Inference of Haplotypes from Samples of Diploid Populations: Complexity and Algorithms

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

Appeared in J. Computational Biology August, 2001, Vol. 8 No. 3 p. 305-324

Abstract

The next phase of human genomics will involve large-scale screens of populations for significant DNA polymorphisms, notably single nucleotide polymorphisms (SNP's). Dense human SNP maps are currently under construction. However, the utility of those maps and screens will be limited by the fact that humans are diploid, and that it is presently difficult to get separate data on the two "copies". Hence genotype (blended) SNP data will be collected, and the desired haplotype (partitioned) data must then be (partially) inferred. A particular non-deterministic inference algorithm was proposed and studied in [3] and extensively used in [2]. In this paper, we more closely examine that inference method and the question of whether we can obtain an efficient, deterministic variant to optimize the obtained inferences. We show that the problem is NP-hard, and in fact Max-SNP complete; that the reduction creates problem instances conforming to a severe restriction believed to hold in real data [3]; and that even if we first use a natural exponential-time operation, the remaining optimization problem is NP-hard. However, we also develop, implement and test an approach based on that operation and (integer) linear programming. The approach works quickly and correctly on simulated data.

1 Introduction to SNP's and Haplotypes

The next phase of human genomics efforts will involve large-scale screens of human populations for significant DNA polymorphisms. The polymorphisms of greatest current interest are SNP's, *single nucleotide polymorphisms*, i.e., single nucleotide sites where at least two (of four) different nucleotides occur in a large percentage of the population. There is great hope that SNP-based screens will help identify the genetic elements of many complex diseases (diseases affected by more than a single

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

gene). Recently, ten of the largest pharmaceutical companies, along with several academic partners, have entered into a consortium to collectively find and release a map of 300,000 SNP sites in humans, at an estimated cost of forty-five million dollars. NIH has put thirty million dollars into a different program to find 100,000 human SNP sites [13, 19].

Once a dense SNP map is known, polymorphism screens in a population can be carried out to try to find genes related to disease susceptibility and drug response. There are several ways that dense SNP maps can be used to locate genes. In *association mapping* [18] a polymorphism state that is correlated with the occurrence of a particular disease (or other phenotype) in the population, suggests that the causal agent of the disease (gene, regulatory region, or aberration such as a long triplet repeat) is located near the polymorphism site. In the older *pedigree analysis*, it is the conservation of polymorphism state (and not any specific state) between certain relatives, that indicates a polymorphism site close to the causal agent. In either of these (and other) methods, the next step is to obtain detailed information (mostly DNA sequence data) in the region around the located SNP sites, from the individuals in the study, both those affected and unaffected by the disease of interest. Significant differences between the affected and unaffected individuals identify candidate causal agents.

In diploid organisms 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 region is called a *haplotype*, while a description of the conflated data on the two regions is called a *genotype*. In complex diseases (those affected by more than a single gene) it is much more informative to have haplotype data (identifying a set of gene alleles inherited together) than to have only genotype data. However, because polymorphism screens will be conducted on large populations, it is not feasible to examine the two regions separately and genotype data rather than haplotype data is usually obtained.

Various methods can be used to try to infer (partial) haplotype information from genotype data. One such inference method was proposed and studied by A. Clark [3] before the current interest in SNP's, but has been extensively used in the first large-scale study of haplotypes inferred from SNP data [2, 14], and more recently in additional studies of hapoltype variation [4]. There is great interest in the results of these first SNP/haplotype studies because of the great hope, and doubt, that SNP's can be used to make association mapping practical in identifying genes related to complex diseases[1, 22, 9].

2 Introduction to the algorithmic issues

In this paper, we examine more deeply the algorithmic issues raised by Clark's method to infer haplotypes from SNP data. In short, in Clark's method, data from m sites (SNP's) in n individuals is collected, where each site can have one of two states (alleles). For simplicity of description, assume the m sites are all on a single chromosome. 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. Determining the two haplotypes is also called "determining the linkage phase". 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. Hence, data from an individual with h hetrozygous sites (where both states are present) would be consistent with any of 2^{h-1} haplotype pairs.

A particular (non-deterministic) algorithm was suggested in [3] to try to infer a set of $2n$ haplotype pairs from n individuals screened as described above. Recently, the algorithm was extensively used in the first large-scale study to infer haplotypes from sequence variation data in humans [2]. Tested on simulated genomic data, the solution (set of haplotype pairs) created by an execution of the algorithm is said to *fail* if it infers a haplotype not used to create the simulated data (this is called

an "anomalous match"), or it terminates without suggesting haplotypes for some of the input genotypes (each such unresolved genotype is called an "orphan"). But the simulation in [3] showed that

> ... in no case was a solution that obtained anomalous matches the solution with the fewest orphans. This empirically demonstrates a parsimony rule that the solution with the fewest orphans is the valid solution and suggests that when a solution resolves all haplotypes, it is likely to be unique [3].

We call this the *maximum-resolution hypothesis*.

Hence the major open algorithmic question from [3] is whether *efficient* rules exist to break choices in the execution of the algorithm, so as to minimize the number of resulting orphans, or (equivalently) maximize the number of resolutions.

In this paper we formalize the above problem, calling it the **Maximum resolution (MR) problem**, and study the problem in detail. We first show that the MR problem is NP-hard, and in fact, Max-SNP complete. The reduction will also show that two variants of the MR problem are NP-hard. We next reformulate the MR problem as a problem on directed graphs, with an exponential time (worst case) reduction. The appeal of the reformulation is the expectation that the reduction will be efficient on typical problem instances, and that the resulting graph problems can be efficiently solved in practice. In fact, we initially hoped that the graph problem could be solved efficiently in a worst-case sense by well-developed combinatorial optimization methods. However, we show that the purely graph-theoretic problem is also NP-hard, so that even if the reduction took no time, this approach to the maximum resolution problem would not avoid an NP-hard problem. Never-the-less, this is the approach we take in practice, formulating an integer linear-programming solution to the graph theoretic problem. We discuss efficient implementation of the reduction from genomic data to the (integer) linear program. 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. Finally, we discuss a use of linear programming to assess how well *any* algorithm based on the maximum-resolution hypothesis can make *correct* resolutions.

3 Formal specification of the Maximum Resolution problem

Abstractly, input to the MR problem 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". We can assume that any ambiguous vector contains at least two ambiguous positions, since a vector with only one ambiguous site has a forced haplotype pair. Given two non-identical resolved vectors R and NR, the *conflation* of R and NR produces the ambiguous 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 paper of Clark [3] 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 inference rule is justified in [3] by arguments from theoretical population genetics, and by simulations, and is also commonly employed in other haplotyping methods used in similar problem settings [20]. Clark proposed an algorithm that applies the inference rule in a sequential, *local* manner, searching for an application of the rule, and immediately applying the rule whenever an application is found. Input to the algorithm must contain some resolved vectors, or vectors with only one ambiguous site (the two haplotypes of the latter vectors are forced).

