A new formulation for haplotype inference by pure parsimony

Daniel G. Brown   Ian M. Harrower

School of Computer Science, University of Waterloo,
Waterloo ON N2L 3G1 Canada
{browndg,imharrow}@cs.uwaterloo.ca

Technical Report CS-2005-03

Abstract

We present a new integer programming formulation for the haplotype inference by pure parsimony problem. Previous formulations for this problem are either of potentially exponential size [7] or are of polynomial size, yet can require too much time to solve reasonable-sized instances [8, 3]. We present an approach to this problem that is a hybrid between these approaches and inherits many of the strengths of both. Problems that are too complex for the exponential-sized formulations can still be solved by our new formulation in a reasonable amount of time. Our formulation can also be used in a variety of extensions that allow errors in the input to the problem or some modelling of the structure of the population under consideration.

1 Introduction

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

This process is made more difficult because of a technological limitation. At a genomic position for which an individual inherited two different alleles, it is currently difficult to identify from which parent each allele was inherited. Instead, researchers can only identify that the individual is heterozygotic at that position. If we could identify maternal and paternal inheritance better, we could trace the structure of the human population more accurately and improve our ability to map disease genes. This process of going from genotypes (which include this ambiguity at heterozygotic positions) to haplotypes (where we know from which parent each allele is inherited) is called *haplotype inference*.

Gusfield [6] brought this problem into the combinatorial literature. In 2003, he studied the variant where, given the 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 [7]. This problem, called *Haplotype Inference by Pure Parsimony (HIPP)*, is the focus of this paper.

Gusfield [7] gave an integer linear programming (IP) formulation for the HIPP problem. Its size varies exponentially with the number of heterozygous positions in genotypes. For problems giving rise to moderate-sized IP instances, the IP solved extremely quickly, but many instances were simply too big.

Haldórsson *et al.* [8] presented a polynomial-sized integer program for this problem, which was also independently discovered and extended by us [3]. In our experiments, we noted that the smaller IP often solved well, but that it was typically much slower to solve than Gusfield's formulation, assuming Gusfield's was small enough to be stored in memory.

Here, we attempt to produce a compromise between the strengths of both of these approaches. We give an IP formulation of the HIPP problem that is of polynomial size, and yet which achieves much of the speed advantage of the exponential-size formulation of Gusfield [7]. Our new IP approach, which we call the HybridIP, combines these two approaches. 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 an exponential set. Like the polynomial IP of Haldórsson *et al.*, 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 previous 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; for problems where Gusfield's would be too large to be stored, we still show good success in solving them in practical runtimes.

We also show how our new formulation can be adapted to a variety of extensions of HIPP recently presented by Kalpakis and Namjoshi [10]. Most of these are to various models for error in the data, but one concerns possible technological limitations in the instruments, and two begin to bring the problem into a scenario where some of the structure of the population under study is known.

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 SNP s_j.

Using notation suggested by He and Zelikovsky [9], 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 let $G[i, j] = 0$ 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, where the roles of 1 and 2 are typically reversed.) We will refer to the rows of G as genotype vectors; each corresponds to all SNPs tested 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 parents of the genotypes in G. Genotype g is *explained* by two haplotypes h_1 and h_2 if $g[j] = h_1[j] + h_2[j]$ for all positions j of the genotype g, and h_1

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

Figure 1: Gusfield's TIP formulation for HIPP [7]

and h_2 form an *explaining pair* for g. At homozygotic positions in g, with value 0 or 2, both parents must have the same value, while at heterozygotic positions, with value 1, one parent must have allele 1, while the other has allele 0.

Haplotype inference involves computing 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 [7] justifies this objective by noting that few haplotypes seem to be found in recombination-free blocks of the human genome [5, 12], and by noting that previous methods for haplotype inference, most notably those due to Clark [4], found that their results were more likely correct when they returned small sets of haplotypes. This goal of minimizing the number of haplotypes is called *Haplotype Inference by Pure Parsimony*, and has been studied by several authors (*e.g.* [7, 10, 8, 13, 11, 3]).

