In Computational Methods for SNPs and Haplotype Inference, S. Istrail, M. W aterman, and A. Clark (eds).
Lecture Notes in Computer Science, vol. 2983, Springer, p. 9-25, 2004.

An Overview of Combinatorial Methods for Haplotype Inference

Dan Gusfield[1]
Department of Computer Science, University of California, Davis
Davis, CA. 95616

Abstract

A current high-priority phase of human genomics involves the development of a full *Haplotype Map* of the human genome [23]. It will be used in large-scale screens of populations to associate specific haplotypes with specific complex genetic-influenced diseases. A key, perhaps bottleneck, problem is to computationally infer haplotype pairs from genotype data. This paper follows the talk given at the DIMACS Conference on SNPs and Haplotypes held in November of 2002. It reviews several combinatorial approaches to the haplotype inference problem that we have investigated over the last several years. In addition, it updates some of the work presented earlier, and discusses the current state of our work.

1 Introduction to SNP's, Genotypes and Haplotypes

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*. In complex diseases (those affected by more than a single gene) it is often much more informative to have haplotype data (identifying a set of gene alleles inherited together) than to have only genotype data.

The underlying data that forms a haplotype is either the full DNA sequence in the region, or more commonly the values of *single nucleotide polymorphisms (SNP's)* in that region. A SNP is a single nucleotide site where exactly two (of four) different nucleotides occur in a large percentage of the population. The SNP-based approach is the dominant one, and high density SNP maps have been constructed across the human genome with a density of about one SNP per thousand nucleotides.

[1]Research partially supported by grants DBI-9723346 and EIA-0220154 from the National Science Foundation. email: gusfield@cs.ucdavis.edu

1.1 The biological problem

In general, it is not feasible to examine the two copies of a chromosome separately, and *genotype* data rather than haplotype data will be obtained, even though it is the haplotype data that will be of greatest use.

Data from m sites (SNP's) in n individuals is collected, where each site can have one of two states (alleles), which we denote by 0 and 1. For each individual, we would ideally like to describe the states of the m sites on each of the two chromosome copies separately, i.e., the haplotype. However, experimentally determining the haplotype pair is technically difficult or expensive. Instead, the screen will learn the $2m$ states (the genotype) possessed by the individual, without learning the two desired haplotypes for that individual. One then uses computation to extract haplotype information from the given genotype information. Several methods have been explored and some are intensively used for this task [6, 7, 12, 33, 17, 31, 28, 29]. None of these methods are presently fully satisfactory, although many give impressively accurate results.

1.2 The computational problem

Abstractly, input to the haplotyping problem consists of n *genotype* vectors, each of length m, where each value in the vector is either 0,1, or 2. Each position in a vector is associated with a site of interest on the chromosome. The position in the genotype vector has a value of 0 or 1 if the associated chromosome site has that state on both copies (it is a *homozygous* site), and has a value of 2 otherwise (the chromosome site is *heterozygous*).

Given an input set of n genotype vectors, a *solution* to the *Haplotype Inference (HI) Problem* is a set of n pairs of binary vectors, one pair for each genotype vector. For any genotype vector g, the associated binary vectors v_1, v_2 must both have value 0 (or 1) at any position where g has value 0 (or 1); but for any position where g has value 2, exactly one of v_1, v_2 must have value 0, while the other has value 1. That is, v_1, v_2 must be a feasible "explanation" for the true (but unknown) haplotype pair that gave rise to the observed genotype g. Hence, for an individual with h heterozygous sites there are 2^{h-1} haplotype pairs that could appear in a solution to the HI problem.

For example, if the observed genotype g is 0212, then the pair of vectors 0110, 0011 is one feasible explanation, out of two feasible explanations. Of course, we want to find the explanation that actually gave rise to g, and a solution for the HI problem for the genotype data of all the n individuals.

However, without additional biological insight, one cannot know which of the exponential number of solutions is the "correct one".

1.3 The need for a genetic model

Algorithm-based haplotype inference would be impossible without the implicit or explicit use of some genetic model, either to asses the biological fidelity of any proposed solution, or to guide the algorithm in constructing a solution. Most of the models use statistical or probabilistic aspects of population genetics. We will take a more deterministic or combinatorial approach.

In this paper we will review several combinatorial investigations into the haplotype inference problem, and in each case, discuss the implicit or explicit genetic model that is involved.

2 Optimizing Clark's method

A. Clark, in [6] was the first to propose an algorithm to solve the haplotype inference problem. It has been widely used, and is still in use today. We will explain the method is a somewhat more abstract setting.

Abstractly, input consists of n vectors, each of length m, where each value in the vector is either 0,1, or 2. Each position in a vector is associated with a site of interest on the chromosome. The state of any site on the chromosome is either 0 and 1. The associated position in the vector has a value of 0 or 1 if the chromosome site has that state on both copies (it is a homozygous site), and it has a value of 2 if both states are present (it is a hetrozygous site). A position is considered "resolved" if it contains 0 or 1, and "ambiguous" if it contains a 2. A vector with no ambiguous positions is called "resolved", and otherwise called "ambiguous". haplotype pair. Given two non-identical resolved vectors R and NR, the *conflation* of R and NR produces the ambiguous genotype vector A, with entry 0 (respectively 1) at each site where both R and NR have 0 (respectively 1) entries, and with entry 2 at each site where the entries of R and NR differ.

