Integer Programming Formulations and Computations solving Phylogenetic and Population Genetic Problems with Missing or Genotypic Data

Dan Gusfield[1], Yelena Frid[1], and Dan Brown[2]

[1] Department of Computer Science, University of California, Davis
gusfield@cs.ucdavis.edu
[2] David R. Cheriton School of Computer Science, University of Waterloo, Canada
browndg@cs.uwaterloo.ca

Abstract. Several central and well-known combinatorial problems in phylogenetics and population genetics have efficient, elegant solutions when the input is complete or consists of haplotype data, but lack efficient solutions when input is either incomplete, consists of genotype data, or is for problems generalized from decision questions to optimization questions. Unfortunately, in biological applications, these harder problems arise very often. Previous research has shown that integer-linear programming can sometimes be used to solve hard problems in practice on a range of data that is realistic for current biological applications. Here, we describe a set of related integer linear programming (ILP) formulations for several additional problems, most of which are known to be NP-hard. These ILP formulations address either the issue of missing data, or solve *Haplotype Inference Problems* with objective functions that model more complex biological phenomena than previous formulations. These ILP formulations solve efficiently on data whose composition reflects a range of data of current biological interest. We also assess the biological quality of the ILP solutions: some of the problems, although not all, solve with excellent quality. These results give a practical way to solve instances of some central, hard biological problems, and give practical ways to assess how well certain natural objective functions reflect complex biological phenomena. Perl code to generate the ILPs (for input to CPLEX) is on the web at wwwcsif.cs.ucdavis.edu/~gusfield

1 Introduction

Several well-studied problems in computational phylogenetics and population genetics have efficient, elegant solutions when the data is "simple" or "ideal", but lack efficient solutions when the data is more complex, due to recombination, homoplasy, missing entries, site incompatibility, or the need to use genotypic rather than haplotypic data. Similarly, optimization variants of many decision problems are often more difficult to solve. In this paper we discuss integer linear programming (ILP) formulations for several such problems and report on empirical investigations done with these formulations. For most of the problems, the concept of incompatibility is fundamental.

Definition: Given a matrix M whose entries are 0's and 1's, two sites (or columns) p and q in M are said to be *incompatible* if and only if there are four rows in M where columns p and q contain all four of the ordered pairs 0,1; 1,0; 1,1; and 0,0. The test for the existence of all four pairs is called the "four-gamete test" in population genetics.

The concept of incompatibility is central to many questions concerning phylogenetic trees and population histories [14, 25, 7] for the following reason. Considering each row of M as a binary sequence, the classic Perfect Phylogeny Theorem says that there is a rooted phylogenetic tree that derives the sequences in M, starting from some unspecified root sequence and using only one mutation per site, if and only if *no* pair of sites of M are incompatible. Moreover, if the root sequence is specified, then the phylogenetic tree is *unique*. For expositions of this classic result, see [9, 25]. The assumption of one mutation per site is called the "infinite sites" assumption in population genetics.

2 Missing Data Problems

When there are no missing values in M, the decision question of whether there are incompatible pairs of sites can be answered in linear time [8], and of course the number of incompatible pairs can trivially be computed in polynomial time. However, the situation is more interesting and realistic when some entries of M are missing, and this leads to three natural, biologically motivated problems, which we call M1, S1, and R1.

2.1 Imputing values to minimize incompatibility

Problem M1: Given a binary matrix M with some entries missing, fill in (impute) the missing values so as to *minimize* the number of incompatible pairs of sites in the resulting matrix M'.

Problem M1 generalizes the following decision question: Can the missing values be imputed so that the the sequences in M' can be generated on a perfect phylogeny? That decision question has an efficient, elegant solution [21] when a required root sequence of the unknown phylogeny is specified as input, but is NP-complete when the root sequence is not specified [27]. When the missing values can be imputed so that M' has no incompatible site pairs, the sequences in M are consistent with the hypothesis that they originated from a perfect phylogeny; when the values cannot be imputed with zero incompatibilities, the solution to Problem M1 gives a measure of the deviation of M from the Perfect Phylogeny model.

The ILP formulation for Problem M1 Our ILP formulation for Problem M1 is direct and simple. Its importance is that it generally solves very quickly,

M	p	q
1	0	0
2	?	1
3	1	0
4	?	?
5	?	0
6	0	?

Table 1. Six rows and two columns in the input M to Problem M1.

imputing missing values with high accuracy, and that it can be built upon to address more complex problems. More generally, these formulations and computations illustrate that for applied problems whose range of data is known, the fact that a problem is NP-hard does not necessarily imply that exact solutions cannot be efficiently obtained for that data.

