Empirical Exploration of Perfect Phylogeny Haplotyping and Haplotypers

Ren Hua Chung and Dan Gusfield*

Computer Science Department, University of California, Davis, Davis CA 95616, USA
gusfield@cs.ucdavis.edu

Abstract. The next high-priority phase of human genomics will involve the development of a full *Haplotype Map* of the human genome [15]. 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 determine haplotype pairs from genotype data. An approach to this problem based on viewing it in the context of perfect phylogeny was introduced in [14] along with an efficient solution. A slower (in worst case) variation of that method was implemented [3]. Two simpler methods for the perfect phylogeny approach that are also slower (in worst case) than the first algorithm were later developed [1, 7]. We have implemented and tested all three of these approachs in order to compare and explain the practical efficiencies of the three methods. We discuss two other empirical observations: a strong phase-transition in the frequency of obtaining a unique solution as a function of the number of individuals in the input; and results of using the method to find non-overlapping intervals where the haplotyping solution is highly reliable, as a function of the level of recombination in the data. Finally, we discuss the biological basis for the size of these tests.

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.

* Research Supported by NSF grants DBI-9723346 and EIA-0220154

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 [4, 5, 8, 23, 13, 22, 20, 21]. 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".

2 Perfect Phylogeny

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.

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 sequences (or haplotypes) in sampled individuals [24, 16]. The key observation is that "In the absence of recombination, each sequence has a single ancestor in the previous generation." [16].

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. This assumption is usually made without contention in SNP data.

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. See [24] 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 an (unrooted) *perfect phylogeny* [11, 12]:

Definition Let B be an $2n$ by m 0-1 (binary) matrix, and V be an m-length binary string. A *rooted perfect phylogeny for* B 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.

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 columns that label the edges along the unique path from the root to leaf i specify the columns of B where row i has a value that is different from the value of V in that column. In other words, knowing V, that path is a compact representation of row i.

If we don't know V at input, the *unrooted perfect phylogeny* problem is to determine a V so that B has perfect phylogeny rooted at V.

The classic Theorem of Perfect Phylogeny is that a binary matrix B has a perfect phylogeny if and only if for each pair of columns, there are no four rows in those columns with values (0,0), (0,1), (1,0) and (1,1). Moreover, if the columns of B are distinct, then there is only one perfect phylogeny for B. Note that an edge to a leaf need not have a column label.

The above condition is known as the "Four-Gametes Test" in the population genetics literature, and is known as the "Compatibility Test" in the phylogenetic literature.

2.1 The Perfect Phylogeny Haplotype (PPH) Problem

Under the coalescent model of haplotype evolution, the HI problem now has precisely the following combinatorial interpretation:

Given a set of n genotypes G, find an HI solution consisting of at most $2n$ distinct haplotype vectors B, such that the vectors in B fit a perfect phylogeny. This is the unrooted version of the PPH problem. See Figure 1 for a trivial example.

If a root sequence V is specified, then the perfect phylogeny is required to have the root sequence of V. The two versions of the problem are actually equivalent in the sense that an instance of one variation can be reduced to the other variation, so that an algorithm for one variation can be used for both variations.

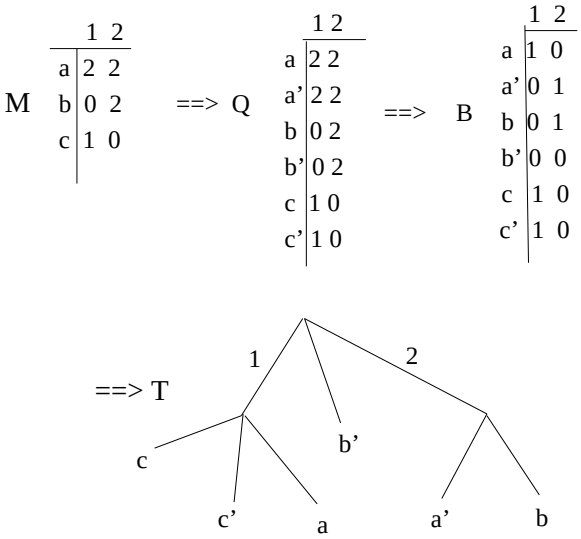

Fig. 1. A simple example where there are two solutions to the HI problem, but only the one shown solves the PPH problem. Matrix Q is created from matrix M, doubling the rows to prepare for the PPH solution B.