The method of Clark begins by identifying any vector in the input with zero or one ambiguous sites, since in the first case the original two haplotypes are identical to the genotype vector, and in the second case the one ambiguous site has a forced resolution, producing two forced haplotype vectors. These identified haplotypes are called the *initial resolved* vectors (haplotypes). Clark's method requires that some initial resolved haplotypes are available in this way.

The main part of Clark's method resolves remaining ambiguous genotypes by expanding out from the initial resolved haplotypes. Clark [6] proposed the following rule that infers a new resolved vector NR (or haplotype) from an ambiguous vector A and an already resolved vector R. The resolved vector R can either be one of the input resolved vectors, or a resolved vector inferred by an earlier application of the Inference Rule.

> **Inference Rule:** Suppose A is an ambiguous vector with h, say, ambiguous positions, and R is a known resolved vector which equals one of the 2^h potential resolutions of vector A (where each of the ambiguous positions in A is set to either 0 or 1). Then infer that A is the conflation of one copy of resolved vector R and another (uniquely determined) resolved vector NR. All the resolved positions of A are set the same in NR, and all of the ambiguous positions in A are set in NR to the *opposite* of the entry in R. Once inferred, vector NR is added to the set of known resolved vectors, and vector A is removed from the vector set.

For example, if A is 0212 and R is 0110, then NR is 0011. The interpretation is that if the two haplotypes in a screened individual are 0110 and 0011, then the observed genotype would be 0212. The inference rule resolves the vector 0212 using the belief that 0110 is a haplotype in the population, to infer that 0011 is also a haplotype in the population.

When the Inference Rule can be used to infer the vector NR from the vectors A and R, we say that R can be *applied* to (resolve) A. It is easy to determine if a resolved vector R can be applied to resolve an ambiguous vector A: R can be applied to A if and only if A contains no resolved position s such that the values of A and R differ at position s. A resolved position s in A whose value does differ from that in R is said to *block* the application of R to A. For example, 0110 can not be applied to 2012 because position two (from the left) blocks the application.

Clark's entire algorithm for resolving the set of genotypes is to first identify the initial resolved set, and then repeatedly apply the Inference Rule until either all the genotypes have been resolved, or no further genotypes can be resolved.

The implicit genetic model (I believe) behind the Inference Rule is that the genotypes of individuals in the current population resulted from random mating of the parents of the current population. The sampled individuals are also drawn randomly from the population, and the sample is small compared

to the size of the whole population, so the initial resolved vectors likely represent common haplotypes that that appear with high frequency in the population. Hence these haplotypes are likely to have been used in the creation of currently unresolved genotypes. So, if an unresolved genotype A can be explained the by conflation of two initial resolved haplotypes, or by using one of the initial resolved haplotypes, it is sensible to resolve A in that way, and then to deduce that vector NR is also in the population. We can define the "distance" of an inferred haplotype NR from the initial resolved vectors, as the number of inferences used on the shortest path of inferences from some initial resolved vector, to vector NR. The above explanation for the correctness of an Inference Rule becomes becomes weaker as it is used to infer vectors with increasing distance from the initial resolved vectors. However, Clark's Inference Rule is "globally" justified in [6] by an additional empirical observation that will be discussed shortly.

Note that in the application of the Inference Rule, there may be choices for vectors A and R, and since A is removed once it is resolved, a choice that is made at one point can constrain future choices. Hence, one series of choices might resolve all the ambiguous vectors in one way, while another execution, making different choices, might resolve the vectors in a different way, or leave orphans, ambiguous vectors that cannot be resolved. For example, consider a problem instance consisting of two resolved vectors 0000 and 1000, and two ambiguous vectors 2200 and 1122. Vector 2200 can be resolved by applying 0000, creating the new resolved vector 1100 which can then be applied to resolve 1122. That execution resolves both of the ambiguous vectors and ends with the resolved vector set 0000, 1000, 1100 and 1111. But 2200 can also be resolved by applying 1000, creating 0100. At that point, none of the three resolved vectors, 0000, 1000 or 0100 can be applied to resolve the orphan vector 1122.

The problem of choices is addressed in [6] by using an implementation of the method where the choices are affected by the ordering of the data. For any input, the data is reordered several times, the method is rerun for each ordering, and the "best" solution over those executions is reported. Of course, only a tiny fraction of all the possible data orderings can be tried. We will refer to this as the *local inference method*.

Without additional biological insight, one cannot know which execution (or data ordering) gives the solution that is nearest to the truth. However, simulations discussed in [6] show that the inference method tends to produce the wrong vectors only when the execution also ends with ambiguous vectors that cannot be resolved. That is, there is some "global" structure to the set of true haplotypes that underly the observed genotypes, so that if

some early choices in the method incorrectly resolve some of the genotypes, then the method will later become stuck, unable to resolve the remaining genotypes. The exact nature of this global structure was not made explicit, but simulations reported in [6] confirmed this expectation. Executions of the method that resolved all the ambiguous genotypes, more accurately found the original haplotypes than did executions that got stuck. Clark, therefore recommended that his method should be run numerous times, randomly reordering the input data each time, and then the execution that resolved the most genotypes should be the one most trusted.

