Integer Programming Approaches to Haplotype Inference by Pure Parsimony

Daniel G. Brown and Ian M. Harrower

Abstract—In 2003, Gusfield introduced the Haplotype Inference by Pure Parsimony (HIPP) problem and presented an integer program (IP) that quickly solved many simulated instances of the problem [1]. Although it solved well on small instances, Gusfield's IP can be of exponential size in the worst case. Several authors [2], [3] have presented polynomial-sized IPs for the problem. In this paper, we further the work on IP approaches to HIPP. We extend the existing polynomial-sized IPs by introducing several classes of valid cuts for the IP. We also present a new polynomial-sized IP formulation that is a hybrid between two existing IP formulations and inherits many of the strengths of both. Many problems that are too complex for the exponential-sized formulations can still be solved in our new formulation in a reasonable amount of time. We provide a detailed empirical comparison of these IP formulations on both simulated and real genotype sequences. Our formulation can also be extended in a variety of ways to allow errors in the input or model the structure of the population under consideration.

Index Terms—Computations on discrete structures, integer programming, biology and genetics, haplotype inference.

1 Introduction

For the last few years, an emphasis in human genomics has been on identifying common human genetic variations. A particular focus has been on identifying *Single Nucleotide Polymorphisms* (SNPs), point mutations found with only two common alleles in the population, and in tracking their inheritance.

Technological limitations make this problem difficult. At genomic positions where an individual inherited two different alleles, it is currently expensive to connect parents with the alleles inherited from them [4]. Instead, instruments can only identify that the individual is heterozygous at that position. If we could identify maternal and paternal inheritance better, we could trace the structure of human populations more accurately and improve our ability to map disease genes. This process of going from genotypes (which include this ambiguity at heterozygous positions) to haplotypes (where we connect parents to the alleles inherited from them) is called *haplotype inference*.

Gusfield [5] brought this problem into the combinatorial literature. In 2003, he studied the variant where, given genotypes of several members of the same population, we are to find the smallest collection of haplotype sequences that could explain all of the genotype sequences [1]. This problem, which he called *Haplotype Inference by Pure Parsimony (HIPP)*, is the focus of this paper.

Gusfield [1] gave an integer linear programming (IP) formulation for the HIPP problem, which he called the RTIP. Its size grows exponentially with the number of heterozygous positions in genotypes. For problems giving rise to moderate-sized IP instances, the IP typically solves extremely quickly, but many instances are simply too big.

Two separate groups, Halldórsson et al. [2] and Lancia et al. [3], have published polynomial-sized IP formulations for HIPP. We independently discovered the same IP as Halldórsson et al. and augmented it with several constraints that can be added to the formulation to assist in solving it (Section 4). One of these classes of cuts is data-driven constraints based on patterns in the input genotype data. These constraints have the interesting property that, as the size of the pattern increases, the constraints become stronger and have great potential in helping to solve many of our instances. We have learned from Edwards [6] that the work of Halldórsson et al. was done at approximately the same time and that they also found similar cuts to those presented in Section 4, but did not present these results.

In our experiments, we note that the polynomial-sized IP solved many problem instances, but was typically much slower than Gusfield's RTIP. The primary concern with the RTIP is whether it is small enough to be stored in memory. For large instances of HIPP, the RTIP is very large.

In Section 5, we produce a compromise between these approaches. We give a polynomial-size IP formulation for the HIPP problem, which we call the HybridIP, that achieves much of the speed advantage of the exponential-size RTIP formulation [1]. Like Gusfield's approach, it chooses a set of possible explaining haplotypes for some of the genotypes. However, it does not consider all possible haplotypes for a population, which could be of exponential size. Like the polynomial IP of Halldórsson et al. [2], it explicitly represents the haplotypes chosen to explain every member of the population.

Our experiments show that our new formulation solves much faster than the initial polynomial-sized IP. While its runtimes never approach the speed of Gusfield's IP for instances where Gusfield's can be stored, they are still practical; many problems where Gusfield's would be too

• The authors are with the David R. Cheriton School of Computer Science, University of Waterloo, 200 University Ave. West, Waterloo, ON N2L 3G1 Canada. E-mail: {browndg, imharrow}@cs.uwaterloo.ca.

Manuscript received 28 July 2005; revised 6 Dec. 2005; accepted 2 Jan. 2006; published online 1 May 2006.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-0087-0705.

large to be stored can still be solved by our HybridIP in practical runtimes.

We also show how our new formulation can be adapted to a variety of extensions of HIPP recently summarized by Kalpakis and Namjoshi [7]. Most of these extensions concern various types of error in the data, but one concerns possible instrument limitations and two begin to bring the problem into a scenario where some of the structure of the population under study is known. We also provide a set of additional constraints which may help in solving some HIPP instances.

Our results give a better approach than was previously known to solve potentially difficult instances of HIPP and offer possible heuristic approaches to improve solution times when they are too large.

2 Problem Definition, Notation, and Previous Formulations

In the haplotype inference problem, we are given as an input a collection of n *genotype vectors*, each of which corresponds to a different population member $p_1, \ldots, p_n$. Each of these n vectors tests the same m SNPs, $s_1, \ldots, s_m$. Our task is to identify which allele each parent of population member p_i contributed at each SNP s_j. To simplify, we identify each allele as being 0 or 1.

Using notation suggested by He and Zelikovsky [8], we will treat this collection of genotypes as an $n \times m$ matrix $G \in \{0,1,2\}^{n \times m}$. In this matrix, we set $G[i,j] = 0$ when population member p_i is homozygous with allele 0 at position s_j, $G[i,j] = 1$ when p_i is heterozygous at position s_j, and $G[i,j] = 2$ when p_i is homozygous with allele 1 at position s_j. (We use this notation because, under it, a genotype is equal to the sum of its haplotypes. It is *not* the standard used in the biological literature, wherein the roles of 1 and 2 are typically reversed.) We will refer to the rows of G as genotype vectors; each vector includes the genotypes of all tested SNPs in one population member.

In this representation, a *haplotype vector* is a binary m-vector, h, which represents the sequence of a single chromosome; $h[i] = 0$ if the chromosome has allele 0 at SNP s_i and $h[i] = 1$ if the chromosome has allele 1 at SNP s_i. In haplotype inference, we attempt to identify haplotype vectors that represent the haplotypes contributed by the parents of the genotypes in G. We make no distinction between the maternal and paternal haplotypes and refer to these haplotypes simply as the *parents* of a genotype. Genotype g is *explained* by two haplotypes h_1 and h_2 (and h_1 and h_2 form an *explaining pair* for g) if $g[j] = h_1[j] + h_2[j]$ at all positions j of the genotype g. At homozygous positions in g, with value 0 or 2, both parents must have the same value, while, at heterozygous positions, with value 1, one parent must have allele 1, while the other has allele 0. We order the parents of a genotype lexicographically, resolving the first ambiguous site in the genotype.

Haplotype inference involves picking a set H of haplotypes such that every genotype in G is explained by a pair of haplotypes from H. A simple objective is to find the *smallest* set H^* of haplotypes, where every genotype is explained by a pair of members of H^*. Gusfield [1] justified this objective by noting that few haplotypes seem to be found in recombination-free blocks of the human genome [9], [10] and by noting that previous methods for haplotype inference, most notably those due to Clark [11], found that their results were more likely correct when they returned small sets of haplotypes. New studies of variations in humans and dogs have confirmed the observation of few haplotypes in recombination-free blocks [12], [13], [14]. This goal of minimizing the number of haplotypes is called *Haplotype Inference by Pure Parsimony* and has been studied by several authors (e.g., [1], [2], [3], [7], [15], [16], [17], [18]).

$$
\boxed{
\begin{array}{rll}
\text{minimize} \sum_{j=1}^{|H|} x_i, & \text{subject to} & \\
\sum_{\rho \in R_i} w_{i,\rho} = 1, & \text{for all genotypes } g_i & (1) \\
x_j \geq w_{i,(j,k)}, & \text{for all } i,j \text{ and } k & (2) \\
x_k \geq w_{i,(j,k)}, & \text{for all } i,j \text{ and } k & (3) \\
x_j, w_{i,\rho} \in \{0,1\}, & \text{for all } i,j \text{ and } k &
\end{array}
}
$$

Fig. 1. Gusfield's TIP formulation for Haplotype Inference by Pure Parsimony [1].

The HIPP problem is NP-complete (Gusfield [1] cites correspondence from Hubbell for this fact; a proof is in Lancia et al. [3]) and the best known approximation ratio for this problem comes from a simple linear programming rounding approach giving an exponential guarantee on the approximation factor [3]. As such, exponential time algorithms such as integer programming are appropriate as a way to solve the problem exactly.

In 2003, Gusfield [1] developed the TIP formulation, an exponential-sized IP formulation for the HIPP problem. We create a set H of all possible haplotypes that can explain the genotypes by expanding all possible pairs of parents for each genotype. For each genotype g_i, its explaining set is $R_i = \{(j,k) : h_j \text{ and } h_k \text{ are an explaining pair for } g_i\}$. For each pair ρ in R_i, we create a binary variable $w_{i,\rho}$, set to 1 exactly when the pair is selected to resolve genotype g_i. For each possible haplotype h_i $(i = 1 \ldots |H|)$, there is a variable x_i; its value is 1 if the haplotype is used and 0 otherwise.