The HIPP problem is NP-complete (Gusfield [7] cites correspondence from Hubbell for this fact; a proof is in Lancia *et al.* [11]), 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 [11]. As such, exponential time algorithms such as integer programming are appropriate as a way to solve the problem.

2.1 Previous integer programming formulations

In 2003, Gusfield [7] 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 of value 1 if the haplotype is used and 0 otherwise.

The 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), haplotypes j and k are included in the set of haplotypes chosen. The objective is to minimize the sum of the x_i variables. The IP is shown in Figure 1.

Gusfield also presented a size reduction that avoids expanding out all possible parents. 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

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

Figure 2: The polynomial-sized IP formulation introduced by Halldórsson *et al.* [8]

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 reasonably.

A polynomial-sized IP In 2003, two groups [8, 11] independently identified similar polynomial-sized formulations for HIPP. Slightly later, and also independently, we discovered the same formulation as Halldórsson *et al.* [8], and experimented with it to see if it is practical [3]. We will call this formulation the PolyIP. It represents the chosen haplotypes 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 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 of haplotypes $1 \le i < j \le 2n$, we create a variable $d_{i,j}$, equal to 1 when $h_i \ne h_j$. If $h_i \ne h_j$, there is some position k in the haplotypes where $h_i[k] = 1$ and $h_j[k] = 0$ or vice versa, and constraint (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 constraint (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 Figure 2.

Brown and Harrower [3] studied this program. We showed that, with some augmentations to the objective function, and the addition of several additional constraints, or cuts, this formulation can solve many instances of HIPP for which the RTIP formulation cannot be stored in a reasonable amount of space. However, solution times were much worse: where the RTIP would typically solve in seconds, our experiments on the same problem instances often required minutes to hours, and could not solve many instances of HIPP in several hours.

3 A new integer linear programming formulation

We are left with the question of whether an integer program can be formulated that has both practical size and reasonable runtimes. Here, we satisfy this goal with a formulation that draws on 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 possible pairs of explaining parents. However, large problems may have some genotypes with billions of pairs of explaining parents, so the IP is too huge to use. In our HybridIP, we expand only a few possible explaining parents, while still allowing parents that were not among those expanded.

3.1 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 decision variables: $h_i[k]$ is represented by decision variable $y_{i,k}$; constraint (11) requires that haplotypes properly explain their genotypes.

Unlike for the PolyIP, we also choose a set $H_e = \{h'_1, \ldots, h'_K\}$ of K 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 h'_j and h'_k are both in H_e, we create a variable $w_{i,(j,k)}$, as in the RTIP formulation. If there is an explaining pair (h'_j, h_k) where h'_j is in H_e, but h_k is not, we create a variable $w_{i,(j,*)}$. If there is an explaining pair (h_j, h_k) for g_i where both haplotypes are not in H_e, we create a variable u_i. Let W_i be the set of all $w_{i,(j,k)}$ and $w_{i,(j,*)}$ variables. The selection constraint (8) 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 register differences between the explicitly enumerated haplotypes, as for the PolyIP. The $d'_{a,b}$ variables identify whether enumerated haplotype h_b equals h'_a, where h'_a is in H_e. If we explain genotype g_i by setting $w_{i,\rho} = 1$, constraints (18) and (19) ensure that the haplotypes are properly set. Meanwhile, if we choose an explaining pair neither of whose members are in H_e, constraint (16) ensures both haplotypes are kept different from all haplotypes in H_e.

As before, we calculate how many unique haplotypes are used by a solution. We have one decision variable x'_j, for all h'_j in H_e, set to 1 if haplotype h'_j is ever chosen. This is enforced by constraints (9) and (10). 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 $(h_1, \ldots, h_i)$; constraint (20) 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 Figure 3.

If K, the size of H_e, is a constant , 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 closely analogous to Gusfield's TIP formulation [7].

Many possible approaches exist to populate the set H_e. We have had good success by sorting the genotypes by their number of heterozygous sites and expanding the genotypes into H_e using the RTIP expansion, until H_e reaches size K, for a constant K. For the sizes of problems evaluated by Brown and Harrower [3], a value of $K = 32$ has yielded good results; the following section presents our preliminary results.