2.1 The MR problem

Given what was observed and proposed in [6], the major open algorithmic question from [6] is whether *efficient* rules exist to break choices in the execution of Clark's algorithm, so as to maximize the number of genotypes it resolves. This leads to the problem studied in [16, 17]

> **Maximum Resolution (MR)**: Given a set of vectors (some ambiguous and some resolved), what is the maximum number of ambiguous vectors that can be resolved by successive application of Clark's Inference Rule?

Stated differently, given a set of vectors, what execution of the inference method maximizes the number of ambiguous vectors that are resolved? We want to answer this question, rather than rely on re-running the method many times, sampling only a miniscule fraction of the possible executions. An algorithm to solve the MR problem needs to take a more *global* view of the data, than does the more local inference method, to see how each possible application of the Inference Rule influences choices later on.

Unfortunately, in [17], we show that the MR problem is NP-hard, and in fact, Max-SNP complete. The reduction also shows that two variants of the MR problem are NP-hard. We next reformulated the MR problem as a problem on directed graphs, with an exponential time (worst case) reduction. We will detail this approach below. That graph-theoretic problem can be solved via integer linear-programming. Experiments with this approach suggest that the reduction is very efficient in practice, and that linear programming alone (without explicit reference to integrality) often suffices to solve the maximum resolution problem.

2.2 A graph-theoretic view of the MR problem

Given the NP-hardness and Max-SNP completeness of the MR problem, we would like a "heuristic" method that feels likely to perform well in practice. That algorithm might not always find the optimal solution to the MR problem, but it should correctly know when it has actually found the optimal and when it has not. To do this, we need an algorithm that takes a more global view of the data before deciding on where to apply the Inference Rule. One approach is to translate the MR problem (via a worst-case exponential-time reduction) to a graph problem as follows.

We create a directed graph G containing a set of nodes $N(A)$, for each ambiguous vector A in the input, and a set of nodes, I, containing one node for each resolved vector in the input. In detail, for each ambiguous vector A, with say h ambiguous positions, $R(A)$ is the set of the 2^h distinct, resolved vectors created by setting the ambiguous positions in A (to zero or one) in all possible ways. $N(A)$ is a set of 2^h nodes, each labeled by a distinct vector in $R(A)$. Note that two nodes (in different $N()$ sets) can have the same label, but the nodes are distinguished by the ambiguous vector they originate from. Then connect a directed edge from any node v to any node v' in G if and only v' is in a set $R(A)$ for some ambiguous vector A (i.e., v' is not in I), and the application of resolved vector labeling v to the ambiguous vector A would create the resolved vector labeling v'.

The application of a resolved vector to an ambiguous vector (if possible) uniquely determines the inferred resolved vector. Hence, for any ambiguous vector A and any node v in G, there is at most one edge from v to the set of nodes $N(A)$. Therefore, any directed tree in G rooted a node $v \in I$ specifies a feasible history of successive applications of the Inference Rule, starting from node $v \in I$. The non-root nodes reached in this tree specify the resolved vectors that would be created from ambiguous vectors by this succession of applications. Therefore, the MR problem can be recast as the following problem on G.

> **The Maximum Resolution on G (MRG) Problem** Find the largest number of nodes in G that can be reached by a set of node-disjoint, directed trees, where each tree is rooted at a node in I, and where for every ambiguous vector A, *at most* one node in $N(A)$ is reached.

Despite the exponential worst-case blowup, this formulation of the MR problem is appealing because the construction of G in practice is likely to be efficient, and because without the last constraint, the MRG problem is

trivial. The construction is efficient because it can be implemented to avoid enumerating isolated nodes of G, and the expected number of ambiguous positions in any vector is generally small. Let $\overline{G}$ denote graph derived from G where every node that can't be reached from I has been removed, along with any incident edges. The implementation given in [17] constructs $\overline{G}$ in time proportional to mn^2 plus the number of edges in $\overline{G}$, which in practice is a very small fraction of the number of edges in G.

2.3 An Exact Integer Programming formulation for the MRG problem

The MRG problem can first be formulated by adding simple non-linear constraints to a network flow formulation. First, let all the edges in $\overline{G}$ be given a huge capacity (larger than the number of genotype in the input). Then, add a source node and sink node to $\overline{G}$; direct an edge of infinite capacity from the source node to each node in I; direct an edge with capacity one from each node not in I to the sink node. Clearly, a feasible solution to the MRG problem that reaches q nodes (and hence resolves q ambiguous vectors) defines a source-sink flow of value q exactly. However, the converse does not yet hold, since we have not excluded the possibility of reaching more than one node in any set $N(A)$. For that, consider a linear programming formulation (for background see [27, 32]) of the above network flow problem, and let x_e denote the variable for the flow on edge e. Then for every pair of edges e, e' that enter nodes in some set $N(A)$, add in the *non-linear* constraint $x_e x_{e'} = 0$ to the network flow formulation. An integer solution to this mathematical program exactly solves the MRG problem. But since this formulation involves non-linear constraints, it is not clear how we would solve it in practice.