Constraint (1) ensures that each genotype is explained by a pair of haplotypes. Constraints (2) and (3) ensure that if $w_{i,(j,k)}$ is 1, selecting pair (j,k), the required haplotypes j and k are included in the set of haplotypes chosen. The objective is to minimize the number of chosen haplotypes, which is the sum of the x_i variables. The IP is shown in Fig. 1.

Gusfield also presented a size reduction for the IP that avoids expanding out all possible parents and, so, reduces the number of variables. Instead, it only includes pairs where at least one parent could partly explain another genotype. This reduced formulation, the RTIP, is much smaller in practice, though still exponential-size in the worst case. Gusfield shows how to create the RTIP in time proportional to its size plus a polynomial in the problem instance size, so, if it is of moderate size, it can be produced in reasonable runtime.

$$\begin{array}{rll}
\text{minimize} \sum_{i=1}^{2n} x_i, & \text{subject to} & \\
y_{2i-1,k} + y_{2i,k} = g_i[k], & \text{for all } 1 \le i \le n & (4) \\
 & \text{and all } k & \\
d_{i,j} \ge y_{i,k} - y_{j,k}, & \text{for all } 1 \le i < j & (5) \\
 & \le 2n \text{ and all } k & \\
d_{i,j} \ge y_{j,k} - y_{i,k}, & \text{for all } 1 \le i < j & (6) \\
 & \le 2n \text{ and all } k & \\
x_i \ge 2 - i + \sum_{j=1}^{i-1} d_{j,i}, & \text{for all } 1 \le i \le 2n & (7) \\
x_i, y_{i,k}, d_{i,j} \in \{0,1\}, & \text{for all } i,j \text{ and } k. &
\end{array}$$

Fig. 2. The polynomial-sized IP formulation for Haplotype Inference by Pure Parsimony introduced by Halldórsson et al. [2].

3 A Polynomial-Sized IP

In 2003, two groups [2], [3] independently identified similar polynomial-sized formulations for HIPP. Slightly later, and also independently, we discovered the same formulation as Halldórsson et al. [2] and experimented with it to see if it is practical [16]. We will call this formulation the PolyIP. It explicitly represents the chosen haplotype vectors in decision variables.

For each genotype g_i, we create variables representing each of the m characters of the two explaining haplotypes. Genotype g_i is explained by h_{2i-1} and h_{2i} and variable $y_{i,k}$ represents the value of haplotype i in position k. Constraint (4) ensures that each genotype is properly explained.

To count the unique haplotypes explaining the input genotypes, we use binary variables to mark pairwise differences between two haplotypes. For each pair (i, j) of haplotypes $1 \le i < j \le 2n$, we create a variable $d_{i,j}$, equal to 1 when $h_i \neq h_j$. If $h_i \neq h_j$, there is some position k in the haplotypes where $h_i[k] \neq h_j[k] = 0$, or $h_i[k] \neq h_j[k] = 1$, and (5) or (6) will force $d_{i,j}$ to 1.

For each haplotype h_i, we introduce a binary variable x_i, which is 1 if h_i is unique in $(h_1, \ldots, h_i)$. This condition is enforced by (7). We can minimize the number of unique haplotypes used by minimizing the sum of these x_i variables. The complete PolyIP formulation is in Fig. 2.

Clearly, many of the variables in the IP can take only one possible value in a feasible solution. When creating an IP to solve, variables with predetermined values should be replaced by constants, thus simplifying or eliminating the constraints involving those variables. We also order the parents of a genotype lexicographically, which resolves the first ambiguous site in every genotype during the construction of the IP.

4 Cuts and Modifications

Solving an IP is NP-Hard in general. Typically, one solves an IP by relaxing the program, replacing integrality constraints with corresponding range constraints, thus producing the linear programming (LP) relaxation. Feasible solutions to a minimization IP are also feasible for its LP relaxation, so an optimal solution to the LP relaxation gives a lower bound on the optimal solution to the IP.

The LP relaxation can be solved efficiently. If its solution is integral, it optimizes the IP. Otherwise, a common approach is to find new constraints, called *valid cuts*, that are violated by the solution to the LP relaxation, but are satisfied by all feasible integer solutions. Or, one can *branch*, by picking a single variable and creating a series of new programs where it is set to each of its possible integer values. The best solution found is the optimal solution to the original IP. The process of adding valid cuts and branching when no effective cuts can be found is called *branch-and-cut* [19].

Unfortunately, the PolyIP formulation in the previous section does not behave well under LP relaxation. At heterozygous positions, the $y_{i,k}$ variables will often have value 0.5 in the optimal LP solution. This does not resolve the haplotypes and allows the $d_{i,j}$ variables to have value zero, which gives no information. Thus, we need to strengthen the formulation. In this section, we describe a series of enhancements to tighten the gap between the optimal solution to the LP relaxation and the optimal solution to the IP. We both augment the objective function and show several new valid constraints.

4.1 Augmented Objective Function

A weakness of the PolyIP (and of its LP relaxation) is that optimal solutions may allow $d_{i,j} = 1$ even when haplotypes h_i and h_j are the same. This is harmless for the IP, but, in the case of the LP, it allows the $y_{i,k}$ variables to take fractional values. Fundamentally, it also violates our intuition in designing the program itself: The $d_{i,j}$ variables should only equal 1 when haplotypes h_i and h_j are different.

The LP relaxation has some other undesirable behaviours. Constraint (7) depends strongly on the order of the genotypes in the input. Consider some haplotype h_i. If $h_i = h_1$, then x_i will be set to 0 in an optimal solution regardless of the values of $d_{2,i}, d_{3,i}, \ldots d_{i-1,i}$. This means that each such $d_{j,i}$ can be assigned 1 in an optimal solution. Again, we want $d_{j,i}$ to be assigned 1 only when h_i and h_j differ.

To bias the $d_{i,j}$ variables toward 0, we slightly perturb the objective function to bias the program away from solutions with positive $d_{i,j}$ variables. Perturbation is a classical linear programming approach to avoid degeneracy [20], which helps optimize linear programs in small amounts of time, but we can also use it to slightly bias the LP solver toward solutions of the same overall objective function value that have more of the $d_{i,j}$ variables resolved to zero. Traditionally, small perturbations either take random form or are treated as symbolic constants; we have chosen the first of these for simplicity. (Indeed, many contemporary LP solvers use perturbation approaches; again, the distinction is that we require the perturbations to always be positive.) We add a small random perturbation $r_{i,j}$ to each coefficient, which gives the following extended objective function:

$$\text{minimize} \sum_{i=1}^{2n} (x_i) + \sum_{i=2}^{2n} \sum_{j=1}^{i-1} (\epsilon + r_{i,j}) \cdot d_{i,j}. \qquad (8)$$

By choosing ϵ so that $\epsilon < \frac{1}{8n^2}$,and choosing $r_{i,j}$ uniformly over $(0, \frac{\epsilon}{2})$, we ensure that the coefficients are all in the same scale and that the sum of all the $d_{i,j}$ coefficients is strictly less than 1.

This difficulty in setting the $d_{i,j}$ variables can be addressed in a different IP modeling approach to the HIPP problem; in this approach, we have a difference variable $d_{i,j}^k$, set to 0 if $h_i[k] = h_j[k]$ and 1 otherwise. We will use these new $d_{i,j}^k$ variables to set the $d_{i,j}$ variables, both from below and from above. This is equivalent to the approach of Lancia et al. [3].

The constraints used to assign these difference variables are based on an observation about the product of binary variables. For two binary variables u and v, let $P(u,v)$ be a binary variable representing their product. We can force $P(u,v) = uv$ with three linear constraints:

$$P(u,v) \leq u, \tag{9}$$

$$P(u,v) \leq v, \tag{10}$$

$$P(u,v) \geq u + v - 1. \tag{11}$$

We now define the constraints for the $d_{i,j}^k$ variables. We wish to have $d_{i,j}^k = y_{i,k} \cdot (1 - y_{j,k}) + (1 - y_{i,k}) \cdot y_{j,k}$. If the two binary y variables take the same value, then the expression evaluates to 0. If the variables take different values, then the expression evaluates to 1. However, this is not a linear constraint. We need to introduce product variables as in (9)-(11). This gives the constraint:

$$d_{i,j}^k = y_{i,k} + y_{j,k} - 2 \cdot P(y_{i,k}, y_{j,k}). \tag{12}$$

Note that if one of the genotypes involved is homozygous at site k, then the product variable $P(y_{i,k}, y_{j,k})$ is not needed. If both genotypes are homozygous at site k, then the $d_{i,j}^k$ variable is a constant. There are $O(mn^2)$ of these constraints, with $O(mn^2)$ additional variables needed for the products. The advantage to using the product for this calculation is that the $d_{i,j}^k$ variables behave as desired in any integer feasible solution.