Note that in such a local approach, 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 [3] 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 [3] show that the local inference method tends to produce anomalous vectors only when the execution also ends with remaining orphans. It is thus conjectured in [3] that when all the vectors can be resolved, the solution is unique and correct. Therefore, the absence of orphans can be used as an indication that the correct haplotypes have been inferred. This leads to the

Maximum Resolution (MR) Problem[2]: 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 local inference method maximizes the number of ambiguous vectors that are resolved? An algorithm to solve this problem needs to take a more *global* view of the data, than does the local inference method, to see how each possible application of the inference rule influences choices later on. Unfortunately, we will first show that the MR problem is NP-hard and Max-SNP-complete.

4 The MR problem is NP-hard

In this section we first show that the MR Problem is NP-hard and Max-SNP complete. The reduction generates problem instances that conform to a very severe restriction asserted to hold in real data [3]. The reduction will also establish that even if a natural but exponential-time operation were allowed for free, the remaining MR Problem (called the MRG problem) remains NP-hard[3].

4.1 The Reduction

We show that the MR problem is NP-hard by reducing the Satisfiability problem to the MR problem. Input to the problem is a boolean formula $\mathcal{F}$ with c clauses

[2]When applied to SNP data, the problem can be called the "Max-SNP Completion problem".

[3]Rob Irving and Paula Bonizzoni each independently found a simpler reduction to show that MR Problem is NP-hard, reducing from the exact 3-cover problem. However, those reductions do not show the problem is Max-SNP complete, and they do not show that the UEMR and UMR problems are NP-hard, i.e., the reductions do not construct problem instances with the unique expression property (these are discussed in the next section). They also do not extend to show that the MRG problem is NP- hard. The last result is of central concern because we propose to solve the MR problem in practice by casting it (in exponential time) as an instance of the MRG problem. Hence we present the more complex reduction in this paper.

containing v variables[4] $X_1, X_2, ..., X_v$. We assume that no clause contains both a literal and its negation since any such clause can be removed from $\mathcal{F}$. We similarly assume that no two clauses contain exactly the same set of literals. The constructed instance of the MR problem is called $V(\mathcal{F})$ and will contain three sets of vectors. The first set, called the *truth set* contains two resolved vectors, denoted T_i and F_i, for each variable X_i in $\mathcal{F}$. The next set, called the *selector set*, contains one ambiguous vector, denoted S_i, for each variable X_i in $\mathcal{F}$. The third set, called the *clause set* contains one ambiguous vector, denoted C_k, for each clause k in $\mathcal{F}$.

4.1.1 High level idea

The high level idea is to construct the vectors of $V(\mathcal{F})$ so that for every i from one to v, either of the (resolved) truth vectors T_i and F_i can be applied to resolve the selector vector S_i, but no other vector can resolve S_i. The vector resolving S_i determines the truth setting for variable X_i in the natural way. Further, the vectors of $V(\mathcal{F})$ have the property that if T_i is applied to resolve S_i, creating resolved vector NR, then NR can be applied to resolve a clause vector C_k if and only if the literal x_i appears in clause k of $\mathcal{F}$. Similarly, if F_i is applied to resolve S_i, creating NR, then NR can be applied to C_k if and only if the literal $\overline{x_i}$ appears in clause k. So a series of applications of the inference rule that resolve all the clause vectors determines a way to make the formula $\mathcal{F}$ true. Conversely, a way to make $\mathcal{F}$ true determines a series of applications that resolve all the clause vectors. Of course, there are many details needed to make the construction work.

4.1.2 Detailed constructions

Each vector in $V(\mathcal{F})$ has $3v+c+1$ positions. We name blocks of positions in order to make the details of the reduction more intuitive. The first v positions are called the

[4] In this reduction, the distinction between a "variable" and a "literal" is crucial. To emphasize the distinction, we write variables in upper case, and literals in lower case. A variable, X_i for example, appears in a clause as a "literal", either unnegated (hence appearing as x_i) or negated (appearing as $\overline{x_i}$). A clause that contains the literal x_i and another clause that contains the literal $\overline{x_i}$ have the variable X_i in common.

"variable identifier" positions; the next $2v$ positions are called the "literal" positions; the next c positions are called the "clause membership" positions; and the last position is called the "clause blocker" position.

The truth set

The truth set contains $2v$ vectors and consists of two resolved vectors, T_i and F_i, for each variable X_i in $\mathcal{F}$. In both vectors T_i and F_i, the first v positions (the variable identifier positions) are set to zero, except for positions i which is set to one. The next $2v$ positions (the literal positions) are set to zero in all vectors of the truth set. The next c positions (the clause membership positions) of T_i and F_i are associated with the c clauses of $\mathcal{F}$, and their values are determined by the clauses that literals x_i and $\overline{x_i}$ appear in. For k from 1 to c, position $3v + k$ is set to one in T_i and to zero in F_i, if x_i appears in clause C_k. (Recall that no clause contains both a literal and its negation.) Symmetrically, position $3v + k$ is set to zero in T_i and to one in F_i, if $\overline{x_i}$ appears in clause C_k. If neither x_i nor $\overline{x_i}$ appear in C_k, then position $3v + k$ is set to *one* in *both* T_i and F_i. Position $3v + c + 1$, the clause blocker position, is set to zero in each vector in the truth set. For an example, see table 4.1.2.

The selector set

The selector set contains v ambiguous vectors S_i, one for each variable X_i in $\mathcal{F}$. The first v positions of S_i are set to zero, except for position i which is set to one. As in the truth vectors, the next $2v$ positions are all set to zero. For every k from 1 to c, position $3v + k$ of S_k is set to one if neither literal x_i nor $\overline{x_i}$ appear in C_k; otherwise it is set to two. The last position (the clause blocker), in every vector in the selector set is set to two. See table 4.1.2.

Lemma 4.1 *For any i, either of the vectors T_i and F_i can be applied to resolve S_i, but neither can be applied to resolve any other selector vector.*

Proof Neither T_i nor F_i can be applied to resolve S_j, for $j \neq i$, because both T_i and F_i have a zero in position j, while vector S_j has a one in that position.