In [17], we explored a different integer programming formulation, that is not exact, but that found the exact solution most of the time. However, recently R. Ravi [**?**] suggested how to reformulate the non-linear constraints as linear, integer constraints, which results in making the above formulation an exact integer, linear-programming formulation of the MRG problem.

Let c_e and $c_{e'}$ denote the capacities on edges e and e' respectively. Ravi suggested replacing the constraint $x_e x_{e'} = 0$, with $x_e \leq d_e$, $x_{e'} \leq c_e d_{e'}$ and $d_e + c_{e'} d_{e'} \leq 1$, where d_e and $d_{e'}$ are 0,1 variables. Since c_e and $c_{e'}$ are constants, these three inequalities are linear, and their effect is to limit the flow to at most one of the edges e or e'.

2.4 Results

The maximum-resolution hypothesis in [6] is that the most accurate solutions tend to come from the executions of the inference method that resolve the most ambiguous vectors. In the simulations done in [17] we verify the basic maximum-resolution hypothesis. We consistently observed that the executions with the most *correct* resolutions were the ones with the most resolutions. More informatively, the ratio of correct resolutions to total resolutions increases as the total number of resolutions increases, and the distribution of the number of correct resolutions tends to be more skewed to the right, as the number of resolutions increases. These simulations are consistent with the basic maximum-resolution hypothesis. However, maximum-resolution alone is not a sufficient guide to finding the largest number of correct resolutions, because the distribution of the number of correct resolutions have high variance, even among those executions that resolve the maximum number of vectors.

So, in order to maximize the number of correct resolutions, it is indeed best to select from those executions that maximize the total number of resolutions (as the maximum-resolution hypothesis states), but in order to decide which of those execution(s) to use, one still needs some additional criteria. The most effective *secondary* criteria to use, in order to find solutions with a large number of correct resolutions, is to minimize the number of *distinct* haplotypes used in the solution. That is, if we first restrict attention to those executions that maximize the number of resolutions, and then within that group of executions, restrict attention to the ones that use the smallest number of distinct haplotypes, the quality of the solution greatly improves. This will be further discussed in the next section.

3 Supercharging Clark's method

The observations reported above suggest that maximum resolution is not enough in order to find the most accurate solution when using Clark's method. Certainly, Clark's method should be run many times, as Clark suggested, but it is not clear how to use the full set of results obtained from these executions. (Interestingly, most published evaluations of Clark's method only run it once, ignoring the intuition that Clark had, that the stochastic behavior of his algorithm was an important factor.)

In [30], we examine several variants of Clark's method, using a set of 80 genotypes, of which 47 were ambiguous; the others were either homozygous at each site or only had a single hetrozygous site. Each genotype contained

nine SNP sites in the human APOE gene. Independently, the correct underlying haplotypes were laboratory-determined in order to calibrate the accuracy of each variation of Clarks's method (accuracy measured by how many of the haplotype pairs reported by the algorithm were actually the original haplotype pairs for that genotype). We observed that most variations of Clark's method produce a large number of different solutions, each of which resolved all of the 47 ambiguous genotypes. The solutions had a high accuracy variance, and choosing a solution at random from among the 10,000 solutions would give a poor solution with high probability. Hence, an important issue in using Clark's method is how to make sense, or exploit, the many different solutions that it can produce, and that each resolve all of the ambiguous genotypes.

The main result is that the following strategy works to greatly improve the accuracy of Clark's method. First, for the input genotype data, run Clark's method many times (we used 10,000 times), each time randomizing decisions that the method makes. Second, over all the runs, select those runs which produce a solution using the fewest or close to the fewest number of distinct haplotypes. The number of such runs was typically in the tens. In those tens of runs, for each genotype g in the input, record the most used haplotype pair that was produced to explain g. The set of such explaining haplotype pairs is called the "consensus solution". We observed that the consensus solution had dramatically higher accuracy than the average accuracy of the 10,000 solutions. For example, in the APOE data, in one of the variants, out of the 10,000 executions, there were 24 executions that used 20 or 21 distinct haplotypes, and twenty was the smallest observed number. The average accuracy of the 10,000 executions was 29 correct haplotype pairs out of the 47 ambiguous genotypes, and the execution with the highest accuracy in the 10,000 had 39 correct pairs. However, the average of the 24 selected executions was 36. The consensus solution of those 24 executions had 39 correct pairs. Hence, this simple rule allowed us to home in on a single solution that was as good as the most accurate solution over all the 10,000 solutions. In another variant of Clark's method, the consensus solution had 42 correct pairs, while the average of all the 10,000 solutions had 19 correct. For comparison, the program PHASE always got 42 correct solutions, and the program Haplotyper produced a range of solutions with most getting either 43 or 42 correct, with one solution getting 44 correct, and three solutions getting only 37 correct.

We also observed that among the tens of solutions that use few haplotypes, any haplotype pair that is used with high frequency, say above 85% of the time, was almost always correct. This allows one to home in on those

pairs that can be used with high confidence.