The ILP for problem M1 has one binary variable $Y(i,j)$ for each cell (i,j) in M that is missing a value; the value given to $Y(i,j)$ is then the imputed value for $M(i,j)$. The program that creates the ILP for problem M1 identifies all pairs of columns (p,q) of M that are not necessarily incompatible, but can be made incompatible depending on the imputed values are set. We let P be the set of such pairs of columns; for each pair (p,q) in P, the program creates a variable $C(p,q)$ in the formulation, which will be forced to 1 whenever the imputations cause an incompatibility between columns p and q. For each pair in P, the program also determines which of the four binary combinations are not presently found in column pair (p,q); let $d(p,q)$ represent those missing (deficient) binary combinations. The program creates a binary variable $B(p,q,a,b)$ for every ordered binary combination a,b in $d(p,q)$; $B(p,q,a,b)$ will be forced to 1 if the combination a,b has been created (through the setting of the Y variables) in *some* row in columns (p,q). The program next creates inequalities that set a binary variable $C(p,q)$ to 1 if $B(p,q,a,b)$ has been set to 1 for *every* combination a,b in $d(p,q)$. Therefore, $C(p,q)$ is set to 1 if (but not only if) the imputations of the missing values in columns (p,q) cause those sites to be incompatible. To explain the formulation in detail, consider the pair of columns shown in Table 1, where a missing entry is denoted by a "?". Then $d(p,q)$ is $\{(0,1),(1,1)\}$.

For each pair a,b in $d(p,q)$, the program will make one inequality involving $B(p,q,a,b)$ for each row r where the pair a,b can be created in columns p,q. The specific inequality for a pair a,b and a row r depends on the specific pair a,b and whether there is a fixed value in row r in sites p or q. The full details are simple and omitted but explained for variable $B(p,q,1,1)$ using the above example. Those inequalities are:

$$Y(2,p) \leq B(p,q,1,1) \tag{1}$$

$$Y(4,p) + Y(4,q) - B(p,q,1,1) \leq 1, \tag{2}$$

which force the variable $B(p,q,1,1)$ to 1 when the missing value in the second row is set to 1 or the two missing values in the fourth row are set to 1; this is the general pattern for this combination.

In the above example, the inequalities to set variable $B(p, q, 0, 1)$ are:

$$Y(2,p) + B(p,q,0,1) \geq 1 \tag{3}$$
$$Y(4,q) - Y(4,p) - B(p,q,0,1) \leq 0 \tag{4}$$
$$Y(6,q) - B(p,q,0,l) \leq 0 \tag{5}$$

The ILP for the example has the following inequality to set the value of variable $C(p, q)$ to one, *if* all the combinations in $d(p, q)$ have been created in the column pair (p, q):

$$C(p,q) \geq B(p,q,1,1) + B(p,q,0,1) - 1$$

In general, the constant on the right-hand side of the inequality is one less than the number of pairs in $d(p, q)$. These inequalities assure that $C(p, q)$ will be forced to 1 if (but not only if) the missing values in columns (p, q) are imputed (by the setting of the Y variables) in a way that makes site pair (p, q) incompatible.

The overall objective function for the ILP is therefore to Minimize $[|F| + \sum_{(p,q)\in P} C(p, q)]$, where F is the set of pairs of incompatibilities forced by the initial 0 and 1 entries in matrix M. We include $|F|$ in the objective for continuity in a later section.

Because the objective function calls for minimizing, we do not need inequalities to assure that $C(p, q)$ will be set to 1 *only if* the missing values in columns (p, q) are imputed in a way that makes site pair (p, q) incompatible. However, such inequalities are possible, and would be added to (or used in place of) the above inequalities if we want to solve the the problem of imputing missing values in order to *maximize* the resulting number of incompatible pairs. Details are omitted for lack of space. The solution to the maximization problem, along with the solution to Problem M1, bracket the number of incompatible pairs in the true data from which M was derived.

If M is an n by m matrix, the above ILP formulation for problem M1 creates at most nm Y variables, $2m^2$ B variables, $\frac{m^2}{2}$ C variables, and $O(nm^2)$ inequalities, although all of these estimates are worst case and the numbers are typically much smaller. For example, if I is the expected percentage of missing entries (which is as low as 3% in many applications), then the expected number of Y variables is nmI. The formulation as described can create redundant inequalities, but we have left them in our description for conceptual clarity, and in practice we have found that the preprocessor in CPLEX removes such redundancies as effectively as any of our more refined programs do. There are additional practical reductions that are possible that we cannot discuss here due to limited space.

Empirical results for Problem M1 We extensively tested the ILPs for Problem M1 to answer two questions: 1) How quickly are the ILPs for problem M1 solved when problem instances are generated using data from a population genetic process; 2) When parts of the data are removed, and the missing data imputed by the ILP solution, how accurately do those imputations reconstruct the original values? In phylogenetic applications, missing data rates of up to

30% are common, but in population genetic applications, rates from one to five percent are more the norm.