Now we show that either of T_i or F_i can be applied to S_i. In the first $3v$ positions, S_i, T_i and F_i all have zero entries in all positions except for position i, where the three vectors all have a one. Hence none of the first $3v$ positions block the application of either T_i or F_i to S_i. In the next c positions, all values of S_i are either one or two. A position where S_i has value two does not block the application of any resolved vector to S_i, so we need only consider positions in that range where S_i has a one. A position $3v + k$ of S_i is set to one if and only if neither x_i nor $\overline{x_i}$ appears in C_k. Both that is exactly the condition for setting position $3v + k$ of T_i and F_i to one. Hence none of these positions block the application of either T_i or F_i to S_i. The last position in S_i is set to two, again not blocking any application. Since no position in S_i blocks the application of either T_i or F_i, either of these vectors can be applied to resolve S_i. □

Corollary 4.1 *In any series of applications of the inference rule, for every i from one to v, at most one of the vectors T_i or F_i can be applied to resolve any selector vector.*

Corollary 4.2 *By interpreting the application of T_i to S_i as setting variable X_i true, and an application of F_i to S_i as setting variable X_i false, applications of the inference rule that resolve all the selector vectors determine a consistent setting of the variables of $\mathcal{F}$. Conversely, any setting of all the variables of $\mathcal{F}$ determines applications of the inference rule that resolve all the selector vectors of $V(\mathcal{F})$.*

Lemma 4.2 *For any i and $j \neq i$, if the resolved vector NR is created by resolving S_i, then NR cannot be applied to resolve any selector vector S_j. Hence NR can only be applied (if at all) to resolve some clause vectors.*

Proof Vector S_i has a zero in position j, so NR must also. But S_j has a one in that position, so position j blocks the application of NR to S_j. □

The clause set

The clause set contains c ambiguous vectors, $C_1, ..., C_c$, one for each clause in $\mathcal{F}$.

For any k, the first v positions of vector C_k are set to zero, except for any positions i such that either x_i or $\overline{x_i}$ appears in clause k; those positions are set to two. The next $2v$ positions of C_k are set to zero except for any position $v + 2i - 1$ where x_i appears in clause k, and any position $v + 2i$ where $\overline{x_i}$ appears in clause k; those positions are also set to two.

For each r from 1 to c, position $3v + r$ of C_k is set to zero if and only if $r = k$. For $r \neq k$, position $3v + r$ is set to two if and only if clauses k and r contain a variable (not necessarily the same literal) in common; otherwise position $3v + r$ is set to one. For example, the clauses $(x_1 \vee x_2)$ and $(\overline{x_1} \vee x_3)$ contain the *variable* X_1 in common, even though they do not contain any literal in common. The clause blocker position of every clause vector is set to one. See Table 4.1.2.

Lemma 4.3 *No truth vector can be applied to resolve a clause vector.*

Proof The clause blocker is set to one in each truth vector but is set to zero in each clause vector. Hence it blocks the application of any truth vector to resolve any clause vector. □

Lemma 4.4 *Let NR be a resolved vector created by applying the truth vector T_i (respectively F_i) to resolve the vector S_i. Then NR contains a zero in position $3v + k$ if and only if x_i (respectively $\overline{x_i}$) appears in clause k.*

Proof By construction, position $3v+k$ of S_i is set either a one or a two. Hence the only way for *NR* to have a zero in that position is for S_i to have a two in that position, and for the resolving vector T_i (respectively F_i) to have a one in that position. But, by the construction rules for S_i, T_i and F_i, those two conditions can happen if and only if x_i (respectively $\overline{x_i}$) appears in clause k. □

Lemma 4.5 *Let NR be a resolved vector created by applying the truth vector T_i (respectively F_i) to resolve the vector S_i. Then NR can be applied to resolve the clause vector C_k if and only if the literal x_i (respectively $\overline{x_i}$) appears in clause k.*

Proof Suppose *NR* can be applied to C_k. Position $3v + k$ of C_k is set to zero by construction, so that position must be set to zero in *NR* as well. But by Lemma 4.4, that can only happen if the literal x_i (respectively $\overline{x_i}$) appears in clause k.

Now suppose *NR* was created by applying T_i (respectively F_i) to S_i, and that x_i (respectively $\overline{x_i}$) appears in clause k. None of the first v positions of C_k will block the application of *NR* because those positions are set to two, except for any position j where neither x_j nor $\overline{x_j}$ is contained in clause k. Those positions are set to zero in C_k. But they are also set to zero in S_i, T_i and F_i, so they will be set to zero in *NR*. None of the next $2v$ positions of C_k will block application of *NR* because these positions are all set to zero or two, while all those positions are set to zero in S_i, T_i and F_i, and hence are all set to zero in *NR*.

Position $3v + k$ of C_k is set to zero, and since x_i (respectively $\overline{x_i}$) is in clause k, Lemma 4.4 establishes that position $3v + k$ of *NR* will also be set to zero. Hence that position does not block. The positions in C_k set to two cannot block. Only the positions set to one in C_k can block the application of *NR* to C_k. Let $3v + r$ be such a position. By construction, that position in C_k is set to one if and only if $r \neq k$ and clauses r and k have no variables in common. But x_i (respectively $\overline{x_i}$) is contained in clause k, so neither it nor $\overline{x_i}$ is contained in clause r. Therefore, position $3v + r$ is set to one in each of S_i, T_i and F_i, and hence that position is set to one in *NR* as well. So position $3v + r$ does not block the application of *NR* to C_k.

Finally, note that the clause blocker position of *NR* is set to one (since it is set to two in S_i and to zero in T_i and F_i). The clause blocker position of C_k is set to one, hence it does not block the application of *NR* to C_k.

Since no position in C_k blocks the application of *NR* to C_k, *NR* can be applied to resolve C_k. □

Lemma 4.6 *Let NR be a resolved vector created by resolving a clause vector C_k. Then NR cannot be applied to resolve any clause or selector vector.*

Proof There are $2v$ clause membership positions in each vector, and each positions corresponds to exactly one literal in $\mathcal{F}$. In each truth and selector vector, the clause membership positions are set to zero. In each clause vector C_k, the positions are set to zero except for those positions corresponding to literals appearing in clause k; those are set to two. Consequently, the resolved vector *NR* will have a zero in all clause membership positions except for the positions where C_k has a two. In each of those positions, *NR* will have a one. Since each selector vector has only zeros in the clause membership positions, *NR* cannot be applied to any selector vector. Since no two clauses contain exactly the same set of literals (by assumption), for any $j \neq k$, there will be some clause membership position in *NR* that is set to one, but set to zero in C_j. □

Given $\mathcal{F}$, the constructed set of vectors is called $V(\mathcal{F})$.

Theorem 4.1 *There is solution to the MR problem on input $V(\mathcal{F})$ that resolves all the vectors if and only if formula $\mathcal{F}$ can be satisfied. Hence, the MR problem is NP-hard.*

Proof The proof of the "only if" direction has been laid out in the preceding lemmas. That is, a solution to the UMR problem on $V(\mathcal{F})$ that resolves all the vectors determines a way to set the variables of $\mathcal{F}$ to make $\mathcal{F}$ true. The "if" part is simpler and follows from Corollary 4.2 and Lemma 4.5. □

Note that the vectors in the construction are all distinct. This fact will be used in the next section.