4 Perfect Phylogeny

As mentioned earlier, the haplotype inference problem would be impossible without some implicit or explicit genetic model guiding the method or selecting the most promising solutions. The most powerful such genetic model is the population-genetic concept of a *coalescent*, i.e., a rooted tree that describes the evolutionary history of a set of haplotypes in sampled individuals [34, 25]. The key observation is that "In the absence of recombination, each sequence has a single ancestor in the previous generation." [25].

That is, if we follow backwards in time the history of a single haplotype H from a given individual I, when there is no recombination, that haplotype H is a copy of one of the haplotypes in one of the parents of individual I. It doesn't matter that I had two parents, or that each parent had two haplotypes. The backwards history of a single haplotype in a single individual is a simple path, if there is no recombination. That means that the history of a set of $2n$ individuals, if we look at one haplotype per individual, forms a tree. The histories of two sampled haplotypes (looking backwards in time) from two individuals merge at the most recent common ancestor of those two individuals. (The reason for using $2n$ instead of n will be clarified shortly.)

There is one additional element of the basic coalescent model: the *infinite-sites* assumption. That is, the m sites in the sequence (SNP sites in our case) are so sparse relative to the mutation rate, that in the time frame of interest at most one mutation (change of state) will have occurred at any site.

Hence the coalescent model of haplotype evolution says that without recombination, the true evolutionary history of $2n$ haplotypes, one from each of $2n$ individuals, can be displayed as a tree with $2n$ leaves, and where each of the m sites labels exactly one edge of the tree, i.e., at a point in history where a mutation occurred at that site. This is the underlying genetic model that we assume from here on. Note that we may not know the ancestral haplotype at the root of the tree, although the algorithms can be somewhat simplified when we do know it. See [34] for another explanation of the relationship between sequence evolution and coalescents.

In more computer science terminology, the no-recombination and infinite-sites model says that the $2n$ haplotype (binary) sequences can be explained by a *perfect phylogeny* [14, 15] which is defined next.

Definition Let M be an $2n$ by m 0-1 (binary) matrix. Let V be an

m-length binary vector, called the *ancestor vector*.

A *perfect phylogeny for M and V* is a rooted tree T with exactly $2n$ leaves that obeys the following properties:

1) Each of the $2n$ rows labels exactly one leaf of T, and each leaf is labeled by one row.

2) Each of the m columns labels *exactly one* edge of T.

3) Every interior edge (one not touching a leaf) of T is labeled by *at least* one column.

4) For any row i, the value $M(i, j)$ is unequal to $V(j)$ if and only if j labels an edge on the unique path from the root to the leaf labeled i. Hence, that path is a compact representation of row i.

The biological interpretation is that an edge label j indicates the point in time where a mutation at site j occurred, and so the state of site j changes from its ancestral value to the opposite value. The motivation for the perfect phylogeny model is based on recent observations of little or no recombination in long segments of Human DNA, and the standard infinite-sites assumption.

In the *rooted* version of perfect phylogeny, V is given as input. There is also an *unrooted* version of perfect phylogeny, where V is not specified. In that case, a binary matrix M is said to have a perfect phylogeny if *there exists* a V such that there is a (rooted) perfect phylogeny for M and V.

Formally, the **Perfect Phylogeny Haplotype (PPH) Problem** is: Given a set of genotypes, M, find a set of explaining haplotypes, M', which defines an unrooted perfect phylogeny.

What happened to the genotype data?

How does the perfect phylogeny view of haplotypes relate to the problem of deducing the haplotypes when only the n genotype vectors are given as input?

The answer is that each genotype vector (from a single individual in a sample of n individuals) was obtained from the mating of two of $2n$ haplotype vectors in an (unknown) coalescent (or perfect phylogeny). That is, the coalescent with $2n$ leaves is the history of haplotypes in the *parents* of the n individuals whose genotypes have been collected. Those $2n$ haplotypes are partitioned into pairs, each of which gives rise to one of the n observed genotypes.

So, given a set S of n genotype vectors, we want to find a perfect phylogeny T, and a pairing of the $2n$ leaves of T which explains S. In addition to efficiently finding one solution to the PPH problem, we would like to deter-

mine if that is the unique solution, and if not, we want to efficiently represent the set of all solutions, so that each one can be generated efficiently.

4.1 Algorithm and program GPPH

The PPH problem was introduced and first solved in [18]. The algorithm given in [18] is based on reducing the PPH problem to a well-studied problem in graph theory, called the graph realization problem. The theoretical running time of this approach is $O(nm\alpha(nm))$, where α is the inverse Ackerman function, usually taken to be a constant in practice. Hence, the worst case time for the method is nearly linear in the size of the input, nm. The reduction in [18] has a small error, and a corrected, simplified reduction is detailed at:

wwwcsif.cs.ucdavis.edu/~gusfield/recomberrata.pdf.

The time for the reduction is $O(nm)$, and the graph realization problem can be solved by several published methods. The method in [3] is based on a general algorithm due to Lofgren, and runs in $O(nm\alpha(nm))$ time. That algorithm is the basis for the worst-case time bound established in [18], but we found it to be too complex to implement. In [18] it was explained that after one PPH solution is obtained, by whatever method, one can get an implicit representation of the set of all PPH solutions in $O(m)$ time.