3 Three Algorithms and programs for the PPH Problem

In this section we briefly describe three algorithms and three programs (GPPH, HPPH and DPPH) for the PPH problem. The first algorithm for the PPH problem was developed in [14]. After that publication, two additional methods were developed and presented [1] and [7]. Program GPPH follows the basic approach presented in [14], with some modifications as detailed below. Programs DPPH and HPPH follow the algorithms developed in [1] and [7], respectively.

The three programs are available at: cs.ucdavis.edu/~gusfield

3.1 Solution by Graph Realization: Program GPPH

The first algorithm for the PPH problem, given in [14], is based on reducing the problem to a well-studied problem in graph theory.

Let E_r a set of r distinct integers. A "path set" is an *unordered* subset P of E. A path set is "realized" in a undirected, edge-labeled tree T consisting of r edges, if each edge of T is labeled by a distinct integer from E_r, and there is a contiguous path in T whose labels consist of the integers in P. For simplicity, we refer to each integer in E_r as an "edge", since in the tree T that we seek, each edge will be uniquely labeled with an integer in E_r. Note that since P is unordered, its presentation does not specify or constrain the order that those edges appear in T. In quite different terms, from the 1930's to the 1960's Whitney and Tutte and others studied and solved the following problems:

The Graph Realization Problem Given E_r and a family $\Pi = P_1, P_2, ..., P_k$ of path sets, find an undirected tree T in which each path set is realized, or determine that no such tree exists. Further, determine if there is only one such T, and if there is more than one, characterize the relationship between the realizing trees.

In [14] we reduce the PPH problem to the graph realization problem, and have implemented this approach in a program called here GPPH[1]. Program GPPH implements a slightly different reduction than is described in [14]. The actual reduction is detailed at:

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

After reducing a problem instance, program GPPH solves the graph realization instance, using a variation [10] of Tutte's classic algorithm for graph realization [25]. In GPPH, we implement the reduction and the graph realization solution separately. The reduction takes $O(nm)$ time to create an instance of the graph realization problem, and the graph realization module runs in $O(nm^2)$ time, and is completely general, not incorporating any particular features of the PPH problem. Having a general solution to the graph realization problem gives us a program that can be used for other applications of graph realization besides the PPH problem. It is of interest to see how this general approach performs relative to methods that incorporate particular insights about the PPH problem.

Tutte's graph realization algorithm is a method that recursively solves graph realization problems for a subset E of E_r, and a family of path sets $\Pi(E)$, formed from the family Π by restricting each path set in Π to E. Given E and $\Pi(E)$, and knowledge of what decisions have already been made, the algorithm chooses an edge e (called here the "pivot" edge) from E and determines, by examining the path sets in $\Pi(E)$ and using general insights about paths in trees, whether there are edges in E that must be on one particular side of e or the other side of e. It also determines whether there are pairs of edges in E that must be separated

[1] This program was previously called PPH, but now that multiple programs exist for the PPH *problem*, each is given a distinct name.

by e, and if so, whether these pairs form a bipartition. To do that, it constructs a graph $G(E)$ containing one node for each edge in E, and one undirected arc between every pair of edges (e', e'') in E that must be separated by e. If $G(E)$ is bipartite, Tutte's algorithm arbitrarily puts the edges from one side of $G(E)$ on one side of e, and the edges from the other side of $G(E)$ on the other side of e. It then recursively solves the graph realization problem for the two sets of edges on either side of e, and when those two subproblems have been solved (returning two subtress), the algorithm determines how to attach those two subtrees to e. Hence, the recursion is central to the method. If $G(E)$ were determined not to be bipartite, then the algorithm correctly determines that there is no tree realizing all the path sets in E_r.