We used the program *ms* created by Richard Hudson [15] to generate the binary sequences. That program is the widely-used standard for generating sequences that reflect the population genetic coalescent model of binary sequence evolution. The program allows one to control the level of recombination (defined in Section 2.3) through a parameter r, and a modified version of the program provided by Yun Song, allows one to control the level of *homoplasy* (recurrent or back mutations violating the infinite sites assumption). After a complete dataset was generated, each value in the data was chosen for removal with probability p, varied in the study; we use $I = p \times 100$ to denote the expected percent of missing values. All computations were done on a 1.5 ghz Intel itanium, and the ILPs were solved using CPLEX 9.1. The time needed to generate the ILPs was minimal and only the time used to solve the ILP is reported. We tested our ILP solution to Problem M1 on all combinations of n (# rows) = {30, 60, 90, 120, 150}; m (# columns) = {30, 60, 90}; $r = \{0, 4, 16, 30\}$; and I (expected percent missing values) = {5%, 10%, 15%, 20%, 25%, 30%}[3]. We generated and tested fifty datasets for each parameter combination.

The running times were slightly more influenced by m than by n, and mildly influenced by r, but were mostly a function of $n \times m$ and I (increasing with $n \times m$ and I, and decreasing with r). The largest average execution time (averaged over all 50 datasets) occurred when n, m and I, were at their maximum values and r was zero, but that average time was only 3.98 seconds. Figure 1 (a) shows average running times as a function of I. Hence, despite being NP-hard, problem M1 can be solved very efficiently on a wide range of data whose parameters reflect datasets of current interest in phylogenetics and population genetics.

The imputation accuracy was also excellent, when the product $n \times m$ is large. Error is the percentage of missing values that were incorrectly imputed. The case of 5% missing data illustrates this. When $n = m = 30$, and $r = 4$, the missing values were imputed with an average error of 6.4%, but when $n = 120$, $m = 90$ the average error dropped to 1.8%. In general, error increased with increasing r, and fell with increasing $n \times m$. The likely reason for such good results when $n \times m$ is large is the high level of redundancy in the data, and that these redundancies are exploited when the objective is to minimize incompatibilities. Figure 1 (b) shows error rates as a function of $n \times m$.

2.2 Imputing values to minimize site-removals

In this section we discuss another natural objective function related to Problem $M1$. We first define the **Site-Removal** problem on *complete* data: Given a binary matrix M with no missing entries but some pairs of incompatible sites, find the *smallest* set of sites to remove from M so that no remaining pair of sites is incompatible.

[3] We had earlier studied the case of $I = 1\%$ using a slower ILP formulation, but almost all of the computations took zero recorded time (to three decimal places) on that data, and so we started with a higher percentage of missing data for this study.

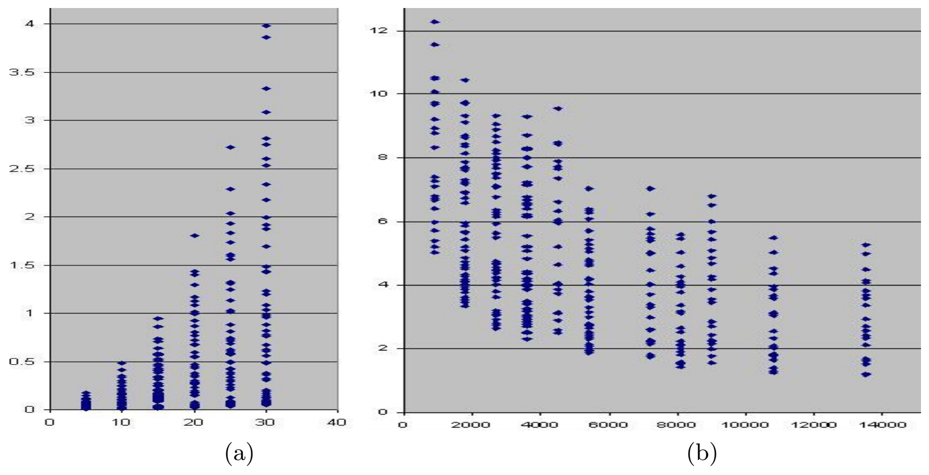


Fig. 1. (a) Average time (in seconds) needed to solve the ILP for Problem M1 as a function of I, the expected percentage of missing data. Each diamond is the average of 50 datasets for a particular combination of the parameters n, m, r and I. (b) Average imputation error rates (percentages) in the solution of the ILP for Problem M1, as a function of $n \times m$. Each diamond is the average of 50 datasets for a particular combination of the parameters n, m, r and I.