Using a different solution to the graph realization problem [13], this approach yields a time bound of $O(nm^2)$, and that approach has been implemented in a program now called GPPH [5]. GPPH contains within it a fully general solution to the graph realization problem, even for instances that do not arise from any PPH problem instance.

4.2 Algorithm and Program DPPH

The second method to solve the PPH problem is called the DPPH method. The method in DPPH [1, 2] is not based (explicitly or implicitly) on a graph realization algorithm, but is based on deeper insights into the combinatorial structure of the PPH problem and its solution. The running time for the method is also $O(nm^2)$, and the algorithm produces a graph that represents the set of all solutions in a simple way. That approach has been implemented and is called DPPH.

4.3 Algorithm and program HPPH

A third method to solve the PPH problem was developed in [11]. It also has worst-case running time of $O(nm^2)$, and it can be used to find and represent

all the solutions. This approach has been implemented and is called HPPH. Although developed independently of GPPH, one can view the method in HPPH as a specialization of graph realization method used in GPPH to the PPH problem, simplifying the general purpose graph realization method to problem instances coming from the PPH problem.

4.4 Comparing the execution of the programs

All three of the programs GPPH, DPPH and HPPH are available at
http://wwwcsif.cs.ucdavis.edu/~gusfield/.

The three programs have been extensively tested on the same datasets. Before the methods were tested against one another, we expected that DPPH would be the fastest, and that HPPH would be faster than GPPH. The reason is the DPPH exploits the deepest insights into the special combinatorial structure of the PPH problem, GPPH is the most general, and HPPH can be viewed as a simplification of GPPH obtained by exploiting some structural properties of the PPH problem.

The empirical testing of the programs exactly matched our expectations, and details can be found in [4]. The empirical testing also uncovered two interesting phenomena that we discuss next.

Uniqueness of the solution: a Strong phase transition

For any given input of genotypes, it is possible that there will be more than one PPH solution. When designing a population screen and interpreting the results, a unique PPH solution is very important. So the question arises: for a given number of sites, how many individuals should be in the sample (screen) so that the solution is very likely to be unique? This is a question that was raised in [18]. Therefore, we did several experiments which determine the frequency of a unique PPH solution when the number of sites and genotypes changes. Intuitively, as the ratio of genotypes to sites increases, the probability of uniqueness increases. We studied precisely this issue, and the striking observation is that there is a strong phase transition in the frequency of unique solutions as the number of individuals grows. That is, the frequency of unique solutions is close to zero for a given number of individuals, and then jumps to over 90% with the addition of just a few more individuals.

Handling haplotypes generated by some recombinations

The PPH problem is motivated by the coalescent model without recombination. However, the programs can be useful for solving the HI problem when the underlying haplotypes were generated by a history involving some amount of recombination. In that case, it is not expected that the entire data will have a PPH solution, but some intervals in the data might have one. We can use one of the PPH programs to find maximal intervals in the input genotype sequences which have unique PPH solutions. Starting from position 1, we find the longest interval in the genotype data which has a unique PPH solution. We do this using binary search, running a PPH program on each interval specified by the binary search. Let us say that the first maximal interval extends from position 1 to position i. We output that interval, and then move to position 2 to determine if there is an interval that extends past i containing a unique PPH solution. If so, we find the maximal interval starting at position 2, and output it. Otherwise, we move to position 3, etc. We continue in this way to output a set of maximal intervals, each of which contains a unique PPH solution. This also implicitly finds, for each starting position, the longest interval starting at that position that contains a unique PPH solution.

In principle, the intervals that are output could overlap in irregular, messy ways. However, we have observed that this is rarely the case. Generally, the output intervals do not overlap, or two intervals overlap at one site, i.e., the right end of one interval may overlap in one position with the left end of the next interval. This provides a clean decomposition of the data into a few intervals where in each, the data has a unique PPH solution.

The most striking thing is that when the recombination rate is moderate, the accuracy of the PPH solutions inside each interval, compared to the original haplotypes, is very high. There are many ways that such a decomposition can be used. The most obvious is to reduce the amount of laboratory work that is needed to fully determine the correct haplotypes. For example, in a problem with 100 sites and 10 intervals, we can form new shorter genotype vectors based on one site per interval, hence 10 sites. If the correct haplotype pairs for these shorter genotype sequences are determined, we can combine that information with the (assumed correct) haplotype pairs determined in each interval by a PPH program. The lab effort is reduced to one tenth of what it would be to determine the haplotypes from 100 sites. That is a huge reduction in laboratory effort.

5 Pure Parsimony

5.1 The Pure-Parsimony-criteria

One natural approach to the HI problem that is often mentioned in the literature is called the *Pure-Parsimony* approach[2]: Find a solution to the HI problem that minimizes the total number of distinct haplotypes used.

For example, consider the set of genotypes: 02120, 22110, and 20120. There are HI solutions for this example that use six distinct haplotypes, but the solution 00100, 01110; 01110, 10110; 00100, 10110, for the three genotype vectors respectively, uses only the three distinct haplotypes 00100, 01110, and 10110.