Theorem 4.2 *For any k from one to c, there is a way to satisfy k of the clauses of $\mathcal{F}$ if and only if there is a way to resolve k of the clause vectors of $V(\mathcal{F})$. Hence the construction also reduces the MAX-SAT (and MAX-3-SAT) problems to the MR problem.*

Note that all the selector vectors get resolved in any optimal solution to a MR problem, so the reduction from a MAX-3-SAT instance $\mathcal{F}$ creates an instance where the optimal solution is v plus the number of clauses that can be satisfied in $\mathcal{F}$. Note also that in any MAX-3-SAT instance with v variables, at least $v/3$ clauses can be satisfied, so the optimal solution to the MR instance is at most four times the number of clauses that can be satisfied in $\mathcal{F}$. It follows that

Theorem 4.3 *The MR problem is Max-SNP complete*[5]*, hence there is some fixed limit on the approximation accuracy of every polynomial-time approximation method for the MR problem, unless $P = NP$.*

4.2 How realistic is the reduction?

The relevance of NP-hardness for practical problems (where realistic input to the problem is a proper subset of all possible input) can be questioned if the proof constructs problem instances that are very unlike those encountered in practice. It is always conceivable that realistic problem instances have some special features that allow the problem to be solved efficiently on that subset of instances. An NP-completeness reduction that constructs unnatural problem instances still has value, since it establishes that unless an algorithm has an a efficient way to screen out those unnatural instances, that algorithm can not both be correct and efficient, unless P = NP. However, the more one can make the reduction construct natural-like instances the more one reduces the possibility than an efficient, correct algorithm exists for natural input.

The reduction presented for the MR problem can be expanded and modified to largely agree with gross features of realistic data, such as percentage of vectors that are homozygous, or the percentage of sites that are homozygous. A seemingly more severe restriction on the permitted construction comes from the statement in [3] that

[5]Thanks to Antonio Piccolboni for pointing out that the reduction is an L-reduction [16], proving Max-SNP completeness. This result connects the biological SNP to the complexity theory SNP, and should lead to some good puns. Finding them is left as an open problem. For now all we say is that the Max-SNP completion problem is Max-SNP complete.

	T_1	F_1	T_2	F_2	T_3	F_3	T_4	F_4
X_1	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*
X_2	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*0*
X_3	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*
X_4	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*
x_1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
$\overline{x_1}$	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
x_2	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
$\overline{x_2}$	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
x_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
$\overline{x_3}$	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
x_4	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
$\overline{x_4}$	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
C_1	*1*	*1*	*1*	*0*	*1*	*0*	*0*	*1*
C_2	*1*	*0*	*0*	*1*	*1*	*1*	*1*	*0*
C_3	*0*	*1*	*1*	*1*	*0*	*1*	*1*	*1*
CB	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Table 1: The Truth Set of Vectors for the formula
$\mathcal{F} = (x_2 \vee x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_2} \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3})$.
The rows are separated into four blocks: the variable identifier positions, the literal positions, the clause membership positions, and the clause blocker position. Each row, other than the clause blocker row, is associated with a variable, a literal or a clause, written at the left. The name of each vector is written above it.

	X_1	X_2	X_3	X_4
X_1	*1*	*0*	*0*	*0*
X_2	*0*	*1*	*0*	*0*
X_3	*0*	*0*	*1*	*0*
X_4	*0*	*0*	*0*	*1*
x_1	*0*	*0*	*0*	*0*
$\overline{x_1}$	*0*	*0*	*0*	*0*
x_2	*0*	*0*	*0*	*0*
$\overline{x_2}$	*0*	*0*	*0*	*0*
x_3	*0*	*0*	*0*	*0*
$\overline{x_3}$	*0*	*0*	*0*	*0*
x_4	*0*	*0*	*0*	*0*
$\overline{x_4}$	*0*	*0*	*0*	*0*
C_1	*1*	*2*	*2*	*1*
C_2	*2*	*2*	*1*	*1*
C_3	*2*	*1*	*2*	*2*
CB	*2*	*2*	*2*	*2*

Table 2: The Selector Set of Vectors for the formula $\mathcal{F}$. Each vector in this set is named by a variable in $\mathcal{F}$.

	C_1	C_2	C_3
X_1	*0*	*2*	*2*
X_2	*2*	*2*	*0*
X_3	*2*	*0*	*2*
X_4	*2*	*2*	*0*
x_1	*0*	*2*	*2*
$\overline{x_1}$	*0*	*0*	*0*
x_2	*2*	*0*	*0*
$\overline{x_2}$	*0*	*2*	*0*
x_3	*2*	*0*	*0*
$\overline{x_3}$	*0*	*0*	*2*
x_4	*0*	*2*	*0*
$\overline{x_4}$	*2*	*0*	*0*
C_1	*0*	*2*	*2*
C_2	*2*	*0*	*2*
C_3	*2*	*2*	*0*
CB	*1*	*1*	*1*

Table 3: The Clause Set of Vectors for the formula $\mathcal{F}$.

the proposed algorithm "is intended to be able to resolve ambiguous sequences in the situation where there is only one unique pair of true haplotypes that correspond to a given ambiguous sequencing gel[6]."

To formalize that statement, we say that a set of haplotypes (resolved vectors) $\mathcal{R}$ has the *unique expression property* for a set of ambiguous vectors $\mathcal{A}$ if every ambiguous vector in $\mathcal{A}$ can be expressed as the conflation of at most one pair of haplotypes in $\mathcal{R}$. Clark's statement implies that in order for solutions to realistic instances of the MR problem should satisfy the unique expression property, since the set of "true" (but unknown) haplotypes underlying the input genotype data has the unique expression property for that data. Note that this assertion does not say that there is only one solution to the MR problem on realistic data, or trivialize the problem, but rather puts additional conditions on what a realistic solution looks like.

Clearly, not all solutions to the MR problem will have the unique expression property, and there are instances where the best solution to the MR problem resolves all the ambiguous vectors, but there is no solution with the unique expression property that resolves even one ambiguous vector. However, we will show that the reduction given for the MR problem constructs problem instances where *every* feasible solution does have the unique expression property. Hence, the MR problem is NP-hard even when restricted to problem instances where the solution is required to have the unique expression property. This extends the relevance of the NP-hardness result. We begin with the following lemma whose proof is immediate and left to the reader.

Lemma 4.7 *Let R and R' be resolved vectors and let A be an ambiguous vector. A is the conflation of R and R' if and only if R can be applied to A to produce R', and R' can be applied to A to produce R. Further, there is only one resolved vector R' such that the conflation of R and R' produces A.*

[6]The phrase "ambiguous sequencing gel" refers to the specifics of how the genotype data is obtained, and is abstracted to "ambiguous vector" in this paper.