We now have an indication of whether the haplotypes differ at each site. We need to mark differences at the string level. To do this, we introduce variables $d_{i,j}$, which are assigned 0 if the haplotypes h_i and h_j are equal and 1 otherwise. Each $d_{i,j}$ variable needs to be larger then than each of the individual character variables $d_{i,j}^k$. To force tight conformance to the desired behavior, we also require that the $d_{i,j}$ is less than the sum of the character difference variables $d_{i,j}^k$. We introduce the following constraints:

$$d_{i,j} \geq d_{i,j}^k \quad \text{for all } 1 \leq i < j \leq 2n \text{ and each site } k, \tag{13}$$

$$d_{i,j} \leq \sum_k d_{i,j}^k \quad \text{for all } 1 \leq i < j \leq 2n. \tag{14}$$

As an integer programming modeling technique, this is appropriate, though it builds a much larger integer program. Unfortunately, it may not extend well to the LP relaxation since we can wind up with the $d_{i,j}$ variables constrained to a very large range when the lower bound (any single value of $d_{i,j}^k$) is as small as $1/m$, while the upper bound (the sum of the $d_{i,j}^k$ values) is 1. As such, while this approach properly sets the $d_{i,j}$ values in the integer program, it need not do so for their LP relaxations.

4.2 Transitivity Constraints

The $d_{i,j}$ variables are intended to equal 1 if h_i and h_j are different and 0 otherwise. The new objective function partially helps us achieve this behavior for the LP relaxation, but adding new constraints can help further. We add the following transitivity constraints for all $1 \leq i < j < k \leq 2n$:

$$d_{i,j} + d_{i,k} \geq d_{j,k}, \tag{15}$$

$$d_{i,j} + d_{j,k} \geq d_{i,k}, \tag{16}$$

$$d_{i,k} + d_{j,k} \geq d_{i,j}. \tag{17}$$

Consider three haplotypes, h_i, h_j, and h_k. If $h_i = h_j$ and $h_i = h_k$, then we can conclude that $h_j = h_k$. Therefore, if $d_{i,j}$ and $d_{i,k}$ are both 0, (15) forces $d_{j,k}$ to also be 0. The constraint also enforces the contrapositive: If $h_j \neq h_k$, then $d_{j,k} = 1$ and (15) ensures that $d_{i,j} + d_{i,k} \geq 1$. Since these variables are binary, this constrains at least one of h_j and h_k to be different from h_i.

With the original objective function, it is possible that integer solutions to the IP will violate this new constraint, but there are always equivalent integer solutions that do not. We show that these constraints are still valid for an optimal solution to the HIPP problem by considering any optimal solution to the problem. If we modify the solution such that $d_{i,j} = 1$ if h_i and h_j are different and $d_{i,j} = 0$ when h_i and h_j are the same, leaving all other variables the same, this is an optimal solution for the HIPP instance that satisfies all the transitivity constraints.

4.3 Data-Driven Cuts

We next consider cuts that depend on input data patterns. These constraints are violated by many fractional solutions. They thus reduce the gap between the optimal solutions of the LP relaxation and the IP.

Consider input genotypes g_1 and g_2 and suppose there exists a k such that $g_1[k] = 1$ and $g_2[k] = 0$. Then, $d_{1,3} + d_{2,3} \geq 1$ since the parents of g_1 cannot both be 0 at position k, while the parents of g_2 must both be 0 at position k. This can be concluded directly from the transitivity constraint (17), which says that $d_{1,3} + d_{2,3} \geq d_{1,2}$, since the parents of g_1 must be different, and $d_{1,2} = 1$. In general, if $g_i[k] = 1$ and $g_j[k] = 0$, then the following are valid cuts:

$$d_{2i-1,2j-1} + d_{2i,2j-1} \geq 1, \tag{18}$$

$$d_{2i-1,2j} + d_{2i,2j} \geq 1. \tag{19}$$

While these constraints are implied by the transitivity constraints, the data-driven cuts can be extended to stronger cuts based on larger patterns. Consider once again two genotypes, but consider two characters in each, such as k_1 and k_2. Then, an occurrence of this pattern in population members i and j,

$$g_i[k_1, k_2] = 10$$
$$g_j[k_1, k_2] = 01,$$

gives the valid constraint:

$$d_{2i-1,2j-1} + d_{2i,2j-1} + d_{2i-1,2j} + d_{2i,2j} \geq 3. \tag{20}$$

This constraint requires that at most one parent of g_i is the same as a parent of g_j. Assume, without loss of generality, that the first parent of g_i equals the first of g_j. Then, both of these parents have value 0 in positions k_1 and k_2. The second haplotype for g_i has $h_{2i}[k_1, k_2] = 10$, while the second haplotype for g_j has $h_{2j}[k_1, k_2] = 01$, so no other equalities can occur among these haplotypes. The constraint $d_{2i-1,2j-1} + d_{2i,2j-1} + d_{2i-1,2j} + d_{2i,2j} \geq 3$ is valid for all feasible integer solutions to the IP.

This new constraint is strictly stronger than the transitivity constraints (18) and (19). Consider the fractional solution $d_{2i-1,2j-1} = d_{2i,2j-1} = d_{2i-1,2j} = d_{2i,2j} = \frac{1}{2}$. These variable assignments satisfy $d_{2i-1,2j-1} + d_{2i,2j-1} \geq 1$ and $d_{2i-1,2j} + d_{2i,2j} \geq 1$, but violate the new constraint $d_{2-1i,2j-1} + d_{2i,2j-1} + d_{2i-1,2j} + d_{2i,2j} \geq 3$, which forces the average of the four pairwise difference variables to be at least $\frac{3}{4}$.

This approach generalizes to patterns in a set $A = \{g_i, g_{j_1}, g_{j_2}, \ldots, g_{j_{\ell-1}}\}$ of ℓ genotype vectors. As before, we generate a constraint that allows at most one parent of the first genotype g_i to be the same as a parent of at most one of the remaining $\ell - 1$ genotypes in A. The constraint also enforces that the genotypes of $A - \{g_i\}$ all have entirely different parents. We ensure this is correct by requiring that all pairs of genotypes in $A - \{g_i\}$ have a position in which one genotype vector has a 2 and the other a 0. We also require one column k_1 in which $g_i[k_1] = 1$ and all other genotypes in A have the same non-1 value. This column k_1 guarantees that if h_{2i-1} equals the parent of a member of $A - \{g_i\}$, then h_{2i} is different from all the other parents of the members of $A - \{g_i\}$. Finally, we require a column k in which $g_i[k]$ is not 1 and all other genotypes are 1. This column forces the parents of each g_j to be different from each other and thus equal to only one of g_i's parents. When such a collection of rows and columns occurs, we obtain the valid cut:

$$\sum_{x=1}^{\ell-1} (d_{2i-1,2j_x-1} + d_{2i-1,2j_x} + d_{2i,2j_x-1} + d_{2i,2j_x}) \geq 4\ell - 5. \tag{21}$$

This constraint allows at most one parent of the first genotype g_i to be a parent of at most one of the $\ell - 1$ remaining genotypes. It forces the average of the pairwise difference variables to be at least $\frac{4\ell-5}{4\ell-4}$.

For example, here is a case for $\ell = 4$:

$$g_i[k_1, k_2, k_3, k] = 1110$$
$$g_{j_1}[k_1, k_2, k_3, k] = 0001$$
$$g_{j_2}[k_1, k_2, k_3, k] = 0021$$
$$g_{j_3}[k_1, k_2, k_3, k] = 0221.$$

From the argument above, this data pattern gives the valid cut:

$$\sum_{x=1}^{3} (d_{2i-1,2j_x-1} + d_{2i-1,2j_x} + d_{2i,2j_x-1} + d_{2i,2j_x}) \geq 11. \tag{22}$$

This constraint is stronger than the constraints for the subpatterns of size-3×3. By assigning all pairwise difference variables the fractional value $\frac{7}{8}$, all of the size-3×3 constraints will be satisfied. However, the size-4×4 constraint is violated by this fractional solution since $12 \cdot \frac{7}{8} < 11$.

These data patterns need not strictly occur as described. In particular, the final column k exists to show that each g_j has a heterozygous position; in fact, they need not all be in the same column.

We wish to find an algorithm to enumerate all cuts of this type for a given genotype matrix. Here, we show a graph formulation where maximal cliques in an incompatibility graph correspond to maximal data driven cuts.

All of the data driven cuts are relative to a specific reference genotype g, which serves the role of g_i in the constraint. For each g, we construct an incompatibility graph G_g as follows: The vertex set is all genotypes that can share a haplotype with g. Formally, $V(G_g) = \{g' : g'$ is heterozygous and $g'[j] = g[j]$ or $g'[j] = 1$ or $g[j] = 1$ for all positions $1 \leq j \leq m\}$. We connect two vertices g_1 and g_2 if they can both participate in one of our data-driven cuts. This is the case if the following two conditions hold:

- There exists a site j_1 where $g_1[j_1] = 0$ and $g_2[j_1] = 2$ or $g_1[j_1] = 2$ and $g_2[j_1] = 0$.
- There exists a site j_2, where $g[j_2] = 1$ and $g_1[j_2] = g_2[j_2] \neq 1$.

The first condition ensures that the two genotypes cannot share a common haplotype. The second condition guarantees that the two genotypes cannot both be partially explained by the two haplotypes for genotype g. They cannot both share the same haplotype because of the first condition and they cannot each be partially explained by different haplotypes since the two haplotypes for g disagree at site j_2, while both g_1 and g_2 require the same value at j_2.

Cliques in this incompatibility graph have the property that no pair of genotypes can share a haplotype and no two genotypes can simultaneously both share a haplotype with g. This is exactly the condition that was required for the data driven cuts and only maximal cliques need to be explored since the constraints for nonmaximal cliques will be strictly dominated by those for maximal cliques.