There are other solutions to the graph realization problem not based on Tutte's algorithm. The method in [2] is based on a general algorithm due to Lofgren, and runs in $O(nm\alpha(nm))$ time, where α is the inverse-Ackerman function. That algorithm is the basis for the worst-case time bound established in [14], but we found it to be too complex to implement. In [14] 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.

In program GPPH, about 1000 lines of C code were written to implement the reduction part of the method and about 4000 lines of C code were written to implement the graph realization part of the method.

3.2 Algorithm and program HPPH

Although obtained independently of Tutte's method, the method of Eskin, Halperin and Karp [7] can be viewed as a specialization, to the PPH problem, of Tutte's general graph realization method. The method is developed under the assumption that the root V is known, and it exploits the rooted nature of the PPH problem (even when V is not given, a solution to the PHH problem is a rooted tree) to find simpler, specific rules to use when picking a pivot edge e, and to determine the placement of the other edges to e. In particular, it builds a tree top-down from the root, by choosing for a pivot edge e, a "maximal" edge, i.e., one that is guaranteed not to be below any of the edges that are not yet in the tree. Finding such a maximal pivot edge is a simple matter in the context of the PPH problem using the "leaf-count" idea in [14]. Because of the maximality, the general operation in Tutte's algorithm of determining which edges are on which side of e becomes simpler. One determines for each other edge e', whether e' must be placed below e (on the same path from the root as e), or must not be placed below e, or whether both placements are permitted. As in Tutte's method, edges in the last category are considered pairwise to find pairs of edges where exactly one of the pair must be placed below e, but either one of the pair can be the edge chosen to be below e. These pairs are represented by a graph, and the algorithm checks that the graph is bipartite. If so, one set of the bipartition is arbitrarily chosen to be placed below e and the other chosen to not be placed below e. After the algorithm makes that decision, it recurses until a tree is built or no further progress can be made. The algorithm can also

implicitly represent the set of all PPH solutions by noting where it could have made alternative choices.

We call the program implementing this method the HPPH program. Because the HPPH program is a specialization of Tutte's general method using simpler rules to determine and implement the pivot, it is expected to run faster in practice than the GPPH program. Moreover, the top-down structure of the method has the consequence that the recursion used to describe the algorithm in [7] is actually tail-recursion, so that the method can be implemented iteratively without any explicit recursion. This can speed up the program, particularly for large data sets, and we have implemented HPPH without recursion. Viewed as a specialization of Tutte's general method, it is of interest to see how much of a speed-up the simpler pivot rules provide. About 1500 lines of C code were written to implement HPPH.

Segue to DPPH One of the central details in HPPH (and also in the DPPH method to be discussed) is that for certain pairs of columns (c, c') in M, we can determine from looking at those columns alone (without *any* other information from M) whether the edge labeled with c must be placed below the edge labeled with c', or the c' edge must be placed below the c edge, or neither edge can be placed below the other. This is called a "forced relationship". The HPPH program spends $O(nm^2)$ time at the start to examine the columns in pairs to find any forced relationships (actually it determines another weaker relationship as well). After this first stage, the algorithm does operations whose best time-bound is $O(nm^2)$.

3.3 Algorithm and Program DPPH

The method in DPPH [1] 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. These insights are exploited to avoid any recursion or iteration, in contrast to the GPPH and HPPH programs. Rather, after the initial $O(nm^2)$-time stage, where all the forced relationships are found, the algorithm finds a PPH solution in $O(q^2)$ time, where q is the minimum of n and m, by a simple depth-first search in a graph that encodes the forced relationships and the additional decisions that remain to be made. Moreover, the graph represents the set of all solutions in a simple way. We will not fully detail that here, but we can explain how the graph determines the number of solutions, assuming there is a solution.

We use the column indices in M for vertex labels, and let GF be a graph with a node c for each column c in M, and an edge between two nodes c and c' if there is a row in M with a 2 in both columns c and c'. Let k be the number of connected components of GF. Now mark any edge (c, c') where c and c' are in a forced-relationship, and let z be the number of connected components of the subgraph of GF induced by the marked edges. Then the number of solutions to the PPH problem, assuming there is one, is exactly $2^{(z-k)}$. Hence it is actually easier to count the number of solutions than to find one.