The Pure parsimony criteria reflects the fact that in natural populations, the number of observed distinct haplotypes is vastly smaller than the number of combinatorially possible haplotypes, and this is also expected from population genetics theory. We also saw this in the experiments reported in [17]. Moreover, the parsimony criteria is to some extent involved in existing computational methods for haplotype inference. For example, some papers have tried to explain Clark's method [6] in terms of parsimony [29], although the role of parsimony is not explicit in the method, and the haplotyping program PHASE [33] has been explained in terms of the parsimony criteria [9]. However, the indirect role of the parsimony criteria in those methods, and the complex details of the computations, makes it hard to see explicitly how the parsimony criteria influences the computation. This makes it difficult to use those methods to evaluate the effectiveness of the parsimony criteria as a genetic model.

In [19] we detail how to compute, via integer-linear-programming, an HI solution that minimizes the number of distinct haplotypes, i.e., is guaranteed to solve the Pure-Parsimony problem. However, the worst-case running time increases exponentially with the problem size, so the empirical issue is whether this approach is practical for problem instances of current interest in population-scale genomics. The paper shows wasy to improve the practicality of the basic integer programming formulation in a way that is very effective in practice. Extensive experimentation was done to show the time and memory practicality of the method, and to compare its accuracy against existing HI methods that do not explicitly follow the Pure-parsimony criteria.

Empirically, the end result is that for haplotyping problem instances of

[2]This approach was suggested to us Earl Hubbell, who also proved that the problem of finding such solutions is NP-hard [24].

current interest, Pure-parsimony *can* be computed efficiently in most cases. However, it's accuracy is somewhat inferior to the solutions produced by the program PHASE, although this depends on the number of sites and the level of recombination.

In more detail, the practicality and accuracy of our approach depend on the level of recombination in the data (the more recombination, the more practical but less accurate is the method). We show here that the Pure-Parsimony approach is practical for genotype data of up to 30 sites and 50 individuals (which is large enough for practical use in many current haplotyping projects). Up to moderate levels of recombination, the haplotype calls are 80 to 95 percent correct, and the solutions are generally found in several seconds to minutes, except for the no-recombination case with 30 sites, where some solutions require a few hours.

These results are a validation of the genetic model implicit in the Pure-Parsimony objective function, for a Purely randomly picked solution to the HI problem would correctly resolve only a minuscule fraction of the genotypes.

Recently the paper [35] gave experimental results on a pure parsimony approach to the haplotype inference problem, solving it by branch and bound instead of integer programming. Theoretical results on pure parsimony appear in [26].

6 Adding Recombination into the model

The perfect phylogeny haplotyping problem is motivated by the assumption that in some interesting cases (haplotype blocks for example) the evolution of the underlying haplotypes can be represented on a perfect phylogeny. To increase the applicability of the model, we want to relax that stringent assumption. This was done in a heuristic way in [10] where they observed that the haplotypes reported in [8] cannot generally be derived on a perfect phylogeny, but with the removal of only a small number of individuals, the remaining sequences can be derived on a perfect phylogeny. Those observations validate the underlying, ideal perfect phylogeny model and the utility of having an efficient, clean solution for the PPH problem, but they also highlight a need to introduce more robust models of haplotype evolution into the haplotyping model. Probably, the most important modification of the perfect phylogeny model would be to allow sequences to recombine, as is common in the real evolution of sequences in populations. When recombination is allowed, the history of the sequences no longer fits a tree-like

structure, but rather a *network* is required to represent the derivation of the haplotypes.

Once recombination is allowed into the underlying model, we can define the formal analog of the PPH problem: given a set of genotypes whose underlying haplotypes were believed to have evolved on a network with recombination, find a set of explaining haplotypes (for the genotypes) which can be derived on a phylogenetic network with a small amount of recombination. We call this the "Phylogenetic Network Haplotyping (PNH) Problem". By specifying a small amount of recombination, we limit the number of distinct haplotypes that can be created on the network. As discussed earlier, the number of haplotypes observed in real populations tends to be relatively small.

The solutions to the PPH problem exploit basic theorems about when a set of haplotypes can be derived on a perfect phylogeny. Those theorems are relatively simple to state and crucial to the PPH solutions. The PPH problem has a clean solution partly because of the simplicity of the necessary and sufficient condition for a set of haplotypes to be derived on a perfect phylogeny. The PNH problem is harder, because we have no analogous, simple theorems about phylogenetic networks. Hence, we have recently focussed attention on the derivation of haplotypes on phylogenetic networks, with the ultimate goal of applying what we have learned to the PNH problem. We have focussed mostly on phylogenetic networks with constrained recombination. Results to date can be found in in [21, 20, 22].

References

[1] V. Bafna, D. Gusfield, G. Lancia, and S. Yooseph. Haplotyping as perfect phylogeny: A direct approach. Technical report, UC Davis, Department of Computer Science, 2002.

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

[3] R. E. Bixby and D. K. Wagner. An almost linear-time algorithm for graph realization. *Mathematics of Operations Research*, 13:99–123, 1988.

[4] R.H. Chung and D. Gusfield. Empirical exploration of perfect phylogeny haplotyping and haplotypers. In *Proceedings of COCOON 03* -

The 9'th International Conference on Computing and Combinatorics, volume 2697 of *LNCS*, pages 5–19, 2003.

[5] R.H. Chung and D. Gusfield. Perfect phylogeny haplotyper: Haplotype inferral using a tree model. *Bioinformatics*, 19(6):780–781, 2003.