Finding a small(est) set of sites to remove so that no remaining pairs are incompatible is often suggested and employed (particularly in phylogenetic studies) as a means to clean up data that does not perfectly conform to the Perfect Phylogeny model. The expectation is that while there may be some sites where homoplasy is observed (due to recurrent or back mutations at that site), a Perfect Phylogeny constructed from the remaining sites will give valid evolutionary information about the taxa. There are similar scenarios in population genetics. Of course, in order to get the most informative phylogeny, we want to remove as few sites as possible, motivating the Site-Removal problem. The Site-Removal Problem is NP-hard and is typically formulated as a Node-Cover problem in a graph where each node represents a site and each edge connects an incompatible pair of sites [25, 7].

When M has missing entries, the Site-Removal problem is generalized to: **Problem S1:** Over all matrices created by imputing missing values in M, find the matrix M' to minimize the the solution to the Site-Removal problem on M'.

An ILP formulation for Problem S1 is easily obtained from the ILP formulation for Problem M1 as follows: Let $D(i)$ be a binary variable used to indicate whether or not site i will be removed. Then for each pair $(p, q) \in P$, add the inequality $D(p) + D(q) - C(p, q) \geq 0$, which says that if the missing values are imputed so that site pair (p, q) becomes incompatible, then either site p or site q must be removed. Also, for each pair $(p, q) \in F$, add the inequality $D(p) + D(q) \geq 1$. Finally, change the objective function in the formulation for M1 to Minimize $\sum_{i=1}^{m} D(i)$.

Problem S1 can be solved efficiently on the range of data considered in Section 2.1 (with most computations taking less than one second) but slightly slower than for Problem M1, and with a higher error rate. For example, the average time and error of the 50 datasets with $n = 120$, $m = 90$, $r = 16$, $I = 15$, was 0.53 seconds and 3.1% for Problem M1 and 0.75 seconds and 5.4% for Problem S1.

2.3 Estimating recombination in data with missing values

Recombination (crossing-over) is a fundamental molecular phenomena, where during meiosis two equal length sequences produce a third sequence of the same length consisting of a prefix of one of the sequences followed by a suffix of the other sequence. A central problem is to determine the *minimum* number of recombinations, denoted $Rmin(M)$, needed to generate a set of binary sequences M from some known or unknown ancestral sequence, when the infinite sites assumption applies [16]. There is a large literature on this problem, but there is no known efficient algorithm to exactly compute $Rmin(M)$. However, there are efficient algorithms that give relatively good *lower bounds* on $Rmin$, and there are biological questions concerning recombination (for example, finding recombination hotspots) that have been successfully addressed using lower bounds on $Rmin$ rather than using $Rmin$ itself [1],[5]. The first published, and most basic lower bound, called the HK bound [16], is obtained as follows: Consider the m sites of M to be integer points $1...m$ on the real line and pick a minimum number of non-integer points R so that for every pair of incompatible sites (p, q) in M, there is at least one point in R (strictly) between p and q. It is easy to show that $|R| \leq Rmin(M)$. When the data in M is complete, the HK bound $|R|$ can be computed in polynomial-time by a greedy-like algorithm. However, under the realistic situation that some entries in M are missing, we have **Problem R1:** Over all matrices created by imputing missing values in M, find the matrix M' to *minimize* the resulting HK lower bound on the $Rmin(M')$.

The reason for minimizing is that the result is then a valid lower bound on the number of recombinations needed to generate the true underlying data, when the infinite sites assumption applies.

Problem R1 is NP-hard [30], but an ILP formulation for it can be easily obtained from the formulation for Problem M1: For each c from 1 to $m-1$, let $R(c)$ be a binary variable used to indicate whether a point in R should be chosen in the open interval $(c, c+1)$. Then, for every pair of sites (p, q) in $F \cup P$, add the inequality $\sum_{p \leq c < q} R(c) \geq C(p, q)$ to the ILP for Problem M1, and change the objective function to Minimize $\sum_{c=1}^{m-1} R(c)$.

In our computations (details omitted due to space limitations), we have established that Problem R1 can be solved in practice over the same range of data discussed in Section 2.1; the computation times were longer than for Problem M1, but generally not more than twice as long. Of greater interest is the quality of the imputed values obtained in the solution to Problem R1 on data generated with recombination. Because Problem R1 more explicitly reflects recombination than does Problem M1, we conjectured that the solutions to Problem R1 would impute the original values better than solutions to Problem M1. Surprisingly, the average error for solutions to Problem R1 was somewhat larger than for Problem M1. For example, the average error over all datasets with $n = 150$ was 4% for Problem M1 and 4.75% for problem R1.