We can find all maximal data-driven cuts by building the incompatibility graph G_g for each genotype g and then enumerating all maximal cliques, using a heuristic clique enumeration algorithm, such as the Bron-Kerbosch algorithm [21].

For a given genotype, we can compute the vertices of the graph in $O(nm)$ time. We can find edges of the graph in $O(n^2m)$ time. So, all the graphs can be built in $O(n^3m)$ time. The time spent on the enumeration of the cliques may be very long. However, this is only likely when there are many maximal cliques, at which point these cuts will be most useful. We evaluate the effect of these maximal clique constraints in Section 6.4.

$$\begin{array}{rll}
\text{minimize} \sum_{i=1}^{|H_e|} x'_i + \sum_{i=1}^{2n} x_i, & \text{subject to} & \\
\sum_{w_{i,\rho}\in W_i} w_{i,\rho} + u_i = 1, & \text{for each genotype } g_i,\ 1 \le i \le n & (23)\\
x'_j \ge w_{i,(j,k)}, & \text{for all } 1 \le i \le n \text{ and each} & (24)\\
 & w_{i,(j,k)} \in W_i & \\
x'_k \ge w_{i,(j,k)}, & \text{for all } 1 \le i \le n \text{ and each} & (25)\\
 & w_{i,(j,k)} \in W_i,\ k \ne * & \\
y_{2i-1,k} + y_{2i,k} = g_i[k], & \text{for each } 1 \le i \le n \text{ and } 1 \le k \le m & (26)\\
y_{i,k} - y_{j,k} \le d_{i,j}, & \text{for each } 1 \le i,j \le 2n \text{ and } 1 \le k \le m & (27)\\
y_{j,k} - y_{i,k} \le d_{i,j}, & \text{for each } 1 \le i,j \le 2n \text{ and } 1 \le k \le m & (28)\\
y_{j,k} \le d'_{i,j}, & \text{if } h'_i[k] = 0, \text{ for each } 1 \le i \le |H_e|, & (29)\\
 & 1 \le j \le 2n,\ 1 \le k \le m & \\
1 - y_{j,k} \le d'_{i,j}, & \text{if } h'_i[k] = 1, \text{ for each } 1 \le i \le |H_e|, & (30)\\
 & 1 \le j \le 2n,\ 1 \le k \le m & \\
d'_{j,2i-1} \ge u_i, & \text{for all } 1 \le j \le |H_e| & (31)\\
 & \text{and all genotypes } g_i,\ 1 \le i \le n & \\
d'_{j,2i} \ge u_i, & \text{for all } 1 \le j \le |H_e| & (32)\\
 & \text{and all genotypes } g_i,\ 1 \le i \le n & \\
d'_{j,2i-1} \le 1 - w_{i,(j,k)}, & \text{for all } 1 \le i \le n, \text{ such that} & (33)\\
 & w_{i,(j,k)} \in W_i & \\
d'_{k,2i} \le 1 - w_{i,(j,k)}, & \text{for all } 1 \le i \le n, \text{ such that} & (34)\\
 & w_{i,(j,k)} \in W_i,\ k \ne * & \\
x_i \ge 2 - (K+i) + \sum_{j=1}^{K} d'_{j,i} + \sum_{j=1}^{i-1} d_{j,i}, & \text{for each } 1 \le i \le 2n & (35)\\
x_i, x'_i, y_{i,k}, d_{i,j}, d'_{i,j}, w_{i,\rho}, u_i \in \{0,1\}, & \text{for all } i,j \text{ and } k\ . &
\end{array}$$

Fig. 3. The HybridIP formulation for Haplotype Inference by Pure Parsimony.

5 A New Integer Linear Programming Formulation

The PolyIP formulation presented in Section 3 establishes a nice theoretical result. However, although we are able to solve many problem instances with it, the experimental results presented in Section 6 show that, when the RTIP formulation can solve the problem, it is much faster.

Thus, we seek an integer programming formulation with both practical size and reasonable runtimes. Here, we satisfy this goal with a formulation that draws upon ideas from both the RTIP and the PolyIP. We call this IP the HybridIP.

The RTIP formulation's strength is that many genotypes have few explaining pairs. However, large problems may have some genotypes with billions of explaining pairs, so the IP is too huge to use. In our HybridIP, we expand only a few possible explaining parents and allow parents that were not among those expanded.

5.1 Construction of the HybridIP

As in the PolyIP, every genotype g_i is explained by two haplotypes, h_{2i-1} and h_{2i}. These haplotypes are expanded in the decision variables: $h_i[k]$ is represented by decision variable $y_{i,k}$; (26) requires that haplotypes properly explain their genotypes.

Unlike for the PolyIP, we also choose a set $H_e = \{h'_1, \dots, h'_K\}$ of K specific possible haplotypes. We can explain genotype g_i using zero, one, or two haplotypes from H_e. For every explaining pair (h'_j, h'_k) for genotype g_i, where both h'_j and h'_k are in H_e, we create a variable $w_{i,(j,k)}$, as in the RTIP formulation. If there is an explaining pair (h'_j, η), where h'_j is in H_e, but η is not, we create a variable $w_{i,(j,*)}$. If there is an explaining pair (η_1, η_2) for g_i where both haplotypes are not in H_e, we create a variable u_i; this variable will be set to 1 when the chosen explaining pair for g_i uses no haplotypes from H_e. Let W_i be the set of all $w_{i,(j,k)}$ and $w_{i,(j,*)}$ variables. The selection constraint (23) mandates that we explain all genotypes, either through expanded or partially expanded parent pairs (corresponding to one or two haplotypes from H_e) or through two parents not from H_e.

We use d and d' variables to mark differences between haplotypes; the $d_{a,b}$ variables, set by (27) and (28), register differences between the explicitly enumerated haplotypes, as for the PolyIP. The $d'_{a,b}$ variables, set by (29) and (30), identify whether the haplotype h_b, described by the vector of $y_{b,i}$ variables, equals the preselected haplotype choice h'_a from H_e. If we explain genotype g_i by setting $w_{i,\rho} = 1$, (33) and (34) ensure that the haplotypes are properly set. Meanwhile, if we choose an explaining pair neither of whose members are in H_e, (31) and (32) ensure both haplotypes are kept different from all haplotypes in H_e.

As before, we calculate how many unique haplotypes are used by a solution. As in Gusfield's TIP formulation, we have one decision variable, x'_j, for all h'_j in our specified set of haplotypes H_e, set to 1 if haplotype h'_j is ever chosen. This is enforced by (24) and (25). We also must count the unique haplotypes not from H_e. As in the PolyIP, for each haplotype vector, we create a variable x_i, equal to 1 if h_i is not in H_e and is unique in $(h_1, \dots, h_i)$; (35) enforces this. Our objective of minimizing the number of unique haplotypes used is equivalent to minimizing the sum of the x' and x variables. The complete HybridIP is in Fig. 3.

If K, the size of H_e, is a constant or even polynomial in the size of the input, the program is polynomial in size and can be created in polynomial time. If K is zero, the program is essentially the same as the PolyIP. If every possible explaining haplotype is included in H_e, the program is analogous to Gusfield's TIP formulation [1], but with many added variables.

As with the PolyIP, we again introduce small positive perturbations to the coefficients of the d and d' variables in the objective function. We also introduce similar small random perturbations for the w variables. Finally, we introduce a slightly larger positive perturbation to the coefficients for the u_i variables. This causes the program to favor selecting from the expanded set H_e. We justify this bias by the assumption that the algorithm used to populate the set H_e chooses haplotypes that are more likely to be in the optimal solution.

Many possible approaches exist to populate the set H_e of prespecified haplotypes. We populated the set of haplotypes to be partially expanded by sequentially expanding parents of the genotypes starting with those with fewest heterozygous sites, until the number of unique haplotypes expanded reached a cutoff K. This set completely determines which $w_{i,\rho}$ variables were in the formulation. When generating the IP, we resolved $w_{i,(j,*)}$ to get rid of the unspecified parents (identified by asterisks) by adding the appropriate haplotype to the expansion set. We used this new haplotype only to resolve the unspecified parents and did not introduce any new $w_{i,\rho}$ variables. This resolution appears to give a stronger formulation and remains polynomial in size. In Section 6.2, we evaluate the effect of varying the value of K. Although there is no clear best choice, we find that a value of $K = 24$ yields good results.

All of the valid cuts shown in Section 4 are also valid for the HybridIP, giving an additional way to improve the speed of solving the HIPP problem through integer programming.

6 Experiments

The HIPP problem is both an abstraction of a real biologically motivated problem and a discrete optimization problem with interesting structure. It is natural to empirically analyze the performance of any new proposed method to solve HIPP. Our work was motivated by two goals: to eliminate the exponential dependency of the RTIP and to achieve similar runtime to that of the RTIP on the instances where the RTIP can be formulated. With these goals in mind, we will use the performance of the RTIP as a baseline for comparison.

We wish to emphasize that our experiments are simply to evaluate the runtime performance of these IP techniques for solving HIPP. We neither attempt to evaluate the effectiveness of HIPP for haplotype inference nor compare our algorithm to haplotype inference solvers that do not use the parsimony objective; this analysis has previous been performed by both Gusfield [1] and Wang and Xu [15].