Theorem 4.4 *Let $\mathcal{F}$ be any boolean formula and let $V(\mathcal{F})$ be the instance of the MR problem constructed by the reduction in the previous section. Let $\mathcal{R}$ be the set of resolved vectors, or haplotypes, (including the input resolved vectors) returned by some solution to $V(\mathcal{F})$. Then $\mathcal{R}$ has the unique expression property for the ambiguous vectors in $V(\mathcal{F})$.*

Proof First, consider a selector vector S_i. If S_i is resolved in the solution, let NR denote the resolved vector created by that resolution. (If S_i is not resolved in the solution, then omit NR from the following argument.) Lemmas 4.1, 4.2 and 4.6 establish that in any solution to $V(\mathcal{F})$, only the resolved vectors T_i, F_i or NR can be applied to resolve S_i. Therefore, by Lemma 4.7 there are only three candidate pairs of resolved vectors whose conflation could form S_i: either (T_i, F_i) or (T_i, NR) or (F_i, NR). However, T_i and F_i have a 0 in the clause blocker position, while S_i has a 2 there, so the conflation of T_i and F_i cannot be S_i. Now T_i and F_i are not identical since they must differ in some clause membership position. NR is produced by applying exactly one of the vectors T_i or F_i to S_i, say T_i without loss of generality. Lemma 4.7 then says that S_i is not the conflation of the pair (NR, F_i). Hence, if S_i is resolved in the solution, only the pair (NR, T_i) conflates to form S_i, and the unique expression property holds for all the selector vectors.

Now consider a clause vector C_k, and let (R, R') denote a pair of resolved vectors in the solution whose conflation forms C_k. If C_k is resolved in the solution, let X denote the resolved vector applied to C_k, and let NR denote the inferred resolved vector. Then (X, NR) is one pair whose conflation forms C_k and it remains to prove that this is the only such pair. By Lemmas 4.3, 4.5 and 4.6, the only vectors that can be applied to C_k are the vectors created by resolving a selector vector of a variable contained in C_k (either negated or not). Call that set of resolved vectors RV_k. Hence, either one of R and R' is NR and the other is X, or both R and R' are in RV_k. The first case has already been examined, so we assume the later case and let S denote the selector vector from which R was created. By Lemma 4.7 (since C_k is the conflation

of R and R') R' can be applied to C_k to produce R. Also by Lemma 4.7 (since R was created from S), R can be applied to S. But that violates Lemma 4.6. Hence, the unique expression property holds for all clause vectors. □

To summarize, we define the **unique expression MR (UEMR) problem**: Given a set of resolved and ambiguous vectors, find an execution of Clark's algorithm that maximizes the number of ambiguous vectors resolved, and returns a set of haplotypes with the unique expression property for the given set of ambiguous vectors.

Clark's statement is interpreted as the assertion that on realistic data, a solution to the MR problem should likely solve the UEMR problem as well. However, this offers no escape from NP-hardness, as Theorems 4.1 and 4.4 show that the UEMR problem is is NP-hard and Max-SNP complete.

Note that algorithms based on applying the inference rule have no mechanism to solve the UEMR problem explicitly. Rather they explicitly try to solve the MR problem. However, the desire to solve the UEMR problem motivates the following variant of the MR problem. The **unique MR (UMR)** problem: find an execution of Clark's method that maximizes the number of ambiguous vectors resolved, under the constraint that identical vectors in the input *must* be resolved in the same way, i.e., creating the same haplotype pair.

The UMR problem lies between the MR problem and the UEMR problem, since identical vectors must be resoled in the same way by any solution to the UEMR problem. In practice, the UMR problem reduces to the MR problem after any duplicate vectors are removed, and in this way, any algorithm for the MR problem can be used to solve the UMR problem. The construction used to prove the NP-hardness of the MR problem creates distinct vectors, hence that construction also proves the NP-hardness of the UMR problem.

5 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. In the raw data reported in [3], the percentage of sites that are ambiguous is around 0.5%, and the number given in [2] is even lower. However, once all the sites with no variation in the study population are removed, the percentage of ambiguous sites per individual falls in a range of 20 to 50 percent (the average number of variable sites per individual was seventeen in that study).

We initially hoped that the MRG problem would have an efficient (worst-case) solution, and initially sought one. An efficient solution to the MRG problem would not contradict the NP- hardness of the MR problem, because of the worst-case exponential cost to reduce MR to MRG. (Similarly, the reduction given earlier does not establish that the MRG problem is NP-hard.) Unfortunately, the MRG problem is itself NP-hard, as we will show next. So even if we could construct G for free, the remaining problem is likely to lack an efficient worst-case solution.

Theorem 5.1 *The MRG problem is NP-hard.*

Proof The Satisfiability problem is NP-complete even when each clause is restricted to contain at most three literals, and when each variable appears at most three times in the formula, and (hence) each literal appears at most twice [16]. This is called the bounded 3-SAT problem. Clearly, the reduction presented in Section 4.1

works without any modification to reduce the bounded 3-SAT problem to the MR problem.

We next show that when the formula $\mathcal{F}$ satisfies the conditions of the bounded 3-SAT problem, every vector in the resulting vector set $V(\mathcal{F})$ (created by the reduction) contains at most twelve ambiguous positions. Each selector vector has at most four positions whose value is set to two. One of those is the clause blocker position, and the other three are in the clause membership block. The number is bounded by three since no variable is contained in more than three clauses. Similarly, each clause vector C_k has at most three positions in the variable identifier block and three in the literal block, whose value is two, because each clause contains at most three literals. There are at most six positions in the clause membership block of a clause vector C_k, that are set to two. This follows because clause k contains at most three variables and no variable appears in more than three clauses, one of which must be clause k. So for any variable in clause k, there are at most two other clauses that contain that variable (either negated or unnegated). Hence every vector in $V(F)$ contains at most a constant number of ambiguous positions (twelve at most[7]), independent of the number of number of variables or the number of clauses in $\mathcal{F}$. In this case, the number of nodes in G at most a constant multiple of the number of vectors in $V(\mathcal{F})$. Essentially, we have detailed a polynomial-time reduction of the bounded 3-SAT problem to the MRG problem, proving that the MRG problem is NP-hard. □

6 Solving the MRG problem via Mathematical Programming

Despite the potential exponential blowup, we approach the MR problem in practice by reducing it to the MRG problem. The reduction is more efficient than suggested in the prior description, because we do not first enumerate all the nodes in G, but only those nodes that are reachable from some node in I. Let $\overline{G}$ denote graph derived from

[7]This number can be reduced to ten at the cost of increasing the length of the vectors.

G where every node that can't be reached from I has been removed, along with any incident edges. A naive implementation of the reduction would first generate all the nodes, then consider all nodes pairs to construct the edges, and then remove any nodes not reachable from I. For simplicity of analysis, imagine that each of n ambiguous vectors have h ambiguous positions. The naive approach would generate $n2^h$ nodes and then consider $n^2 2^{2h}$ potential edges, using $O(m)$ time on each edge. Instead, our implementation constructs $\overline{G}$ in time proportional to mn^2 plus the number of edges in $\overline{G}$. This should be vastly smaller in general than the number of potential edges examined in the naive implementation, and our empirical tests confirm this.