Because of the deeper insights encoded in DPPH, we expect it will be the fastest method in practice. About 1500 lines of C code were written to implement DPPH.

4 Generating the Test Data

We used the program created by Richard Hudson [17] to generate the haplotypes. That program is the widely-used standard for generating sequences that reflect the coalescent model of SNP sequence evolution. The program allows one to control the level of recombination through a parameter r. When r is set to zero, there is no recombination, and hence the haplotypes produced are guaranteed to fit a perfect phylogeny.

After obtaining $2n$ haplotypes from Hudson's program, the haplotypes are randomly paired to create the n genotype vectors given to the three programs. The three PPH programs were tested with thousands of datasets and all of the outputs were verified to see that perfect phylogenetic trees were created and that the output haplotypes explain the input genotypes. In the output file, the input genotypes and their corresponding haplotypes are reported. The output file also reports the perfect phylogenetic tree and whether the tree is unique or not. If it is not, the number of different trees which fit the input data is reported. Program DPPH also produces the implicit representation of the set of all solutions in a simple format.

5 Performance comparison

For genotype data with 50 individuals and 50 sites, the three PPH programs typically solve the problem in less than one second on a computer equipped with AMD K6 1.33GHz CPU and 256 MB RAM. However, GPPH spends notably longer than the others when the data is large. The relative speed of program DPPH compared to the other two methods, increases as the data size increases. It typically spends under two minutes to handle genotype data with 500 individuals and 1000 sites. This makes it possible to handle long sequences with large samples.

Our basic expectation that DPPH would be the fastest, followed by HPPH and GPPH in that order, was confirmed, as is shown in Figure 2. We should note that our implementation of HPPH included a number of minor ideas not made explicit in the original paper [7], that considerably sped up the execution. Also, Shibu Yooseph [27] has implemented DPPH and reports running times that are two to three times faster than our implementation. We implemented the graph realization program used in GPPH literally as described in [10] and did not attempt any optimizations, although many are clearly possible.

		Average Running Times (seconds)		
sites	individuals	GPPH	DPPH	HPPH
30	50	0.6574	0.0206	0.0215
100	100	1.5697	0.3216	0.37345
300	150	9.31835	3.041	4.497
500	250	36.12485	11.5275	21.5901
1000	500	256.1847	75.5935	189.4267
2000	1000	2331.167	639.93	1866.569

Fig. 2. The comparison of the running times of three methods. Each number is the average of 20 datasets.

6 Related Topics

6.1 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? The general issue of the number of individuals needed in a study was raised in [14]. Theoretical and empirical results on this question, addressing the number of individuals needed in a study as a function of the number of distinct haplotypes in the population, appear in [9]. Here, we report on 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.

The input datasets are generated by Hudson's program [17] discussed earlier. While generating the data, we added an essential, biologically relevant restriction that every mutation in the tree generating the haplotypes must be on an edge that has at least 5 percent of total leaves beneath it. That is, at that site, the least frequent allele must appear in at least 5 percent of individuals in the sample. Without adding this restriction, the data are not biologically realistic, and the frequency of a unique PPH solution remains low even when the ratio of genotypes to sites is high. The table in Figure 3 shows the frequency, for datasets with 50 and 100 sites respectively, that a unique PPH solution is observed in 5000 datasets, as the number of individuals in the sample varies. Note that the number of individuals is the number of input genotype vectors, so that the underlying tree generating the haplotypes has twice that many leaves.