3 Haplotyping Problems

One of the key technical problems in the acquisition of variation data in populations is called the "Haplotype Inference (HI) Problem", or the problem of determining the "phase" of unphased genotype data. A very large literature now exists on this problem (see [12] for one survey). Abstractly, input to the HI problem consists of n *genotype* vectors, each of length m, where each value in the vector is either 0,1, or 2. A site with value 2 is called a "heterozygous" site, while the other sites are called "homozygous" sites. In the context of this problem, a vector with only entries of 0 and 1 is called a "haplotype". Given an input set of n genotype vectors, a solution to the HI Problem is a set of n pairs of haplotypes, one pair for each genotype vector. For any genotype vector g, the associated haplotypes 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. Hence, for an individual with h heterozygous sites there are 2^{h-1} pairs of haplotypes that could appear in a solution to the HI problem. For example, if the observed genotype g is 0212, then the pair of haplotypes 0110, 0011 is one feasible solution out of two feasible solutions. Of course, we want to find the HI solution that is most biologically plausible, and for that we need additional criteria to to guide the algorithm solving the HI problem. The goal is to devise criteria that reflect biological reality and yet allow efficient solution to the HI problem. Criteria have been previously proposed that were encoded as optimization problems with precise objective functions. In this paper, we discuss four additional biologically-motivated optimization problems that have practical ILP solutions.

3.1 Haplotyping versions of M1, S1, R1

Each of the three problems M1, S1 and R1 has a natural analog as a haplotyping problem, and has biological and historical connections to other haplotyping problems.

Problem HM1: Solve the HI problem so that the number of incompatible pairs of sites in the HI solution is *minimized* over all HI solutions.

Problem HM1 is a natural extension of the following "Perfect Phylogeny Haplotyping (PPH)" problem [10]: Find, if possible, a solution to the HI problem so that the haplotypes in the solution can be derived on a perfect phylogeny; in other words, so that there are *no* incompatible pairs of sites in the HI solution. Such an HI solution is called a "PPH solution", and if there is one, it can be found in linear time [6, 22]. See [10] for a discussion of the biological justification of the PPH problem. The PPH model is justified in some applications, but not all, and there are additional applications where the true haplotypes deviate by only a small amount from the PPH model (low recombination "haplotype blocks" are the prime example). In [13], a heuristic approach was developed to handle small deviations from the PPH model. An attempt to more formally model small deviations from the PPH model was explored in [26, 23]. Problem HM1 is an alternative way to formalize, and quantify, deviations from the PPH model: a set of genotypes that allow HI solutions with a small number of incompatible pairs deviate less from the PPH model than do genotypes that only allow HI solutions with a large number of incompatible pairs. We were therefore interested in whether the HM1 problem can be solved efficiently in a range of biologically relevant data, and how well the HI solutions obtained this way reconstruct the correct haplotypes.

An ILP formulation for Problem HM1 can be easily obtained by modifying the formulation for Problem M1: First, duplicate each row of M creating matrix $\overline{M}$, and create the ILP for Problem M1 using matrix $\overline{M}$, treating each 2 as a "?". Then for each cell (i, q) where $M(i, q)$ is 2, add the inequality $Y(2i-1, q) + Y(2i, q) = 1$. This formulation can be further improved, and such improvements have been implemented. For example, if $M(i, p) = 2$ and $M(i, q) = 1$ the binary combinations 0,1 and 1,1 will definitely be generated in columns (p, q) and that information may reduce the elements in $d(p, q)$, and reduce the size of the ILP formulation.

In the same way, we can modify the ILP for Problem S1 to obtain an ILP for **Problem HS1:** Remove the minimum number of columns in the input *genotypes*, so that there is a PPH solution to the HI problem on the remaining data. That is another way to formalize, and quantify, deviation from the PPH model. The same kind of modification also extends the ILP for Problem R1 to an ILP for **Problem HR1:** Solve the HI problem in order to minimize the HK bound on the haplotypes in the solution. That problem has been proposed as a way to search for recombination hotspots in genotypic data rather than haplotypic data [29]. It is interesting to note that Problem HR1 has a polynomial time solution [29], as does the problem of solving the HI problem in order to *maximize* the HK bound [31], even though Problem R1 is NP-hard.

Empirical results for Problem HM1 We extensively tested instances of Problems HM1, HS1 and HR1 using datasets with $\frac{n}{2}$ genotypes created by pairing n haplotypes output by the program *ms* described in Section 2.1. We tested 50 datasets for each combination of n, m and r that we examined. We observed that we could solve these problems in practical time on a wide range of data, but not as extensive as for Problem M1. Further, the computation times were longer (considerably so for larger instances) for the same parameter combinations of n, m and r. In general, the HI solutions given by Problem HM1 were better than for HS1 and HR1, and so we will only discuss those here, although the running times for HM1 were larger than for HR1. In our experiments, we stopped any computations that exceeded three hours, and considered only combinations of $n = \{10, 20, 30, 60, 80\}$ haplotypes and $m = \{30, 60, 90\}$ sites. As an example, when $n = 80, m = 60, r = 16$, 94% of the datasets terminated within the three hour limit. However, for most of the parameter choices we examined, all of the datasets terminated within the time limit, and most terminated well below that limit.