6.1 Implementations

We created a branch-and-cut prototype implementation of the PolyIP to solve instances of the HIPP problem. We used CPLEX 8.1.1 [22] to solve the LP relaxations and then added cuts or set branched variables directly. Commercial LP solvers provide powerful IP solvers that may have some success solving this problem. However, implementing the simple branch-and-cut algorithm allows us to explore different branching rules and cuts.

In the PolyIP, we branched only on the pairwise difference variables $d_{i,j}$ because they have strong influence on the IP. If all $d_{i,j}$ variables are integral and all transitivity constraints are satisfied, there is an integral solution to the problem with the same objective value. In practice, when the $d_{i,j}$ variables solved to integer values, all other variables were assigned integer variables by the solver. Setting a $d_{i,j}$ variable to 0 is very restrictive: It forces the corresponding parents to be equal. However, there is still flexibility in the actual values of those parents.

Unless otherwise stated, the data driven cuts described in Section 4 that are used in the tests were only those derived from two-by-two patterns in the genotype matrix. We added transitivity constraints when they were violated by the solution to the LP relaxation.

We also created a branch-and-cut prototype implementation of the HybridIP, using CPLEX 8.1.1 to solve the LP relaxations. In the HybridIP, we first branched on the selection variables $w_{i,\rho}$. Then, if we had still not found an integer solution, we branched on the difference variables $d_{i,j}$ and $d'_{i,j}$. This branching rule was used because assigning a selection variables $w_{i,\rho}$ to 1 completely assigns the haplotype vectors for the explaining pair for genotype g_i. We also had assumed that our partially expanded set of haplotypes is likely to contain many of the correct ones.

The HybridIP used the same cuts as the PolyIP, although generally fewer cuts were added while solving the HybridIP.

We implemented a prototype of Gusfield's RTIP formulation [1] to compare the effectiveness of our IPs with his. This implementation used the same simple branch-and-cut framework, although without generating any cuts.

All tests were run on a 1.5GHz Intel Pentium 4 with 1GB RAM running Debian Linux. Our prototype implementations are available by request. Unless otherwise stated, the tests were run with a two hour time limit.

As a preprocessing step, we eliminate duplicate copies of genotypes from the input, assuming they will all be phased identically. This reduction obviously does not change the value of the optimal solution. It is worth noting that allowing the duplicate haplotypes will often increase the gap in between the optimal solution to the IP and its relaxation since this will weaken the effectiveness of (7) in the LP relaxation of the PolyIP. This reduction was performed for all prototype implementations and all tests.

We also implemented an optional preprocessing step that eliminated duplicate columns and complimentary columns. Again, this is an obvious reduction that can be applied to reduce the problem size and not change the value of the optimal solution.

6.2 Simulated Data Sets

We present experimental results using two different simulated methods of data generation. We created parent haplotypes using Hudson's program `ms` [23], which

TABLE 1
Experimental Comparison of Three IP Formulations

# of Sites	Recombination Level	Max # of 1s in a genotype	Mean # of 1s in a genotype	Fraction Solved PolyIP	RTIP	HybridIP	Min # of Branches	Max # of Branches
10	0	8	3.1	$15/15$	$15/15$	$15/15$	1	1
10	4	9	3.3	$15/15$	$15/15$	$15/15$	1	2
10	16	9	2.9	$12/15$	$15/15$	$15/15$	1	53
30	0	23	8.2	$11/15$	$7/15$	$15/15$	1	29
50	0	42	14.1	$27/50$	$0/50$	$35/50$	1	831
75	0	52	19.8	$4/10$	$0/10$	$6/10$	1	484
100	0	68	25.1	$3/10$	$0/10$	$3/10$	1	271

Tests with sequences of length 30 or less had 50 population members, while the tests of length 50 or greater had only 30 population members. For large instances, the number of IP branches required by the HybridIP ranged widely.

simulates neutral evolution and recombination. We then paired the resultant haplotypes randomly to produce offspring populations. The distinction in the two simulated method comes in how the random pairing is performed. In one case, we randomly pair sampling uniformly from the set of distinct haplotypes. In the second case, we sample uniformly from the collection of haplotypes generated by the coalescent process. In this collection, haplotypes may not be unique, so some haplotypes are sampled with higher frequency than others.

We chose to randomly pair uniformly from the set of unique haplotypes since we believed this would lead to more difficult problem instances. This is because the probability of generating a completely homozygous population member is reduced. Also, the number of duplicate population members should be reduced. With respect to the RTIP formulation, this does generate more difficult instances as the IP instances are larger. However, it also has the effect of reducing the presence of rare haplotypes, which we discovered may make for slightly easier instances of the PolyIP and HybridIP. We discuss these issues in further detail below.

For each of the simulated data generation methods, we generated two classes of data sets. First, we ran experiments on inputs of sizes comparable to those used by Gusfield [1]. The data consist of collections of $n = 50$ offspring with haplotypes of length either $m = 10$ or 30. The second class of inputs are designed to be problem instances that should be too large to formulate in the exponential formulation. In these cases, we use collections of 30 offspring with haplotypes of length 50, 75, and 100. It turned out that there was always at least one population member with only zero or one heterozygous sites in all of our experiments.

6.2.1 Problem Results: Distinct Haplotypes

A summary of the findings for the distinct haplotype data set is in Table 1. For shorter sequences, the HybridIP was able to solve all the instances. For the length 10 sequences, all but two instances solved in less than seven seconds, which is very close to the speed of the RTIP and a great improvement over the slow times of the PolyIP.

On the length 30 sequences, six instances solved in less than 30 seconds, and 12 in less than two minutes. The final three instances all took longer than five minutes, with the longest requiring just over an hour. By contrast, for the seven RTIP instances that were of reasonable size from this data set, all solved in fewer than five seconds, but the other eight were too large to produce. Eight of the HybridIP instances solved on the first branch.

On longer sequences, the HybridIP was able to match and, in the case of 50 and 75, exceed the number of instances solved by the PolyIP. For sequences of length 50 and 75, the HybridIP solved at least 60 percent of the instances. For the length 100 sequences, the HybridIP matched the result of the PolyIP.

All of the runtimes of the HybridIP were much faster than for the PolyIP. Many length 50 instances solved in under five seconds, most within 10 minutes, and all within two hours. For length 75 sequences, two instances solved in less than a minute, while others took more than two hours. Two of the three length 100 instances solved in less than two minutes; the other took just under an hour.

6.2.2 Nonuniform Haplotype Frequencies

Uniformly sampling among the unique haplotypes to generate the genotypes is not the standard method of generating simulated genotype populations. For this reason, we also evaluated the three different IP formulations on data with nonuniform haplotype frequencies. In these data sets, $2n$ haplotypes were generated using Hudson's program `ms` [23], then n genotypes were generated by pairing sampling from the $2n$ haplotypes with replacement. Within the $2n$ haplotypes, there may be multiple copies of the same haplotype, giving rise to the nonuniform frequencies. As before, the sampling is done with replacement, so the same haplotype may be chosen as a parent of multiple genotypes or even twice for the same genotype. We again found that every test set included at least one population member with only zero or one heterozygous site.

The results for the nonuniform sampling data set are presented in Table 2. There are significant differences between these results and those presented in Table 1. On the nonuniform data, the PolyIP was unable to solve even all of the length 10 instances and only solved about half of the length 30 instances. All of the length 30 instances took longer than $1,000$ seconds. The PolyIP was unable to solve any of the length 50 or greater problem instances within a two hour time limit.

We believe this difference is explained by very low frequency haplotypes. The genotypes resulting from these haplotypes can often be resolved in many ways without affecting the value of an optimal solution. Around each of

TABLE 2
Experimental Comparison of Three IP Formulations

# of Sites	Max # of 1s in a genotype	Mean # of 1s in a genotype	Fraction Solved PolyIP	RTIP	HybridIP	Min # of Branches	Max # of Branches
10	7	1.7	14/15	15/15	15/15	1	4
30	18	5.0	8/15	15/15	15/15	1	617
50	33	8.4	0/15	8/15	6/15	1	6
75	41	15.8	0/15	2/15	5/15	2	233
100	66	19.1	0/15	0/15	3/15	11	127

Tests with sequences of length 50 *or less had* 50 *population members, while the tests of length* 75 *and* 100 *had only* 30 *population members.*

these equivalent optimal solutions, we suspect that there are many fractional solutions that the branch and bound algorithm explores, leading to lengthy runtimes.

In contrast, these same genotypes formed from low frequency haplotypes benefit the RTIP formulation since they yield fewer expanded haplotypes in the RTIP expansion. On the nonuniform data set, the RTIP formulation solved all length 10 and 30 instances and all in one second. This is a significant improvement over the results on the uniform data set, where fewer than half of the length 30 instances solved with the RTIP. A more dramatic improvement is in the longer sequences, where the RTIP was able to solve over half of the length 50 instances and two out of the 15 length 75 instances. Four of the length 50 instances solved in less than one and a half minutes. One of the length 75 instances was solved in 529 seconds, while the other came in just under the two hour time limit at $6,985$ seconds.