The MRG problem can be solved (in principal) by formulating it as a problem in mathematical programming. We propose two such formulations. The first one exactly captures the MRG problem but involves non-linear constraints, and hence may be harder to solve in practice. The second formulation is based on linear (integer) programming and is not exact, and hence can fail in theory to find the optimal solution. However in our tests, the problem that can occur in theory has rarely been observed. Hence that is the approach we have concentrated on for practical application.

6.1 An exact formulation

The MRG problem can be formulated by adding simple non-linear constraints to a network flow formulation. First, let all the edges in $\overline{G}$ be given infinite capacity. 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 [12, 17]) 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. This formulation also has a relationship to polymatriod flows that may be useful on simpler problem instances. We believe it is conceptually helpful to have an exact formulation, but we have not attempted to solve practical instances of the problem with available software for solving non-linear programming problems. Instead, we have focussed on the following formulation.

6.2 A linear (integer) programming approach to MRG

In practice, we solve the MRG problem by casting it as an integer, linear programming problem. We create one binary (integer linear programming) variable y_v for each node v in $\overline{G}$. We create inequalities so that the variables set to one correspond to nodes that are reached in a solution of the MRG problem, hence correspond to haplotypes that are inferred by some successive application of the inference rule. In detail, for each set of nodes $N(A)$, create the inequality

$$\sum_{v \in N(A)} y_v \leq 1.$$

These inequalities assure that at most one node in $N(A)$ will be reached, hence vector A will be resolved in at most one way. For each node w not in I, let $P(w)$ be the set of nodes with an edge directed into w. That is, a node v is in $P(w)$ if and only if there is a directed edge in G from v to w. Then for each node $w \notin I$, create the inequality

$$y_w \leq \sum_{v \in P(w)} y_v.$$

These are called "predecessor" constraints. These constraints assure that a node w not in I cannot be reached without reaching at least one node that can act as a predecessor of w, i.e., can lead directly to w. There is no such constraint for any node in I, hence the predecessor constraints *try* to assure that if y_w is set to one, then there

is a path from I to w containing nodes whose corresponding variables are all set to one. Finally, the objective function is

$$Maximize \sum_v y_v.$$

It is simple to see that the optimal solution to the MRG problem defines a feasible solution to this integer linear program. Conversely, if $\overline{G}$ is *acyclic* (contains no directed cycles), then an optimal solution to the integer linear program gives an optimal solution to the MRG problem. However, when $\overline{G}$ does contain a directed cycle, the integer program is not guaranteed to solve the MRG problem.

The problem is that an optimal solution to the integer linear program may set a variable y_w to one, but not set to one all the variables corresponding to nodes on some path from I to w. This can happen in theory when the variables corresponding to a *directed cycle* of nodes in $\overline{G}$ are all set to one, but in every set of variables corresponding to a path from I to the cycle, at least one variable is set to zero. This has been observed to happen in simulated data, but very rarely, and is more unlikely than may be initially thought. First, in the data we have tested, the graphs are often acyclic. Second, variables in a directed cycle in $\overline{G}$ cannot all be set to one in the integer linear program unless every node in that cycle is in a different set $N(A)$. Finally, even if all variables in a directed cycle in $\overline{G}$ are set to one, the integer programming solution solves the MRG problem as long as each of these variables is also on a path from I where all variables on the path are set to one. That condition is easy to check, as follows. From the set of variables set to one in the solution, create the induced subgraph of $\overline{G}$ consisting of I and all nodes whose variables are set to one. Then, search for a directed, spanning forest in this subgraph, where each tree in the forest is rooted at a node in I. Since the integer linear program sets to one at most one variable corresponding to nodes in any set $N(A)$, any directed spanning forest of this subgraph is an optimal solution to the MRG problem. Searching for such a forest is trivial, and this is the approach we have built into the our program.

Note that in the integer programming formulation, there are no variables for

edges in $\overline{G}$. This is done for efficiency sake, reducing the number of variables in the integer program. Hence if the integer programming solution does not solve the MRG problem, because no proper spanning forest is found, it is not possible to add a new inequality that forbids the "offending cycle". In fact, the best cutting plane to add may be one that forbids the use of a node that is off the offending cycle. Hence if cycles are problematic, the present integer programming formulation cannot be easily extended to handle the problem. In contrast, an integer programming formulation of the MRB problem that had explicit variables for edges would be able to employ a cutting-plane approach, if cycles proved to be a problem, but would be a much larger integer program. Fortunately, in our simulations, the problem of cycles has not been significant.

Finally, there is the issue of finding an optimal solution to the integer linear program. Integer linear programming is an NP-hard problem, but linear programming, where the variables are allowed to take on any values between zero and one, can be solved very efficiently in both theory and practice. Hence, the approach we have taken is to first relax the integrality constraints and only restrict each variable to be in the range $[0, 1]$. The optimal solution to this linear programming relaxation is guaranteed to have a value that is greater or equal to the optimal value of the integer program, so when the linear program has an optimal integer solution, it is optimal for the integer program as well. We have written programs (in C) to efficiently create the graph $\overline{G}$ and then to create and optimize the linear program, given a set of input vectors. A full example is given in [8].

7 Experimental Results

Experiments were mainly conducted to address two questions. First, does the linear, integer programming methodology work in practice to efficiently solve the MRG problem, or do the cycle and non-integrality problems discussed above make the approach ineffective? Second, do the solutions improve on the number of resolutions obtained

by the local method? We examined these questions under two sets of data: simulated genotype data generated without following a population model, and data generated following a population model.

Our first experiments use randomly generated genotype data of up to 80 vectors, 20 initially resolved and 60 ambiguous, each containing 15 sites. In an ambiguous vector, each site is chosen to be ambiguous with probability one-half. The resulting graphs are up to 300 nodes, but are quite sparse. The optimal linear program executes in a few seconds, and almost always returns an integer optimal. Why integrality is so often the case is an open question. In the rare case that a fractional solution was returned, an integer optimal of equal value, or one less, could usually be obtained by fixing a single variable to 1 or 0. Alternatively, we have used the commercial software package CPLEX, which very efficiently (faster than our own software) solves the integer program optimally. As for the potential cycle problem, in hundreds of trials with randomly generated data, we have never observed a case where the integer program solution did not also solve the MRG problem. For instances where a large graph is generated, the local inference method (reordering the data between each execution) needs to be executed 1,000 to 10,000 times in order to have a reasonable chance of finding the optimal solution once in those executions. In those instances, the integer linear programming solution typically resolves 10% to 70% more ambiguous vectors than does the average of the executions of local inference method, and runs significantly faster than 1,000 executions of that method (using our implementation). These experiments test the methods and programs under extreme conditions that create more complex (although still sparse) graphs than one would expect in more realistic data.

