigned to that path between x and x' on C which is disjoint to the contact set of the part containing v. These observations remain true if, more generally, all precolored vertices belong to the same part.

Something interesting happens if two parts, say P_a and P_b, contain precolored vertices: Since fixed colors also imply fixed parities, by Lemma 5 the elements of either $\{x_{a+1}, \dots, x_b\}$ or $\{x_{b+1}, \dots, x_a\}$ are no longer split vertices. That is, we can shrink the cyclic structure by merging all parts which have contacts in one of the two paths on C between x_a and x_b. Similarly we can proceed if some parts P_{aj} and Q_b, or Q_{aj} and $Q_{bj'}$ have precolored vertices. If some Q_{aj} and $Q_{aj'}$ have precolored vertices then either a is the only possible split vertex or the two parts can be merged into one. If several parts have precolored vertices, we can successively shrink the cyclic structure as above. Details are straightforward.

Corollary 1 *All 2-colorings of a connected, almost bipartite graph are described by a cyclic structure as in Lemma 5, where at most one part has precolored vertices.*

Altogether we obtain:

Theorem 4 *A linear-space description of all solutions of an instance of* ALMOST 2-COLORING *(the cyclic structure) can be computed in linear time.*

The observation that the cyclic structure shrinks so easily in the presence of precolored vertices indicates that random almost-bipartite graphs (appropriately defined) have a unique coloring already for relatively few edges. A probabilistic analysis might be interesting. The cyclic structure is also useful for describing all possible results of some consecutive non-unique steps in incremental PP haplotyping.

We conclude the section with a summarizing high-level description of the algorithm, with standard routines and straightforward details omitted. Note that the (possibly exponentially many) solutions can in fact be completely described in linear space, due to the independent swapping of colors in different parts of the graph. By deciding the colors of these parts we can also output explicit solutions with linear-time delay.

Input: a graph G, where some vertices may be precolored with 0 or 1.
Output: a concise description of all 2-colorings (with 0 and 1) that satisfy the following: precolored vertices keep their colors; the vertices of every edge get colors 0 *and* 1; and at most one vertex (split vertex) gets both colors.

Summary of the Algorithm:
1. Compute the connected components of G.
2. Compute the articulation points and two-connected components of G.
3. Test whether G is bipartite. In this case, 2-color the connected components. If all possible 2-colorings (with colors 0 and 1 switched independently in the components) are in conflict with two precolored vertices, then no solution exists. Otherwise, keep the solutions with at most one split vertex.
4. In the solutions without split vertex, do the following. For each articulation point v (separately), declare v a split vertex and switch the colors independently in the components of $G - v$ that have no precolored vertices. For each other vertex v (separately), declare v a split vertex and just add the other color to v. STOP.
5. (G is not bipartite). Determine the non-bipartite connected components of G. If there is more than one, then no solution exists. Compute the solutions in the bipartite components, without split vertex, as above. Let G' be the unique

non-biparite component.
6. In G' compute the intersection X of all odd cycles, and distinguish some odd cycle C.
7. Fix an orientation of C. Denote the vertices of X by $x_0, \ldots, x_{k-1}$, in the order they appear on C. For all i, let C_i be the path of C from x_i to x_{i+1} (addition is understood modulo k).
8. For all i, let the Q_{ij} be the connected components of $G' - C$ with the only contact x_i, where $j = 1, 2, 3, \ldots$
9. For all i, let P_i be the union of C_i and all connected components of $G' - C$ that have all their contacts (neighbors in C) in C_i, except the components Q_{ij} and $Q_{i+1,j}$ from Step 8.
10. 2-color every Q_{ij} and P_i. (Every part without precolored vertices has two 2-colorings obtained by swapping the colors.)
11. Now the space of all solutions (2-colorings of the parts that go together) is described by the parity conditions (1)-(5) in Lemma 2. STOP.