Similarly to the PolyIP, the HybridIP performance worsened on the nonuniform data. However, the HybridIP seems less sensitive to this change than the PolyIP. The HybridIP solved all of the length 10 and 30 instances. All of the length 10 instances solved in one second and 10 out of the 15 length 30 instances solved in less than 15 seconds. The HybridIP was only able to solve six length 50 instances, which is slightly worse than the RTIP. However, all six instances solved in 39 seconds or less. The Hybrid solved five of the 15 length 75 instances, with two of the instances solving in less than one minute. The HybridIP was the only formulation able to solve a length 100 instance. It solved three length 100 instances, with two of those instances solved in under two minutes.

In this data set, we cannot see any significant difference between the number of instances solved between the RTIP and the HybridIP. However, on the longer sequences, the runtimes of the HybridIP seem much improved over those of the RTIP. It is worth noting that the majority of the time spent in the RTIP is on formulating the instance. A more clever implementation may be able to overcome some of the problems due to large problem instances.

6.2.3 *Parameters of the HybridIP*

The performance of the HybridIP is very sensitive to the choice of the parameter K, the size of the set H_e. For some IPs, selecting a very large K yields a very quick solution, while, for others, choosing a large K greatly increases the time required to solve the LP relaxation and increases the number of branches needed to find the optimal solution. We performed three tests to evaluate values of K. In each test, we ran the HybridIP on 15 inputs with a time limit of five minutes for each instance. Two of the tests evaluated genotypes of length 30 and one test was on length 50 sequences.

Table 3 shows the results on sequences of length 30 with uniform haplotype distribution. This test shows a general trend that expanding more haplotypes causes more instances to be solved. However, besides the one additional test case solved for values 48 and 64, there is little variation among all the nonzero values.

Table 4 evaluates the same parameters on the instances of length 30 with nonuniform haplotype frequencies. Here, the value of 48 still solved the most instances, but both the values of 32 and 64 solved one fewer instance than the value of 24. In this case, there is no clear trend toward larger being better.

Finally, in Table 5, we see the results on sequences of length 50, with uniform haplotype distribution. Here, we see that, on the longer sequences, the smaller number of expanded haplotypes seem to be giving much faster runtimes. The larger values of 48 and 64 had fewer instances solved in under 10 seconds. The value 64 solved the most instances in under five minutes.

Overall, there is no clear best choice for the value of the parameter K. The values tested were all in a very close range and yet there was a fair amount of variation as to how the HybridIP performed. We choose to use the value of 24 for evaluating the HybridIP in our other tests. Experiments with this value of K consistently solved very quickly in our tests and, in the absence of a clear best value, we choose to go with the smallest value that performed well in our tests since each additional expanded haplotype introduces a reasonably large number of variables.

TABLE 3
The Effect of the Number of Haplotypes Expanded on the Performance of the HybridIP

# of haplotypes expanded	Instances solved in time (sec): <10	10-30	30-60	60-300	Total Solved
0 (the PolyIP)	0	0	0	0	0/15
16	0	4	1	7	12/15
24	1	5	2	4	12/15
32	1	5	2	4	12/15
48	2	4	3	4	13/15
64	1	5	3	4	13/15

This table shows the number of instances of length 30 *with uniform haplotype frequencies solved within in each time interval. Small values of* K *show great improvement over setting* K *to zero, which is equivalent to the PolyIP.*

TABLE 4
The Effect of the Number of Haplotypes Expanded on the Performance of the HybridIP

# of haplotypes expanded	Instances solved in time(s): <10	10-30	30-60	≥ 60	Total Solved
0 (the PolyIP)	0	0	0	1	1/15
16	8	1	0	0	9/15
24	8	3	1	0	12/15
32	7	4	0	0	11/15
48	8	3	1	1	13/15
64	8	2	1	0	11/15

This table shows the number of instances of length 30 *with nonuniform haplotype frequencies solved within in each time interval.*

TABLE 5
The Effect of the Number of Haplotypes Expanded on the Performance of the HybridIP

# of haplotypes expanded	Instances solved in time(s): < 10	10-30	30-60	$\geq$60	Total Solved
0 (the PolyIP)	0	0	0	2	2/15
16	0	5	0	1	6/15
24	4	5	1	0	10/15
32	5	3	1	0	9/15
48	1	5	2	2	10/15
64	2	6	0	4	12/15

This table shows the number of instances of length 50 *with uniform haplotype frequencies solved within in each time interval.*

Initially, it may seem surprising that expanding such a small number of haplotypes (such as 16 or 24) makes such a large difference. Still, the haplotypes that are being expanded are from genotypes with very few heterozygous sites. This implies that the expanded set is guaranteed to contain many of the haplotypes found in the optimal solution. We also benefit when the u_i variable is selected since all the d' difference variable to the expanded haplotypes are forced up to one. Overall, this greatly reduces the number of fractional d' values and strengthens (35) in the HybridIP.

6.3 Biological Data Sets

With the completion of the first phase of the HapMap project [24], there is now a large source of publicly available genotype data with which to evaluate haplotype inference programs. We considered inputs from chromosomes 10 and 21, over all four HapMap populations. For each input length we considered, we selected a continuous collection of SNPs with the highest minimum pairwise D' value, a measure of linkage disequilibrium. This was meant to find regions of the desired length with small amounts of recombination.

We tested sequences of lengths 30, 50, and 75, giving a total of 24 instances. Each instance was reduced to contain a unique set of genotypes, so the actual number of genotypes in each sample varies significantly, generally increasing with sequence length. Of the 24 instances, the PolyIP was able to solve 15, the HybridIP was also able to solve 15 and none of the RTIP formulations could be constructed within the two hour time limit. Five of the nine failing HybridIP instances were all from the same HapMap population, the Yoruba in Ibadan, Nigeria [24]. Looking at the instances that were generated for all four populations, it was clear that there were a significant number of duplicate columns that caused the RTIP instance to explode in size.

To attempt to more fairly compare the RTIP and the other IP formulations, we ran the tests again with duplicate and complement columns removed, under the assumption they will be phased in the same manner as the representative column. Under this column reduction scheme, the RTIP was able to solve 21 of the 24 instances. The HybridIP was able to solve 16 of the 24 instances and the PolyIP was able to solve 17 instances. All of the RTIP instances that failed were length 75 instances, which agrees with our hypothesis that the RTIP formulation cannot solve long instances well. Two of the instances that the RTIP could not solve were solved by the HybridIP and the PolyIP, even without column reduction. For all three formulations, the column reduced instances generally solved very quickly, with the exception of the length 75 sequences. It is surprising that the PolyIP performed as well as or better than the HybridIP on these tests.

Interestingly, on the real genotype instances, six of the RTIP instances required branching to find the optimal solution. This branching never occurred on our synthetic data, where the LP-relaxations always solve to integral values. One instance required 67 branches.

Without column reduction, the HybridIP and the PolyIP perform better than the RTIP. With the column reduction, the RTIP solves significantly more instances than the HybridIP or the PolyIP. We note that, since adding a single genotype to the instances could cause the column reduction to no longer be applicable, it may not be desirable to depend on tricks like this column reduction.

6.4 Maximal Clique Enumeration

All of the previous tests were run without the maximal clique enumeration for the data driven cuts described in Section 4. To evaluate the effect of these cuts, we looked at the effect of adding the maximal clique enumeration to the branch-and-cut algorithm for the PolyIP.

Running the PolyIP on the biological data sets without column reduction led to surprising results. First, the clique enumerations all took less than one second, so this process did not affect runtime significantly. However, on almost every test, adding the maximal clique constraints caused slower solve times. The tests that were not slower solved within three seconds. One of the tests that solved within six minutes without all of the maximal clique constraints did not manage to solve within the two hour time limit when the clique constraints were added.

The number of cuts enumerated had a wide range, but did depend on the lengths of the sequences. The maximum number of cuts enumerated in a test was $8,142$, in one of the length 75 biological data tests. It is interesting to note that, on that test, $7,401$ data driven cuts were violated in the initial solution of the PolyIP LP relaxation. However, within the two hour time limit, the second solution to the LP relaxation was never found.

Looking more closely at the test runs, we observe that adding the maximal clique constraints generally reduced the number of branches required to find the optimal solution significantly. However, each solution to the LP relaxation took longer to find and, overall, reduction in branches did no make up for the increased solve time.

We observe similar degradation to the runtimes for some sample tests run with the synthetic data. It seems that adding every violated cut leads to greatly increased work when solving the LP relaxations. It is likely that a more advanced branch-and-cut could use these maximal clique cuts in a more advantageous way.

7 Variations on HIPP and Some More IP Cuts

The HIPP problem can be expanded to accommodate a number of possible variations. These variations include different scenarios of error in the input genotypes or allow representation of known familial relationships among members of the input population. Or, they may be appropriate due to instrument limitations. Kalpakis and Namjoshi [7] presented a number of variations and showed how their semidefinite programming formulation of HIPP could be extended to accommodate them. Here, we show that the HybridIP can easily expand to include these, as well; we do not offer experimental evidence of the speed of solution in part because it is challenging to envision the proper parameters for such experiments.

In addition, we provide one additional type of valid cut for the PolyIP. Although we can show that there are instances where using this cut may help close the gap between the IP and its LP relaxation, we do not believe these cuts would improve our runtimes without discovering an efficient separation algorithm.