The table shown in Figure 3 shows a phase transition between 19 and 20 individuals. This is almost certainly related to the 5% rule. When the number of individuals (genotypes) reaches 20, the number of leaves in the underlying tree is 40, where the 5% rule requires that each mutation (site) be placed on an edge with at least two leaves below it. Hence a mutation on an edge attached to a leaf is prohibited. Before then, a mutation could be placed on such an edge, and when that happens, the solution is very unlikely be unique. A more

sites	individuals	frequency	sites	individuals	frequency
50	10	0.0000	100	10	0.0000
50	19	0.0002	100	19	0.0000
50	20	0.7030	100	20	0.7260
50	28	0.7050	100	28	0.7236
50	30	0.9520	100	30	0.9774
50	40	0.9926	100	40	0.9966
50	50	0.9974	100	50	0.9994
50	60	0.9990	100	60	0.9996
50	70	0.9994	100	70	0.9998
50	80	0.9998	100	80	1.0000
50	90	0.9998			
50	100	1			

Fig. 3. The frequency of a unique PPH solution increases when the number of individuals increases. 5000 datasets were simulated for each entry.

interesting phase transition occurs between 20 and 30 individuals. Notice that the frequency of uniqueness is close to one before the number of individuals reaches the number of sites. This may seem unexpected, and is explained by the fact that there are edges in the tree that receive more than one mutation (site) in Hudson's program. Hence the ratio of the number of individuals to the number of edges in the tree that receive one or more mutations, is higher than suggested by the results shown here. However, the biologically relevant comparison is of the number of individuals to the number of sites of interest, and so the good news is that the number of individuals needed in a sample, in order to have a high probability of a unique PPH solution, is relatively low compared to the number of sites.

6.2 Handling haplotypes generated with 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. We first find the longest interval in the genotype data, starting from position 1, 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.

A program called PPHS, which is available at the website mentioned earlier, was implemented to find the maximal intervals. We performed several experiments for the genotype data with different recombination rates, using 100 genotypes and 100 sites. The input data for the experiments was again generated by Hudson's program [17]. Therefore, we were able to check if the output haplotypes differed from the original haplotypes, in any interval. The results are shown in Figure 4. The most striking result 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. In fact, when $r = 4$, in each of the fifteen runs described in Figure 4, the unique PPH solution found by the algorithm precisely recreated the input haplotypes in each interval. That is, the PPH program found the correct haplotype pairs perfectly in each interval. One might think that this must always be true, since the PPH solution is unique in each interval, but genotypes that can be explained with haplotypes that fit a perfect phylogeny need not have been generated that way.

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 with 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 laboratory effort is reduced to one tenth of what it would be to determine the haplotypes from 100 sites. Another approach is to input the shorter genotype sequences to one of the statistical-based methods for haplotype inference. These methods can handle a moderate level of recombination and are generally quite accurate, but their running times increase greatly with an increasing number of sites.

7 How large should the test data be?

In this paper we tested programs and data with up to 2000 sites. This is a larger number of SNPs than has so far been observed to fit the perfect phylogeny model. Here we discuss the question of how many SNP sites should be included in tests of PPH programs.

We define a set of binary sequences as "tree-compatible" if the sequences pass the four-gametes test. That is, the data do not have two positions (sites) where four sequences in the set contain all four combinations 0,0; 0,1; 1,0, and 1,1. We

	r = 4		r = 16		r = 40	
Experiments	Errors	Intervals	Errors	Intervals	Errors	Intervals
No. 1	0	7	0	17	9	18
No. 2	0	7	1	17	26	17
No. 3	0	5	3	20	0	20
No. 4	0	5	1	16	0	22
No. 5	0	4	2	14	5	24
No. 6	0	3	1	12	1	22
No. 7	0	7	0	18	7	20
No. 8	0	4	0	14	0	25
No. 9	0	7	0	18	12	20
No. 10	0	10	0	15	1	24
No. 11	0	9	1	12	5	19
No. 12	0	5	0	18	1	18
No. 13	0	7	0	16	27	22
No. 14	0	8	0	14	1	25
No. 15	0	10	0	19	0	24
Average	0	6.5	0.6	16	6.3	21.3

Fig. 4. Fifteen experiments with 100 individuals and 100 sites, performed with three different recombination rates. r is the recombination rate used in Hudson's program [17]. When r is high, the probability of recombination is high. The Interval count is the number of intervals output by program PPHS. The error count is the total number of intervals where the given unique PPH solution is not correct for that interval. Hence the number of errors reported can be larger than the number of intervals.

define a "tree-compatible interval" as an interval where the subsequences are tree-compatible. Sequences that are tree-compatible can be derived on a perfect phylogeny. Therefore, in considering the size of relevant input to PPH programs, the key question is: What is the longest tree-compatible interval that one could reasonably expect to find in a set M of binary-encoded biological sequences (SNPs, or other binary sequences) of current or future interest? We define that length as m^*.