[6] A. Clark. Inference of haplotypes from PCR-amplified samples of diploid populations. *Mol. Biol. Evol*, 7:111–122, 1990.

[7] A. Clark, K. Weiss, and D. Nickerson et. al. Haplotype structure and population genetic inferences from nucleotide-sequence variation in human lipoprotein lipase. *Am. J. Human Genetics*, 63:595–612, 1998.

[8] M. Daly, J. Rioux, S. Schaffner, T. Hudson, and E. Lander. High-resolution haplotype structure in the human genome. *Nature Genetics*, 29:229–232, 2001.

[9] P. Donnelly. Comments made in a lecture given at the DIMACS conference on Computational Methods for SNPs and Haplotype Inference, November 2002.

[10] E. Eskin, E. Halperin, and R. Karp. Large scale reconstruction of haplotypes from genotype data. Proceedings of RECOMB 2003, April 2003.

[11] E. Eskin, E. Halperin, and R. Karp. Efficient reconstruction of haplotype structure via perfect phylogeny. Technical report, UC Berkeley, Computer Science Division (EECS), 2002.

[12] M. Fullerton, A. Clark, Charles Sing, and et. al. Apolipoprotein E variation at the sequence haplotype level: implications for the origin and maintenance of a major human polymorphism. *Am. J. of Human Genetics*, pages 881–900, 2000.

[13] F. Gavril and R. Tamari. An algorithm for constructing edge-trees from hypergraphs. *Networks*, 13:377–388, 1983.

[14] D. Gusfield. Efficient algorithms for inferring evolutionary history. *Networks*, 21:19–28, 1991.

[15] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[16] D. Gusfield. A practical algorithm for deducing haplotypes in diploid populations. In *Proceedings of 8'th International Confernece on Intelligent Systems in Molecular Biology*, pages 183–189. AAAI Press, 2000.

[17] D. Gusfield. Inference of haplotypes from samples of diploid populations: complexity and algorithms. *Journal of computational biology*, 8(3), 2001.

[18] D. Gusfield. Haplotyping as Perfect Phylogeny: Conceptual Framework and Efficient Solutions (Extended Abstract). In *Proceedings of RECOMB 2002: The Sixth Annual International Conference on Computational Biology*, pages 166–175, 2002.

[19] D. Gusfield. Haplotype inference by pure parsimony. In R. Baeza-Yates, E. Chavez, and M. Chrochemore, editors, *14'th Annual Symposium on Combinatorial Pattern Matching (CPM'03)*, volume 2676 of *Springer LNCS*, pages 144–155, 2003.

[20] D. Gusfield, S. Eddhu, and C. Langley. Optimal, efficient reconstruction of phylogenetic networks with constrained recombination. J. Bioinformatics and Computational Biology, to appear.

[21] D. Gusfield, S. Eddhu, and C. Langley. Efficient reconstruction of phylogenetic networks (of SNPs) with constrained recombination. In *Proceedings of 2'nd CSB Bioinformatics Conference*. IEEE Press, 2003.

[22] D. Gusfield, S. Eddhu, and C. Langley. The fine structure of galls in phylogenetic networks with recombination. Technical report, UC Davis, Department of Computer Science, 2003.

[23] L. Helmuth. Genome research: Map of the human genome 3.0. *Science*, 293(5530):583–585, 2001.

[24] E. Hubbel. Personal Communication, August 2000.

[25] R. Hudson. Gene genealogies and the coalescent process. *Oxford Survey of Evolutionary Biology*, 7:1–44, 1990.

[26] G. Lancia, C. Pinotti, and R. Rizzi. Haplotyping populations: Complexity and approximations, technical report dit-02-082. Technical report, University of Trento, 2002.

[27] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, USA, 1976.

[28] S. Lin, D. Cutler, M. Zwick, and A. Cahkravarti. Haplotype inference in random population samples. *Am. J. of Hum. Genet.*, 71:1129–1137, 2003.

[29] T. Niu, Z. Qin, X. Xu, and J.S. Liu. Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms. *Am. J. Hum. Genet*, 70:157–169, 2002.

[30] S. Orzack, D. Gusfield, , J. Olson, S. Nesbitt, and V. Stanton. Analysis and exploration of the use of rule-based algorithms and consensus methods for the inferral of haplotypes. *Genetics*, 165:915–928, 2003.

[31] S. Orzack, D. Gusfield, and V. Stanton. The absolute and relative accuracy of haplotype inferral methods and a consensus approach to haplotype inferral. Abstract Nr 115 in Am. Society of Human Genetics, Supplement 2001.

[32] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[33] R. Ravi. Personal Communication.

[34] M. Stephens, N. Smith, and P. Donnelly. A new statistical method for haplotype reconstruction from population data. *Am. J. Human Genetics*, 68:978–989, 2001.

[35] S. Tavare. Calibrating the clock: Using stochastic processes to measure the rate of evolution. In E. Lander and M. Waterman, editors, *Calculating the Secretes of Life*. National Academy Press, 1995.

[36] L. Wang and L. Xu. Haplotype inference by maximum parsimony. *Bioinformatics*, 19:1773–1780, 2003.