7.1 Various Forms of Error

The input genotype matrix G may not be accurate. It could be incomplete, with missing entries. It could have a small number of mistakes. Or, there might be a small number of genotypes with many errors. All three of these scenarios can be adapted to our HybridIP. In the first, where the genotype matrix is incomplete, assume that genotype site $g_i[k]$ is unknown. We expand the set W_i of possible explaining pairs for g_i to include explaining pairs for all possible choices for $g_i[k]$ and we remove (26) of the HybridIP for $g_i[k]$.

In the second scenario, where the input genotypes G are complete, but there is a bound E on the total number of errors in all of them, we can adapt by creating deviation variables $e_{i,k}$, which are integers between -2 and 2, that account for the difference between the value of $g_i[k]$ in the input and the true underlying value. Then, we modify (26) to:

$$y_{2i-1,k} + y_{2i,k} = g_i[k] + e_{i,k} \quad \text{for each } i \text{ and } k. \tag{36}$$

To constrain the system so there are no more than E errors in the genotype matrix, we add integer variables $e'_{i,k}$ that are 1 when $e_{i,k}$ is nonzero and restrict the maximum number of edited sites. These requirements are enforced by these constraints:

$$e'_{i,k} \geq e_{i,k}/2, \qquad \text{for each } i \text{ and } k, \tag{37}$$

$$e'_{i,k} \geq -e_{i,k}/2, \qquad \text{for each } i \text{ and } k, \tag{38}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{m} e'_{i,k} \leq E. \tag{39}$$

We may now explicitly enumerate a haplotype h_j from H_e while still not choosing to set $w_{i,\rho}$ to 1 for any of the previously enumerated explanations for genotype g_i. This would result in not setting x'_j to 1, giving an incorrect answer to the problem. To avoid this, we must add the constraint:

$$x'_j \geq 1 - d'_{i,j}, \quad \text{for each } i \text{ and } j. \tag{40}$$

In the third scenario, where there is a bound E on the number of genotypes that possess errors, but where the number of errors in a particular genotype is irrelevant, we use the $e_{i,k}$ variables as before, but then create variables e_i to identify whether genotype g_i was edited; this is done by requiring that e_i is 1 exactly when any of the $e_{i,k}$ are nonzero in a way analogous to (36); then, we limit the sum of the e_i to be at most E. We must again add (40) to ensure we pay for all chosen haplotypes.

7.2 Shared Haplotypes

External information we may have may tell us that two different genotypes share a common haplotype. For example, we can conclude this if we know that the two genotypes are parent and child and the genotypes have no recombinations or mutations. If genotypes g_i and g_j are known to share a haplotype, we can add this to the HybridIP by adding the constraint:

$$d_{2i-1,2j-1} + d_{2i,2j-1} + d_{2i-1,2j} + d_{2i,2j} = 3. \tag{41}$$

7.3 XOR Genotypes

Barzuza et al. [25] proposed the study of haplotype inference on a new type of data called XOR-genotypes. Some experimental methods can only determine if a site is heterozygous of homozygous, but not which homozygous allele. In this domain, genotype g can be explained by haplotypes h_1 and h_2 if $g = h_1 + h_2 \bmod 2$.

If all data are of this form, the HybridIP is inappropriate; any possible haplotype sequence h is a potential explaining parent for genotype g since the other explaining parent is then $g - h \bmod 2$. Still, the PolyIP is appropriate; we replace (4) with the constraints:

$$y_{2i-1,k} + y_{2i,k} = 1, \text{ if } g_i[k] \text{ is heterozygous}, \tag{42}$$

$$y_{2i-1,k} = y_{2i,k}, \qquad \text{if } g_i[k] \text{ is homozygous}. \tag{43}$$

7.4 Perfect Phylogeny

A final variation is to require that the solution satisfy a perfect phylogeny. This specialization of HIPP, called Min-PPH, has been shown to be NP-Hard by Bafna et al. [26]. Here, we show how to add new variables and constraints to the IP to ensure that the solution satisfies a perfect phylogeny.

The classical characterization of a PPH solution is that the haplotype matrix passes the *four gamete test*. That is to say that there is no 4×2 submatrix that contains all four possible binary strings of length 2. One could attempt to introduce a condition for each 4×2 submatrix that ensures the four gametes do not occur. However, this would introduce far too many variables.

Bafna et al. [27] proved an alternative characterization of PPH solutions that we can exploit to reduce the number of variables required. Following the notation of Bafna et al., we define two columns k and l to be *companion columns* if there exists a genotype g_i that is heterozygous in columns k and k, so $g_i[k] = g_i[l] = 1$. For each pair of companion columns k and l, we introduce a new binary variable $\mathcal{E}_{k,l}$. If $\mathcal{E}_{k,l} = 1$, then we say that columns k and l *equate* and, for all genotypes g_i such that $g_i[k] = g_i[l] = 1$, $h_{2i-1,k} = h_{2i-1,k}$. If $\mathcal{E}_{k,l} = 0$, then we say that columns k and l *negate* and, for all genotypes g_i such that $g_i[k] = g_i[l] = 1$, $h_{2i-1,k} = 1 - h_{2i-1,k}$.

Using the product variables described in Section 4.1, can model the exclusive-or operator $\oplus$, using integer linear constraints. Using the same method we can add the following constraints for each pair of companion columns $1 \leq k < l \leq m$ and each genotype g_i such that $g_i[k] = g_i[l] = 1$:

$$\mathcal{E}_{k,l} = h_{2i-1,k} \oplus h_{2i-1,l}. \tag{44}$$

To complete the categorization of Bafna et al. [27], we need to encode two types of constraints. We start with the easier of these, which is for triplets of columns. Bafna et al. showed that, for three columns k_1, k_2, and k_3, such that, for some genotype g_i, $g_i[k_1] = g_i[k_2] = g_i[k_3] = 1$, if the haplotypes satisfy a perfect phylogeny, it must be the case that

$$\mathcal{E}_{k_1,k_2} = \mathcal{E}_{k_1,k_3} \oplus \mathcal{E}_{k_2,k_3}. \tag{45}$$

Consider two companion columns k and l. Each genotype that is not heterozygous in both positions has a unique expansion, relative to those two columns. If that expansion contains the rows 00 and 11 in these two columns, then the columns must be equated. Similarly, if that expansion contains the rows 01 and 10, then the columns must negate. For all column pairs (k, l) with such a *forcing pattern*, the value of $\mathcal{E}_{k,l}$ is known, if the haplotypes are to satisfy a perfect phylogeny, and we should replace them by their values in the formulation. Finding the forcing pairs can be completed in $O(nm^2)$.

We know by the results of Bafna et al. [27] that, if our solution meets these conditions, it satisfies a perfect phylogeny. In the worst case, this uses $O(m^2)$ indicator variables, $O(m^3)$ product variables for the triplet constraints (45) and $O(nm^2)$ product variables for (44). This also adds $O(m^3 + nm^2)$ constraints to the system. This formulation is rather large and is unlikely to yield practical runtimes.

7.5 Permutation Constraints

One drawback of the PolyIP is that its feasible region depends on the order of the genotypes. Under the LP relaxation, the $d_{i,j}$ variables may take fractional values. If two such variables in the same constraint take value 0.5, the x_i variable can be set to 0. Under a different ordering, it might be possible that more x_i variables are properly assigned the value 1 under the relaxation.

This order dependency introduces an interesting class of permutation constraints. In a second instance of the problem with a different genotype order, the underlying problem would still be the same, so the optimal objective function value would be, too. Conceptually, the permutation constraints combine these two programs such that each has its own x_i variables, but all other variables are shared. We then add the constraint that the two objective function values are equal.

Let π be a permutation of $\{1, \dots, 2n\}$. Define new variables $x_1^\pi, \dots, x_{2n}^\pi$ and add constraints analogous to those defined for the x_i variables. Let $d_{i,j} = d_{j,i}$ if $i > j$. Then, for $1 \leq i \leq 2n$, we have:

$$x_i^\pi \geq 2 - i + \sum_{j=1}^{j<i} d_{\pi(j),\pi(i)}. \tag{46}$$

These constraints are equivalent to (7), only with a different ordering. For integer points, these x_i^π count the number of unique haplotypes used, just as before. Therefore, the additional valid cut can be added:

$$\sum_{i=1}^{2n} x_i = \sum_{i=1}^{2n} x_i^\pi, \tag{47}$$

which forces the number of unique haplotypes for both orders to be equal.

If the sum constraint (47) is violated, we arbitrarily increase the needed x variables. This will not change any of the other types of variables in the formulation. We would like to have the property that if $x_i = 1$, then, for all j less than i, $d_{i,j} = 1$, and similarly for x_i^π. This can be forced by adding this constraint for all $j < i$:

$$x_i^\pi \leq d_{\pi(j),\pi(i)}. \tag{48}$$

Consider the following genotype matrix:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

If we resolve the first ambiguous position in each genotype, the optimal solution to the LP will be:

$$\begin{array}{c} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \end{array} \begin{bmatrix} 0 & 0.5 & 0.5 \\ 1 & 0.5 & 0.5 \\ 0 & 0 & 0.5 \\ 0 & 1 & 0.5 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

for which the value of the LP objective function $\sum_{i=1}^{2n} x_i$ will be 3.5. However, if we permute the haplotypes to the following matrix:

$$\begin{array}{c} h_2 \\ h_4 \\ h_5 \\ h_6 \\ h_1 \\ h_3 \end{array} \begin{bmatrix} 1 & 0.5 & 0.5 \\ 0 & 1 & 0.5 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix},$$

for which the objective function will take value 4, which is the optimal value. This example shows that there are some permutations that are superior to others. If one were to start with the bad permutation, it would be beneficial to add the new cut. However, simply using the better ordering in the first place would have worked as well. During the branching process, we believe there will be cases when adding a permutation constraint could be helpful. We have not identified an efficient separation algorithm and, since there are $(2n)!$ possible permutations, these cuts do not seem of practical use. It is likely the case that the most benefit would be achieved by picking the strongest permutation when constructing the IP formulation. Unfortunately, we are not aware of any algorithm for determining the optimal permutation. If the initial permutation is not the strongest, it is useful to be able to add these new variables and constraints to the system during the branching process without having to abandon all the work already done.