The correct value of m^* is certainly unknown at present, and the existing literature related to this issue (mainly from studies of haplotype structure in humans, and studies of linkage disequilibrium in a few other organisms) represents a minuscule fraction of the molecular diversity studies that are desired and that are expected to be conducted in the future. However, there is already good evidence establishing that m^* is much larger than 30 (a number suggested by some of the studies in humans, for example [6]).

Relevant data can come from two sources: actual sequence and SNP data that can be directly examined for tree-compatible intervals, and less direct studies of linkage disequilibrium (LD). LD causes (or is defined by) a high correlation between the occurrences of alleles at two sites. If f_A is the frequency of allele A at one site, and f_B is the frequency of allele B at a second site, and f_{AB} is the joint frequency of those alleles at the two sites, LD at the two sites can be measured by the deviation of f_{AB} from $f_A \times f_B$. Assuming infinite sites,

long intervals of high LD suggest long tree-compatible intervals, since high LD is generally caused by little recombination between the two sites. LD can be measured without directly determining whether the interval is tree-compatible, and so LD is a more indirect, but much more available, indicator of m^* than is provided by the full SNP sequences in the sample. Presently, very few studies have determined full SNP sequences in large populations, so data on LD is very central in estimating m^*.

In published data on humans, there is considerable variance in the level of LD observed. For example, surveys of LD suggest that among Nigerians, high levels of linkage disequilibrium extend to intervals of only 5 Kb, but in Northern Europeans, high levels extend to intervals of 60 to 80 Kb. Depending on the number of individuals in the survey, that could translate to tree-compatible intervals of between 5 and 80 SNPs. In even more homogeneous populations, for example in Finland or Iceland or Pennsylvania Dutch, even longer tree-compatible intervals of SNPs are expected. Generally, structured subpopulations which are the result of recent migration, historical bottlenecks (reduction of the population size), or admixture (the recent mixing of two distinct populations) are expected to have even longer tree-compatible intervals, and the exact lengths are not yet known. Smaller, more local populations should have even longer tree-compatible intervals. Moreover, structured, local, subpopulations, are very important in human genetics research, so that even if 30 were the human-average number of tree-compatible SNP sites in an interval, there are important studies where one expects, and will search for, longer tree-compatible intervals. Domesticated plants and animals are expected to have longer tree-compatible intervals than humans, and the few studies that have been done are consistent with this expectation [19].

It is also important to understand how the data reported in [6] were obtained. Those studies were looking for long intervals in humans of high LD containing a small number of haplotypes in the given sample. SNP sites were sampled in the genome at a density that was sufficient to identify these long intervals, but once found, the researchers were not interested in determining the total number of SNP sites in the intervals. Therefore, typically under 30 SNPs were found in intervals up to 200,000 nucleotides long. But a human interval of that length is expected to have between 200 and 400 SNPs, and so it is incorrect to interpret the number of SNPs in an interval reported in [6] as an estimate of m^*. Moreover, the number of SNPs found will also be a function of the size of the sample. Fine scale linkage and association mapping aimed at identification of specific genes and variants will often lead to the determination of many more SNPs in candidate regions. And large scale resequencing on a genomic scale will also produce high densities of SNPs throughout the genome [18].

Moreover, there is variation in the density of SNPs, independent of the amount of recombination in a region. For example, recent investigation of the laboratory mouse genome [26] found long regions of the genome that contained 45 SNPs per 10Kb, and other long regions that contained only 1 SNP per 10Kb. Hence, it is expected that in the mouse genome we could encounter tree-

compatible regions with the same number of nucleotides, where one region has many more SNPs than the other.