To test the programs under more realistic genotype data, we ran a widely-used program developed by R. Hudson [11] that uses coalescent theory to generate a simulated population of haplotypes. Haplotypes generated by the program were then randomly paired to create genotype data used as input to the LP and local infer-

ence methods. There are many possible choices of parameter settings, and we picked settings to generate data we believe is consistent with the data in [3]. The graphs generated under these conditions are much simpler than in the case of data generated without a population model.

A typical choice of parameter settings generated data consisting of twenty individuals with 60 sites, averaging 13.5 hetrozygous sites per individual. The resulting graphs have between 35 and 60 nodes and an average degree (number of incident edges per node) between 2 and 3. In these simulations, the LP method (without explicit integrality constraints) always produced an integer optimal solution in one second or less. However, we have observed a few cases where the solution contains a directed cycle, and hence does not solve the MRG problem. In about 20% of the datasets produced by the Hudson program, there are no homozygous vectors or single-site hetrozygous vectors, and in these cases, neither the LP nor the local inference method can be used. In about 10% of the cases, the LP method obtained one more resolution than the best of 1,000 executions of the local inference method. In about 60% of the cases, both the LP method, and the best of 1,000 executions of the local inference method resolved all but one of the input ambiguous vectors. In these cases, the LP method did not outperform the local inference method, but is still useful because it proved, for each such case, that no execution of the local inference method could resolve all the ambiguous vectors. In some cases, the maximum number of resolutions were found by the local inference method in less than 1% of the 1,000 executions. Hence, without use of the LP method, one might still wonder if some additional execution of the local inference method would resolve all the ambiguous vectors. In about 10% of the cases, the local inference method resolved all the ambiguous vectors. In those cases the LP method also resolved all the ambiguous vectors, i.e., gave a non-cyclic, integer solution.

7.1 Checking the maximum-resolution hypotheses

The maximum-resolution hypothesis in [3] is that the most accurate solutions tend to come from the executions of the local inference method that resolve the most ambiguous vectors. The strongest form of this hypothesis is that when an execution resolves all the ambiguous vectors, it is likely to be the only way that all the ambiguous vectors can be resolved by an execution of the local inference method. The logic behind these hypotheses is closely tied to population genetic models and population dynamics, and therefore a careful test of these hypotheses requires the ability to generate data that are faithful to those genetic models and dynamics. That is beyond the scope of this paper, and will be the subject of further (collaborative) work. However, we can make some initial observations from the simulations we have done. For data produced by Hudson's program, we compared the haplotype pairs produced by the LP method and executions of the local inference method to the haplotype pairs used to generate the genotypes. We call a resolution (or haplotype pair) for a ambiguous vector "correct" if it is the same as the original haplotype pair for that ambiguous vector.

Our first observation is a verification of 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. We describe one typical dataset consisting of 20 individuals with 60 sites, and an average of 12.6 ambiguous sites per individual[8]. In that dataset there were 18 ambiguous vectors with more than one hetrozygous site, one homozygous vector, and one vector with only a single hetrozygous site. We ran 10,000 executions of the local inference method on that data. In these executions, 5170 executions resolved 14 of the 18 ambiguous vectors; the number of correct resolutions

[8]The actual parameters for the Hudson program are: nsam: 40, nsites: 1000, segsites: 60, 4Nc = r: 0.5.

ranged between 2 and 7, with less than 1% of those executions correctly resolving 7 vectors. At the other extreme, there were 776 executions that resolved 17 of the 18 ambiguous vectors; the number of correct resolutions ranged between 5 and 11, and almost 6% of those executions resolved 10 or 11 vectors correctly; less than one percent of those executions resolved 11 correctly. In the executions that resolved 15 and 16 vectors, the range of correct resolutions was 3 to 9 and 4 to 9 respectively, with 3.1% and 2.7% of the executions obtaining 9 correct resolutions respectively. The LP solution resolved 17 ambiguous vectors and correctly resolved 9 vectors.

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. In the above example, of the 776 executions that resolved 17 vectors, there were 6 executions that correctly resolved 11, but also 52 that resolved only 5, with the median falling between 7 and 8, and the average of 7.4. In general, the number of correct resolutions returned by the LP method appears to follow the distribution of the number of correct resolutions in the executions of the local inference method that resolve the maximum number of ambiguous 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. This is briefly discussed later.

To further build confidence in the local inference method, and hence in the basic maximum-resolution hypothesis, we made the following computations. For any given set of haplotype pairs produced by Hudson's program, we modified each pair by randomly deciding to flip, or not, the 0 and 1 value at each site where the pair differs. This creates a new, randomized, set of haplotype pairs, that generate exactly

the same genotype vectors generated from the original pairs. These randomized haplotype pairs are then used to score each execution of the local inference method. That is, we note how many times the method produces the associated randomized pair, when it resolves an ambiguous vector.

If the local inference method, which implicitly reflects a population genetic model, is consistent with the explicit model underlying Hudson's data generation program, then the local inference method should preferentially return original haplotype pairs from the Hudson program, over randomized haplotype pairs. This behavior was dramatically demonstrated in the simulations we did. In the example discussed above, out of 10,000 executions of the local inference method, none produced pairs that agreed with more than three of the randomized haplotype pairs. Of the executions that resolved 17 ambiguous vectors, roughly 15%, 60%, and 25% produced pairs that agreed with one, two and three randomized pairs respectively. Compared to the results for the original haplotype pairs, this is very strong evidence for the consistency of the local inference method with data of the type generated by Hudson's program. This implies that the local inference method should also be consistent with real population data to the extent that the real data is consistent with the coalescent model underlying Hudson's program.

As concerns the stronger version of the maximum-resolution hypothesis, the simulations do not support this hypothesis. It is not uncommon to have datasets where all ambiguous vectors are resolved in every execution of the local inference method, but with a large number of distinct solutions. However, it does seem to be true that in datasets where only a small fraction of the 1,000 executions resolve all the ambiguous vectors, each of those executions produce the same haplotype pairs, and that solution maximizes the number of correct resolutions.

Secondary 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.

In the above example, among all the solutions that resolved 17 ambiguous vectors, there were solutions that used 17, 18 and 19 distinct haplotypes. The average number of correct resolutions for solutions using 17 haplotypes was 9.09, with a variance of 0.77. The average correct for solutions using 18 haplotypes was 7.75 and the variance was 0.92. Finally, the average correct for solutions using 19 haplotypes was 6.60, with a variance of 1.05. The average correct for all solutions resolving 17 vectors is 7.4.

If we do not restrict attention first to the solutions that maximize the number of ambiguous vectors, there is no clear correlation between the number of correct resolutions, and the number of distinct haplotypes used.

Finding effective secondary criteria to help identify the better solutions among the executions of the local method, and possible executions of the LP method, is an ongoing topic of research. Some results, based on examination of real data, will be presented in [15]. It is also possible to modify the LP approach so that it efficiently finds a solution that maximizes the number of resolutions, and within that group, minimizes the number of distinct haplotypes used. Other efficient integer programming approaches are possible that find assignments of haplotype pairs (without regard to the local inference rule) that explain all the ambiguous vectors, and minimize the total number of distinct haplotypes used[9]. Details of these new IP methods, and their performance on simulated data will be presented in a future paper.