8 Conclusions

Pure parsimony has been shown to be a reasonable objective function for the haplotype inference problem [1], [15]. All previously evaluated exact approaches to the HIPP problem use a formulation that involves either explicitly or implicitly expanding out possible parents, leading to exponential problem size.

Instead, we have given two polynomial-sized IP formulations of the HIPP problem. Our first approach is different from previous IP approaches to this problem in that we do not expand out all explaining parents. In addition, we have shown how to strengthen the formulation using valid cuts derived from properties of the input genotypes. Our experiments showed that, although our PolyIP formulation takes longer to solve, we can solve problems the same size as previous formulations. Our PolyIP formulation is also capable of solving problem instances significantly larger than previous formulations.

Our second IP, the HybridIP, is a polynomial-sized formulation which outperforms both the RTIP formulation and the PolyIP formulation. The formulation is based on expanding a small set of possible explaining haplotypes, as in the RTIP formulation, while also allowing the choice of haplotypes that have not been expanded. By completely expanding genotypes with few heterozygous sites, we know that some of the optimal haplotypes are in the expanded set. On the vast majority of tests, the HybridIP was able to solve more instances than the other two approaches. Although the runtimes were not as short as the runtimes for RTIP instances that solved, the runtimes for the HybridIP were significantly quicker then those of the PolyIP.

Acknowledgments

This research has been supported by the Natural Science and Engineering Research Council of Canada through a Discovery Grant to Daniel Brown and both a Postgraduate Scholarship and a Canada Graduate Scholarship to Ian Harrower. The authors would like to thank Dan Gusfield and Nathan Edwards for several interesting discussions of this problem and the anonymous referees for useful comments. Preliminary work on the PolyIP was published in the *Proceedings of WABI 2004* [16].

References

[1] D. Gusfield, “Haplotype Inference by Pure Parsimony,” *Proc. 14th Ann. Symp. Combinatorial Pattern Matching,* pp. 144-155, 2003.

[2] B.V. Halldórsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph, and S. Istrail, “A Survey of Computational Methods for Determining Haplotypes,” *Computational Methods for SNPs and Haplotype Inference: Proc. DIMACS/RECOMB Satellite Workshop,* pp. 26-47, 2004.

[3] G. Lancia, C.M. Pinotti, and R. Rizzi, “Haplotyping Populations by Pure Parsimony: Complexity of Exact and Approximation Algorithms,” *INFORMS J. Computing,* vol. 16, no. 4, pp. 348-359, 2004.

[4] A.G. Clark, K.M. Weiss, D.A. Nickerson, S.L. Taylor, A. Buchanan, J. Stengard, V. Salomaa, E. Vartiainen, M. Perola, E. Boerwinkle, and C.F. Sing, “Haplotype Structure and Population Genetic Inferences from Nucleotide-Sequence Variation in Human Lipoprotein Lipase,” *Am. J. Human Genetics,* vol. 63, no. 2, pp. 595-612, 1998.

[5] D. Gusfield, “Inference of Haplotypes from Samples of Diploid Populations: Complexity and Algorithms,” *J. Computational Biology,* vol. 8, no. 3, pp. 305-313. 2001.

[6] N. Edwards, Personal communication to D. Brown, Aug. 2004.

[7] K. Kalpakis and P. Namjoshi, “Haplotype Phasing Using Semidefinite Programming,” *Proc. Fifth IEEE Int’l Symp. Bioinformatics and Bioeng.,* pp. 145-152, 2005.

[8] J. He and A. Zelikovsky, “Linear Reduction for Haplotype Inference,” *Proc. Fourth Ann. Workshop Algorithms in Bioinformatics,* pp. 242-253, 2004.

[9] M.J. Daly, J.D. Rioux, S.F. Schaffner, T.J. Hudson, and E.S. Lander, “High-Resolution Haplotype Structure in the Human Genome,” *Nature Genetics,* vol. 29, no. 2, pp. 229-232, 2001.

[10] N. Patil et al., “Blocks of Limited Haplotype Diversity Revealed by High-Resolution Scanning of Human Chromosome 21,” *Science,* vol. 294, no. 5547, pp. 1719-1723, 2001.

[11] A.G. Clark, “Inference of Haplotypes from PCR-Amplified Samples of Diploid Populations,” *Molecular Biology and Evolution,* vol. 7, no. 2, pp. 111-112, 1990.

[12] K. Lindblad-Toh et al., “Genome Sequence, Comparative Analysis and Haplotype Structure of the Domestic Dog,” *Nature,* vol. 438, no. 7069, pp. 803-809, 2005.

[13] D.A. Hinds, L.L. Stuve, G.B. Nilsen, E. Halperin, E. Eskin, D.G. Ballinger, K.A. Frazer, and D.R. Cox, “Whole-Genome Patterns of Common DNA Variation in Three Human Populations,” *Science,* vol. 307, no. 5712, pp. 1072-1079, 2005.

[14] The Int’l HapMap Consortium, “A Haplotype Map of the Human Genome,” *Nature,* vol. 437, no. 7063, pp. 1299-1300, 2005.

[15] L. Wang and Y. Xu, “Haplotype Inference by Maximum Parsimony,” *Bioinformatics,* vol. 19, no. 14, pp. 1773-1780, 2003.

[16] D.G. Brown and I.M. Harrower, “A New Integer Programming Formulation for the Pure Parsimony Problem in Haplotype Analysis,” *Proc. Fourth Ann. Workshop Algorithms in Bioinformatics,* pp. 254-265, 2004.

[17] Z. Li, W. Zhou, X.S. Zhang, and L. Chen, “A Parsimonious Tree-Grow Method for Haplotype Inference,” *Bioinformatics,* vol. 21, no. 17, pp. 3475-3481, 2005.

[18] Y.-T. Huang, K.-M. Chao, and T. Chen, “An Approximation Algorithm for Haplotype Inference by Maximum Parsimony,” *J. Computational Biology,* vol. 12, no. 10, pp. 1261-1274, 2005.

[19] J.E. Mitchell, *Branch-and-Cut Algorithms for Combinatorial Optimization Problems,* pp. 65-77. Oxford Univ. Press, Jan. 2002.

[20] A. Charnes, “Optimality and Degeneracy in Linear Programming,” *Econometrica,* vol. 20, pp. 160-170, 1952.

[21] C. Bron and J. Kerbosch, “Algorithm 457: Finding All Cliques of an Undirected Graph,” *Comm. ACM,* vol. 16, no. 9, pp. 575-577, 1973.

[22] *ILOG CPLEX 8. 1: Reference Manual,* ILOG SA, Dec. 2002.

[23] R.R. Hudson, “Generating Samples under a Wright-Fisher Neutral Model of Genetic Variation,” *Bioinformatics,* vol. 18, no. 2, pp. 337-338, 2002.

[24] The Int’l HapMap Consortium, “Integrating Ethics and Science in the International HapMap Project,” *Nature Rev. Genetics,* vol. 5, no. 6, pp. 467-475, 2004.

[25] T. Barzuza, J. Beckmann, R. Shamir, and I. Pe’er, “Computational Problems in Perfect Phylogeny Haplotyping: Xor-Genotypes and Tag SNPs,” *Proc. 14th Ann. Symp. Combinatorial Pattern Matching,* pp. 14-31, 2004.

[26] V. Bafna, D. Gusfield, S. Hannenhalli, and S. Yooseph, "A Note on Efficient Computation of Haplotypes via Perfect Phylogeny," *J. Computational Biology*, vol. 11, no. 5, pp. 858-866, 2004.

[27] V. Bafna, D. Gusfield, G. Lancia, and S. Yooseph, "Haplotyping as Perfect Phylogeny: A Direct Approach," *J. Computational Biology*, vol. 10, pp. 323-340, 2003.

Daniel G. Brown received the undergraduate degree in mathematics with computer science from the Massachusetts Institute of Technology in 1995 and the PhD degree in computer science from Cornell University in 2000. He then spent a year as a research scientist at the Whitehead Institute/MIT Center for Genome Research in Cambridge, Massachusetts, working on the Human and Mouse Genome Projects. Since 2001, he has been an assistant professor in the David R. Cheriton School of Computer Science at the University of Waterloo, Canada.

Ian M. Harrower received the BMath degree in computer science and combinatorics and optimization from the University of Waterloo, Canada, in 2003 and the MMath degree in computer science from the University of Waterloo in 2005. He is a doctoral candidate in the David R. Cheriton School of Computer Science at the University of Waterloo.