6 Beyond Perfect Phylogeny

Incremental Haplotyping only relies on a certain property of PP's, namely that at most one k-haplotype is extended by both 0 and 1 (Lemma 2). Let us call this property the *1-split property.* Hence the algorithm already reconstructs the frequent haplotypes correctly if they come from a population enjoying this weaker property. The catch is that the 1-split property highly depends on the order the sites are considered. Actually, we have the following equivalence, which is yet another formulation of Lemma 1:

Proposition 1 *A set of binary strings has a PP if and only if the 1-split property holds for every permutation of the sites.*

However, even if the population is not PP, a large fraction of permutations may still have the 1-split property. In this case we would succeed by running the Incremental Haplotyping algorithm on a random order of sites. If it goes through, we know that the result is correct. If the algorithm detects more than one split vertex in some step, we can try another random permutation, etc., for a certain number of trials. This readily extends the applicability of our method to not so perfect phylogenies.

Example: Consider the rows of the matrix

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

One can verify that only those permutations having the first two sites at place 1 and 2 violate the 1-split property. This is only a fraction of $2!4!/6! = 1/15$ of all permutations.

In an evolutionary history, random mutations may easily affect a few sites more than once, i.e., some bits may switch forth and back between 0 and 1.[17] Incremental Haplotyping may also be applied to such populations, by allowing a limited number of split vertices (rather than only one) in every step. Resolution rules and analysis would work similarly as in Section 2-3, however we have to leave this subject for further research.

7 Implementation

The basic Incremental Haplotyping algorithm (Section 2), also for missing entries (Section 4) has been implemented by Michael Ewald and later enhanced by William Garner. A Java application is available at the author's homepage. The user must provide input as a file of equally long words with characters 0,1,2 (standing for $\{0,0\}$, $\{1,1\}$, $\{0,1\}$), any other symbol is treated as a missing item.

Missing entries are recovered if possible, as described in Section 4: ?-edges incident to the split vertex are discarded. All ?-edges are discarded in steps where no split vertex can be identified.

In order to increase the rate of recognized haplotypes for relatively small sample sizes n and for the case of missing entries, several measures have been added to the basic algorithm during the course of that work: The program can run the algorithm on several orders of sites and combine the resulting partial haplotypes. Also, Rules 1 and 2 have been enriched by a few more local rules.

Combining results from several orders of sites works as follows. Consider two partially resolved versions $(a'_1, \ldots, a'_m)$ and $(a''_1, \ldots, a''_m)$ of the same haplotype $(a_1, \ldots, a_m)$, that is, every a'_i and a''_i is either 0 or 1 or "?". If at least one of a'_i, a''_i has a definite value 0 or 1, then this is the true a_i. However, an obvious problem arises: Let x, y be a pair of haplotypes forming a genotype, and x', y' and x'', y'' two partially resolved outputs. Then we must be able to recognize that x', x'' and y', y'', respectively, are parts of the same haplotype (rather than x', y'' and x'', y'). A sufficient condition for correct assignment is that at least one 2 at the same position has been resolved in both x', x'' and y', y'', since then 0 and 1 will "identify" the two haplotypes.

In particular, one option is the straight-and-reverse mode: It runs the algorithm twice, in original and reverse order, which resolves already most of the positions where one run of the algorithm got stuck. An explanation is that Rule 2 fails mostly for k-haplotypes with low frequency, which is not so typical in early steps. The user can also run the algorithm from all start positions k, that is, apply it to the cyclic shifts $k \ldots, n, 1, \ldots, k-1$. Since the basic algorithm works in linear time, such multiple runs are still fast.

We summarize some simulation results from Ewald's master thesis. One suite of experiments was done on sequences of fixed length $m = 50$ from a randomly generated PP population. The straight-and-reverse mode worked significantly better than a single run, and then even cyclic runs with all 50 start positions did not add much to the performance. The rate of resolved genotypes in this mode was about 75% for $n = 50$, 90% for $n = 75$, and reached nearly 100% for $n > 150$. Simulations with randomly deleted entries have been done, e.g., with length $m = 30$ and sample size $n = 150$. For 1% missing data, resolution was still complete, and every further percent of missing entries reduced the rate of resolved *genotypes* by roughly 10%. However, the rate of recognized *haplotypes* weighted by their frequencies is in general higher, because the more frequent haplotype appear in several genotypes. The performance in the case of missing entries was improved later in Garner's implementation by a few additional rules and repeated runs on randomly permuted sites.