[9]This "parsimony" criteria was suggested first by Earl Hubbell, who also proved that the problem of finding such solutions is NP-hard [10].

We note a recent publication [21] which interprets Clark's local inference method as a parsimony method, i.e. one that tries to minimize the number of distinct haplotypes used to resolve the ambiguous vectors. Our simulations are not consistent with that interpretation – the executions of Clark's method return solutions that vary widely in the number of distinct haplotypes used. In the above example, out of the 10,000 executions, 944 use 14 distinct haplotypes, 4454 use 15, 1342 use 16, 2632 use 17, 365 use 18, and 263 use 19. Thus, to minimize the number of distinct haplotypes used, while maximizing the number of resolutions obtained by applications of the inference rule, requires an approach different from the local inference method, as we will detail in a future paper.

7.2 How many *correct* resolutions can *any* algorithm make by using the local inference rule?

The LP method described so far is designed to find, on a given dataset, the maximum number of resolutions that can be made by any algorithm that successively applies Clark's inference rule. But when simulations start with haplotype data (or true haplotype data is known from laboratory work), the LP method can be modified to find the maximum number of *correct* resolutions that could be made by any algorithm that uses the inference rule. We call the problem of computing that maximum number the **MC** problem.

On a given dataset, we solve the MC problem by first finding which, if any, of the LP variables correspond to an original haplotype for the associated ambiguous vector. Then we include only those particular variables in the LP objective function. All the LP inequalities are the same as in the LP for the MRG problem. The optimal solution found by this modified LP gives an upper bound on the solution to the MC problem, and if that solution is integral and cycle-free, then it actually identifies resolutions that achieve that maximum value.

The ability to solve the MC problem in practice will allow us to determine empiri-

cally the ultimate utility of algorithms based on repeated application of the inference rule. Maybe there is some criteria other than the maximum-resolution criteria that could guide an a future algorithm (based on the inference rule) to make more correct resolutions. By solving the MC problem, we can empirically determine how many correct resolutions could ever be made by any such an algorithm, even before such an algorithm has been developed. For example, in the dataset discussed above, where 6 out of 10,000 executions of the local inference method correctly resolved 11 ambiguous vectors out of 17 total resolutions, the solution to the MC problem is 11. So no algorithm that uses the inference rule, no matter what criteria is guiding it, could ever make more than 11 correct resolutions on that dataset. Without this knowledge, since only 6 out of 10,000 executions found 11 correct resolutions, one might be tempted to propose that better solutions might be found by running the local inference method many additional times.

The ability to solve the MC problem in practice will be of greatest value in (future) simulations that generate the most realistic population data. Hence, to date, we have only tested this idea in order to establish that the linear program (without integrality constraints) for the MC problem does generally produce cycle-free integer solutions, and therefore the approach is practical. We can however report that in our simulations to date, the maximum number of correct resolutions that are possible using the inference rule tends to be about 60% to 80% of the maximum possible number of resolutions. That is, the optimal solution to the MC problem tends to be about 60 to 80% of the optimal solution to the MRG problem.

8 Open problems and future work

We mention two related open complexity problems. It is useful to focus on just one ambiguous vector given as input, and ask if there is a way to resolve it via a series of applications of the inference rule starting at some resolved vector. Can that problem be solved efficiently? If so, we would repeat the computation for each of the

ambiguous vectors separately, removing any vector that can't be resolved and getting a quick reflection of the quality of the data. A necessary condition for good data is that a large percentage of the vectors are resolvable in isolation.

Not knowing how to solve the above problem, it again seems desirable, and feasible, to expand each vector and construct the graph $\overline{G}$. Given a node $v \notin I$, one seeks a path from some node in I to v that traverses at most one node in each node set $N(A)$. Can this path problem on $\overline{G}$ be solved efficiently? The following related problem is known to be NP-complete [6]: Given an acyclic, directed graph, a collection of pairs of nodes, and two specified nodes s and t, is there a directed path from s to t that traverses at most one node in each of the given pairs? The node pairs can be made to be disjoint [5] as noted in [7] (problem GT54), and the sets can be expanded to be more than pairs. But the graphs used in the reduction are less restricted than the class of graphs coming from problem MR, so the problem remains open for that restricted class of graphs.

As mentioned earlier, future empirical work will focus on using the most realistic population data, on using more than the maximum-resolution criteria to identify those executions likely to have the most correct resolutions, and on ways to incorporate those new criteria into the LP method. The key to this will be the ability to solve the MC problem in practice. By examining the solutions to the MC problem on that data, we may be able to develop new criteria for guiding algorithms based on Clark's inference rule.

9 Acknowledgements

I would like to thank Chuck Langley for introducing me to the paper of Clark [3] and discussing its present significance. Thanks to Alejandro Schaffer at NCBI for providing additional perspective and citations. Thanks to Jens Stoye, Christian Pedersen, Robert Irving, Paula Bonizzoni and Antonio Piccolboni for helpful conversations on the algorithmic work, and to Hal Gabow for conversations on the paper [6].

References

[1] A. Chakravarti. It's raining SNP's, hallelujah? *Nature Genetics*, 19:216–217, 1998.

[2] 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.

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

[4] 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.

[5] H. Gabow. U. Colorado Dept. of Computer Science. Personal Communication.

[6] H. N. Gabow, S. N. Maheshwari, and L.J. Osterweil. On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, 2:227–231, 1976.

[7] M. Garey and D. Johnson. *Computers and intractability.* Freeman, San Francisco, 1979.

[8] 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.

[9] M. Hagmann. A good SNP may be hard to find. *Science*, 285:21–22, 1999.

[10] E. Hubbel. Personal Communication.

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

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

[13] E. Marshall. Drug firms to create public database of genetic mutations. News item on genomics in Science, volume 284, April 1999, pages 406-407.

[14] D. Nickerson, S. Taylor, K. Weiss, and A. Clark et. al. DNA sequence diversity in a 9.7-kb region of the human lipoprotein lipase gene. *Nature Genetics*, 19:233–240, 1998.

[15] 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.

[16] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

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

[18] N. Risch and K. Merikangas. The future of genetic studies of complex human diseases. *Science*, 275:1516–1517, 1996.

[19] E. Russo and P. Smaglik. Single nucleotide polymorphisms: Big Pharma hedges its bets. *The Scientist*, 13, 1999.

[20] A. Schaffer. NCBI, NIH, 1999. Personal Communication.

[21] 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.

[22] J. Terwilliger and K. Weiss. Linkage disequilibrium mapping of complex disease: Fantasy or reality? *Current Opinion in Biotechnology*, pages 578–594, 1998.