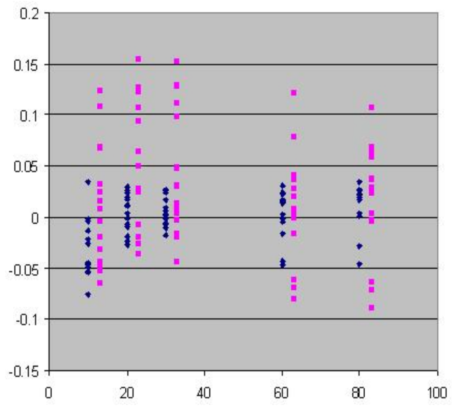
In addition to the solution time, we were interested in the quality of the haplotypes produced and how that quality varied depending on whether the deviation from the PPH model was due to recombination or to homoplasy. Datasets with homoplasy were generated with 5, 10 or 20 sites where additional mutations were forced to occur. Over these ranges of of n, m and r, we did not see a significant difference between the quality of the haplotypes produced from datasets generated with recombination and those with homoplasy, and so we will discuss only the recombination case.

To assess the quality of the solutions, we used the standard *switch error* [17, 19] and the *line error*, comparing the haplotype pairs obtained from solving Problem HM1 with the original pairs used to generate the genotype data. The switch error is the minimum number of runs (blocks) of contiguous sites that need to be exchanged between the computed haplotype pairs in order to make the resulting haplotype pairs agree with the correct pairs, divided by the number of hetrozygous sites in the data. The line error is simply the number of haplotype pairs in the solution that do not agree completely with the corresponding correct pair, divided by the number of genotypes. The ILP executions that were terminated after three hours all found HI solutions, and so we could test their quality also. Hence, no datasets were excluded from our accuracy analysis. We also compared the accuracy of the HM1-computed haplotypes with the haplotypes found by program FastPhase [24], the successor program of the widely-used program PHASE [28].

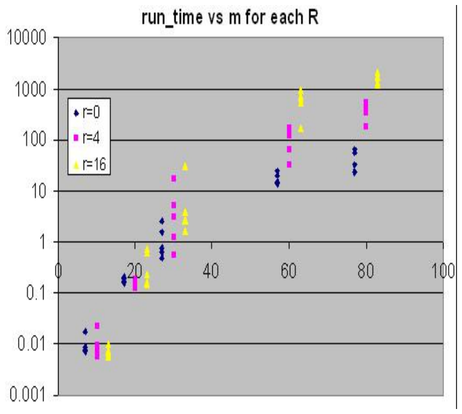
We observed switch errors for the HI solutions produced by solving Problem HM1 that were very good in some parameter ranges, often superior (by a small amount) to the switch error of the solutions produced by FastPhase; in other parameter ranges the observed switch errors were somewhat inferior to those from FastPhase, and to accuracies reported for simulations using HapMap data [20] (although in line with some real data [17]). The line errors of solutions from both FastPhase and Problem HM1 were relatively large, but quite similar to each other. Unlike imputation error, switch accuracy is highly influenced by r, the recombination parameter; consistent with imputation error, all accuracies improved with larger data sets, particularly as the number of rows increase. We should note that in our tests we did not require that the minor allele appear above a minimum frequency as is commonly done; it is well known [19] that accuracies are improved by imposing that requirement.

Figure 2 (a) summarizes the switch and line errors of the HI solutions from obtained Problem HM1 compared to solutions given by FastPhase; Figure 2 (b) shows the time needed to solve Problem HM1.

As an illustration of the influence of n, when $n = 20, m = 30, r = 4$, the average switch errors from HM1 and FastPhase were 0.1189 and 0.136 which are relatively large, but when n increased to 60, the errors declined to 0.0532 and



(a)



(b)

Fig. 2. (a) Comparison of switch and line errors the HI solutions from Problem HM1 and FastPhase, as a function of n. In each simulated dataset the switch and line errors of the HM1 solution were subtracted from switch and line errors of the FastPhase solution. A positive result shows the HM1 solution superior to the FastPhase solution. Each diamond (square) is the average difference of switch (line) errors from the 50 datasets for a particular combination of the parameters n, m and r. (b) Solution times (over the terminating datasets) as a function of m, for three values of parameter r. Each object is the average of 50 datasets with the same values of n.

0.068, and when n increased to 80, the errors were 0.0452 and 0.062. To see the influence of the recombination parameter r, consider the case of $n = 60, m = 30$, where the switch errors for HM1 were 0.046, 0.0532, 0.0984 with $r = 0, 4$ and 16 respectively, and the errors for FastPhase were 0.063, 0.068, and 0.083.