8 Conclusions

We proposed a conceptually very simple algorithm for PP haplotyping, working on increasing prefixes of the given genotypes (in any desired order), that has provably good resolution power under mild probabilistic assumptions about the population. For incomplete genotype data the resolution rate of the basic algorithm drops with growing error rate because of a simple policy of discarding genotypes that are not immediately resolved. But we can overcome this drawback by multiple runs on different orders of sites. It seems to be a great advantage of the algorithm that it can be smoothly enhanced in various ways, in particular to nearly perfect phylogeny cases. It would be nice to elaborate more on these extensions.

References

[1] Onkamo P et al., Association analysis for quantitative traits by data mining: QHPM, *Annals of Human Genetics* **66**:419–429, 2002.

[2] Toivonen HT, Onkamo P, Vasko K, Ollikainen V, Sevon P, Mannila H, Herr M, Kere J, Data mining applied to linkage disequilibrium mapping, *American Journal of Human Genetics* **67**:133–145, 2000.

[3] Zhang S, Zhang K, Li J, Zhao H, On a family-based haplotype pattern mining method for linkage disequilibrium, *Pacific Symposium on Biocomputing*, pp. 100–111, 2002.

[4] Gusfield D, Haplotyping a perfect phylogeny: Conceptual framework and efficient solutions, *6th Int. Conference on Research in Computational Molecular Biology RECOMB*, pp. 166–175, 2002.

[5] Bafna V, Gusfield D, Lancia G, Yooseph S, Haplotyping as perfect phylogeny: A direct approach, *Journal of Computational Biology* **10**:323–340, 2003.

[6] Eskin E, Halperin E, Karp RM, Efficient reconstruction of haplotype structure via perfect phylogeny, *Journal of Bioinformatics and Computational Biology* **1**:1–20, 2003.

[7] Damaschke P, Fast perfect phylogeny haplotype inference, *14th Symposium on Fundamentals of Computation Theory FCT*, *Lecture Notes in Computer Science 2751*, pp. 183–194, 2003.

[8] Halperin E, Karp RM, Perfect phylogeny and haplotype inference, *8th Int. Conference on Research in Computational Molecular Biology RECOMB*, pp. 10–19, 2004.

[9] Pe'er I, Pupko T, Shamir R, Sharan R, Incomplete directed perfect phylogeny, *SIAM Journal on Computing* **33**:590–607, 2004.

[10] Ding Z, Filkov D, Gusfield D, A linear-time algorithm for the perfect phylogeny haplotyping (PPH) problem, *Journal of Computational Biology* **13**:522–553, 2006.

[11] Kimmel G, Shamir R, The incomplete perfect phylogeny haplotype problem, *Journal of Bioinformatics and Computational Biology* **3**:1–25, 2005.

[12] Waterman MS, *Introduction to Computational Biology*, Chapman and Hall, 1995.

[13] Guo J, Gramm J, Hüffner F, Niedermeier R, Wernicke S, Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization, *Journal of Computer and System Sciences* **72**:1386–1396, 2006.

[14] Raman V, Saurabh S, Sikdar S, Improved exact exponential algorithms for vertex bipartization and other problems, *9th Italian Conference on Theoretical Computer Science ICTCS*, *Lecture Notes in Computer Science 3701*, pp.375–389, 2005.

[15] Cai L, Schieber B, A linear-time algorithm for computing the intersection of all odd cycles in a graph, *Discrete Applied Mathematics* **73**:27–34, 1997.

[16] Damaschke P, Linear-time recognition of bipartite graphs plus two edges, *Discrete Mathematics* **262**:99–112, 2003.

[17] Song YS, Wu Y, Gusfield D, Algorithms for imperfect phylogeny haplotyping with a single homoplasy or recombination event, *5th International Workshop on Algorithms in Bioinformatics WABI*, *Lecture Notes in Computer Science 3692*, pp. 152–164, 2005.