In general, the value of m^* depends both on the diversity of biological sequences (from different organisms, regions of the genome, populations and subpopulations) that are of importance and that will be, or have been, studied, and the number of sequences in those studies. More fundamentally, for any given sample, the length of the longest tree-compatible interval is a function of the ratio of the (site) mutation rate, and the recombination rate in that sample. Among all the organisms, genomic regions, populations and subpopulations that are of interest, that ratio is expected, and has already been seen, to vary over a large range [18]. Hence it is not correct to conclude that m^* has been established from the limited, specific studies recently conducted in humans.

In short, the correct value of m^* is unknown, and there is very little that is known, empirically or theoretically, that establishes good bounds for m^*. There is enormous diversity in biology, with a vast number of unexplored organisms, populations, subpopulations and genomic regions of interest. Any suggestion that m^* is already known ignores this. More specifically, the suggestion that m^* is bounded by 30 does not reflect the little data that is already known, and the understanding that compared to the human studies recently done, much longer tree-compatible regions should exist in sequences from subpopulations and organisms whose diversity has not yet been studied.

8 Acknowledgements

Thanks to population geneticists Chuck Langley, Peter Morrell and Steven Orzack for advice on the discussion in Section 7.

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. July 17, 2002.
2. R. E. Bixby and D. K. Wagner. An almost linear-time algorithm for graph realization. *Mathematics of Operations Research*, 13:99–123, 1988.
3. R.H. Chung and D. Gusfield. Perfect phylogeny haplotyper: Haplotype inferral using a tree model. *Bioinformatics*, 19(6):780–781, 2003.
4. A. Clark. Inference of haplotypes from PCR-amplified samples of diploid populations. *Mol. Biol. Evol*, 7:111–122, 1990.
5. 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.
6. 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.
7. E. Eskin, E. Halperin, and R. Karp. Efficient reconstruction of haplotype structure via perfect phylogeny. Technical report, UC Berkeley, Computer Science Division (EECS), August, 2002.

8. 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.
9. S. Cleary and K. St. John. Analysis of Haplotype Inference Data Requirements. Preprint, 2003.
10. F. Gavril and R. Tamari. An algorithm for constructing edge-trees from hypergraphs. *Networks*, 13:377–388, 1983.
11. D. Gusfield. Efficient algorithms for inferring evolutionary history. *Networks*, 21:19–28, 1991.
12. D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.
13. D. Gusfield. Inference of haplotypes from samples of diploid populations: complexity and algorithms. *Journal of computational biology*, 8(3), 2001.
14. 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.
15. L. Helmuth. Genome research: Map of the human genome 3.0. *Science*, 293(5530):583–585, 2001.
16. R. Hudson. Gene genealogies and the coalescent process. *Oxford Survey of Evolutionary Biology*, 7:1–44, 1990.
17. R. Hudson. Generating samples under the Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–338, 2002.
18. C. Langley. U.C. Davis Dept. of Evolution and Ecology. Personal Communication, 2003.
19. J.Z. Lin, A. Brown, and M. T. Clegg. Heterogeneous geographic patterns of nucleotide sequence diversity between two alcohol dehydrogenase genes in wild barley (Hordeum vulgare subspecies spontaneum). *PNAS*, 98:531–536, 2001.
20. S. Lin, D. Cutler, M. Zwick, and A. Cahkravarti. Haplotype inference in random population samples. *Am. J. of Hum. Genet.*, 71:1129–1137, 2003.
21. 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.
22. 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.
23. 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.
24. 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.
25. W.T. Tutte. An algorithm for determining whether a given binary matroid is graphic. *Proc. of Amer. Math. Soc*, 11:905–917, 1960.
26. C. Wade and M. Daly et al. The mosaic structure of variation in the laboratory mouse genome. *Nature*, 420:574–578, 2002.
27. Shibu Yooseph. Personal Communication, 2003.