The comparison of *individual* HI solutions obtained from Problem HM1 and from FastPhase often gave contradictory results, so we averaged the results over all the data examined: the HI solutions from Problem HM1 had an average switch error of 0.13, an average line error of 0.34 and required an average computation time of 186.3 seconds for the terminating computations, while 4% of the computations did not terminate in three hours. The FastPhase solutions had an average switch error of 0.128, an average line error of 0.36 and required an average of 38 seconds to compute. These results suggest that the qualities of the two approaches are very similar. While the time needed to solve Problem HM1 is greater than the time for FastPhase, the main goal in solving HM1 was to see how well this natural extension of perfect phylogeny haplotyping solves the HI problem (although we would have been pleased to report that it solved faster than FastPhase). Having a solvable, simple-to-state objective function allows one to assess the biological fidelity of the model reflected by the objective function, giving much cleaner and clearer semantics compared to more black-box methods whose semantics may be very unclear. We consider the results based on HM1 to be positive and informative.

3.2 The MinPPH Problem

When there is a PPH solution to the HI problem, there may be several solutions, and it is desirable to apply a secondary criterion to choose one. An appealing approach, motivated both by theory and empirical observations, is to solve the following problem called the **MinPPH Problem:** Find a PPH solution that *minimizes* the number of *distinct* haplotypes used in any of the PPH solutions.

The MinPPH Problem is a mixture of the PPH problem and the problem of Haplotype Inference by Pure Parsimony (denoted HIPP) [11, 4, 18]. The MinPPH problem was defined and justified in [2] where it was shown to be NP-hard. An ILP formulation for MinPPH (different than presented here) was described in [3], but not implemented due to the expectation that it would not solve efficiently.

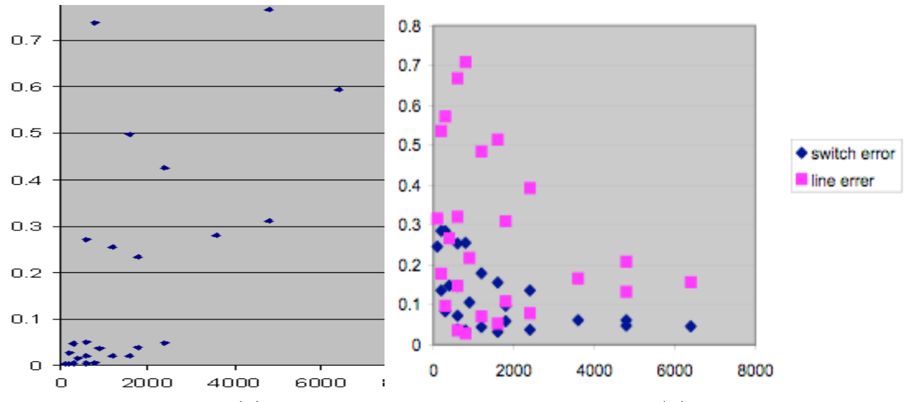
The idea of our ILP formulation is to modify the formulation for Problem HM1 and combine it with the simplest ILP formulation for the HIPP problem given in [4] (see also [12] for a description of that HIPP formulation, and [18] for a similar formulation). We start with the HIPP formulation from [4], but add to it the inequalities from the HM1 formulation along with the equality $\sum_{(p,q)\in P} C(p,q) = 0$. The end result is an ILP formulation that solves the HI problem using the minimum number of distinct haplotypes possible, subject to the constraint that the HI solution is a PPH solution (assuming a PPH solution exists).

We extensively tested this ILP formulation for solution speed and haplotype accuracy, using genotypic data (created from *ms* with $r = 0$) where PPH solutions were assured. We obtained two striking empirical results. The first result is that the ILPs solve extremely fast (generally less than one second) over a range of data up to 80 rows and 80 columns. This speed is even more notable considering that the HIPP formulation from [4] requires hours or days to solve the HI problem on the smaller instances, and cannot solve the larger instances in practical time. Hence, it is the PPH constraints added to the HIPP formulation that makes the resulting formulation solve so quickly. We also examined the question of how the running times were influenced by the number of PPH solutions, and we saw no clear pattern.

The second striking empirical result is that the accuracy of the HI solutions given by the MinPPH solution is notably better than solutions obtained by FastPhase (except for instances with a very small number of rows), while running considerably faster. For example, when $n = m = 80$, the average MinPPH solution time was 0.59 seconds with a switch-error of 0.045, while the run time for FastPhase was 163 seconds with a switch-error of 0.074. In general, the MinPPH switch and line errors decrease with increasing problem size, as measured either by n or $m \times n$. Figures 3 (a) and 3 (b) show the MinPPH runtime and switch-error and line-error as a function of $n \times m$.

Acknowledgements

We thank Yun Song for the use of the code to generate SNP sequences with homoplasy events and for helpful conversations and suggestions. We thank Chuck



(a) (b)

Fig. 3. (a) Average time needed to solve the ILP for Problem MinPPH, as a function of $n \times m$. Each diamond is the average of 50 datasets for a particular combination of the parameters n, m. (b) Average switch and line errors as a function of $n \times m$.

Langley for helpful conversations and suggestions. The research was supported by NSF grants CCF 0515278, IIS 0513910 and REU 0434759.

References

1. V. Bafna and V. Bansal. Improved recombination lower bounds for haplotype data. Proceedings of RECOMB 2005.
2. V. Bafna, D. Gusfield, S. Hannenhalli, and S. Yooseph. A note on efficient computation of haplotypes via perfect phylogeny. *Journal of Computational Biology*, 11(5):858–866, 2004.
3. D. Brown and I. Harrower. A new formulation for haplotype inference by pure parsimony. report cs-2005-03. Technical report, University of Waterloo, School of Computer Science, 2005.
4. D.G. Brown and I.M. Harrower. Integer Programming Approaches to Haplotype Inference by Pure Parsimony. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(2):141–154, 2006.
5. International HapMap Consortium. A haplotype map of the human genome. *Nature*, 437:1299–1320, 2005.
6. Z. Ding, V. Filkov, and D. Gusfield. A linear-time algorithm for the perfect phylogeny haplotyping problem. In *Proceedings of RECOMB 2005, LNBI Vol. 3500*, pages 585–600. Springer, 2005.
7. J. Felsenstein. *Inferring Phylogenies*. Sinauer, Sunderland, MA., 2004.
8. D. Gusfield. Efficient algorithms for inferring evolutionary history. *Networks*, 21:19–28, 1991.
9. D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, UK, 1997.
10. 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.

11. D. Gusfield. Haplotype inference by pure parsimony. In R. Baeza-Yates, E. Chavez, and M. Chrochemore, editors, *14th Annual Symposium on Combinatorial Pattern Matching (CPM'03)*, volume 2676 of *Springer LNCS*, pages 144–155, 2003.
12. D. Gusfield and S. Orzack. Haplotype inference. In S. Aluru, editor, *Handbook of Computational Molecular Biology*, pages 18/1– 18/25. Chapman and Hall/CRC, 2005.
13. E. Halperin and E. Eskin. Haplotype reconstruction from genotype data using Imperfect Phylogeny. *Bioinformatics*, 20:1842–1849, 2004.
14. J. Hein, M. Schierup, and C. Wiuf. *Gene Genealogies, Variation and Evolution: A primer in coalescent theory.* Oxford University Press, UK, 2005.
15. R. Hudson. Generating samples under the Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–338, 2002.
16. R. Hudson and N. Kaplan. Statistical properties of the number of recombination events in the history of a sample of DNA sequences. *Genetics*, 111:147–164, 1985.
17. G. Kimmel and R. Shamir. GERBIL: Genotype resolution and block identification using likelihood. *PNAS*, 102:158–162, 2005.
18. G. Lancia, C. Pinotti, and R. Rizzi. Haplotyping populations by pure parsimony: Complexity, exact and approximation algorithms. *INFORMS J. on Computing, special issue on Computational Biology*, 16:348–359, 2004.
19. S. Lin, D. Cutler, M. Zwick, and A. Chakravarti. Haplotype inference in random population samples. *Am. J. of Hum. Genet.*, 71:1129–1137, 2002.
20. J Marchini, P. Donnelly, and et. al. A comparison of phasing algorithms for trios and unrelated individuals. *Am. J. of Human Genetics*, 78:437–450, 2006.
21. I. Pe'er, T. Pupko, R. Shamir, and R. Sharan. Incomplete directed perfect phylogeny. *SIAM J. on Computing*, 33:590–607, 2004.
22. R. V. Satya and A. Mukherjee. An optimal algorithm for perfect phylogeny haplotyping. In *Proceedings of 4th CSB Bioinformatics Conference*, Los Alamitos, CA, 2005. IEEE Press.
23. R. V. Satya, A. Mukherjee, G. Alexe, L. Parida, and G. Bhanot. Constructing near-perfect phylogenies with multiple homoplasy events. *Bioinformatics*, 22:e514–i522, 2006. Bioinformatics Suppl., Proceedings of ISMB 2006.
24. P. Scheet and M. Stephens. A fast and flexible statistical model for large-scale population genotype data: applications to inferring missing genotypes and haplotypic phase. *Am. J. Human Genetics*, 78:629–644, 2006.
25. C. Semple and M. Steel. *Phylogenetics.* Oxford University Press, UK, 2003.
26. Y. S. Song, Y. Wu, and D. Gusfield. Haplotyping with one homoplasy or recombination event. Proceedings of WABI 2005, LNCS Vol. 3692.
27. M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *J. of Classification*, 9:91–116, 1992.
28. 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.
29. C. Wiuf. Inference of recombination and block structure using unphased data. *Genetics*, 166:537–545, 2004.
30. Y. Wu. Personal Communication.
31. Y. Wu and D. Gusfield. Efficient computation of minimum recombination over genotypes (not haplotypes). In *Proceedings of Life Science Society Computational Systems Bioinformatics (CSB) 2006*, pages 145–156, 2006.