All of the valid cuts shown by Brown and Harrower [3] are also valid for the HybridIP, giving an additional way to improve the speed of solving the HIPP problem through integer programming.

$$
\begin{array}{rll}
\text{minimize} \sum_{i=1}^{|H_e|} x'_i + \sum_{i=1}^{2n} x_i, & \text{subject to} & \\
\sum_{w_{i,\rho} \in W_i} w_{i,\rho} + u_i = 1, & \text{for each genotype } g_i & (8)\\
x'_j \geq w_{i,(j,k)}, & \text{for all expanded pairs } (j,*) \text{ or } (j,k) & (9)\\
x'_k \geq w_{i,(j,k)}, & \text{for all expanded pairs } (j,k) & (10)\\
y_{2i-1,k} + y_{2i,k} = g_i[k], & \text{for each } i \text{ and } k & (11)\\
y_{i,k} - y_{j,k} \leq d_{i,j}, & \text{for each } i,j \text{ and } k & (12)\\
y_{j,k} - y_{i,k} \leq d_{i,j}, & \text{for each } i,j \text{ and } k & (13)\\
y_{j,k} \leq d'_{i,j}, & \text{if } h'_i[k] = 0 & (14)\\
1 - y_{j,k} \leq d'_{i,j}, & \text{if } h'_i[k] = 1 & (15)\\
d'_{j,2i} \geq u_i, & \text{for all } j \text{ and all genotypes } g_i & (16)\\
d'_{j,2i-1} \geq u_i, & \text{for all } j \text{ and all genotypes } g_i & (17)\\
d'_{k,2i} \leq 1 - w_{i,(j,k)}, & \text{for all } j,k \text{ and all genotypes } g_i & (18)\\
d'_{k,2i} \leq 1 - w_{i,(j,k)}, & \text{for all } j,k \text{ and all genotypes } g_i & (19)\\
x_i \geq 2 - (K+i) + \sum_{j=1}^{K} d'_{j,i} + \sum_{j=1}^{i-1} d_{j,i}, & \text{for each } i & (20)\\
x_i, x'_i, y_{i,k}, d_{i,j}, d'_{i,j}, w_{i,\rho}, u_i \in \{0,1\}, & \text{for all } i,j \text{ and } k. &
\end{array}
$$

Figure 3: The HybridIP formulation for haplotype inference by pure parsimony

3.2 Experiments

We created a branch-and-cut implementation of the HybridIP, using the cuts from Brown and Harrower [3] to solve HIPP instances. We use CPLEX 8.1.1 to solve the LP relaxations, and then add cuts or set branched variables directly. We ran the program on the same data instances as in our previous paper [3] that evaluated the PolyIP. We populated the set of expanded haplotypes with the first 32 haplotypes expanded by the RTIP expansion.

There are seven data sets. The four data sets, with each genotype of length 30 or less, contain 50 population members each, and were chosen to evaluate problems of size similar to those used by Gusfield [7] in his analysis of the RTIP. Two of these sets include simulated recombination in the data generation process; see Brown and Harrower [3] for more details.

The three data sets with genotype length 50 or greater each contain 30 population members. At these longer lengths, even the polynomial-sized IPs began to get too large with 50 population members. The results of the tests are presented in Table 1. For the shorter sequences, we count a problem as solved if it required fewer than 2000 integer programming branches to solve to optimality. For the larger sequences, we terminated the HybridIP solution after 1000 branches, since the time to solve the LP relaxations for the HybridIP is slightly longer.

For shorter sequences, the HybridIP was able to solve all the instances. For the length 10 sequences, all but two instance 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 of the instances solved in less than thirty seconds, and twelve in less than two minutes. The final three instances all took longer than five minutes,

Table 1: Experimental comparison of three IP formulations. 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.

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

with the longest 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 HybridIPs solved on the first branch.

On the 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 polynomial IP.

All of the runtimes were much faster than for the PolyIP. Many length 50 instances solved in under five seconds, most within ten 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.

Table 2 shows the effect of varying K, the size of the set H_e. These tests were run on the sequences of length 30, with a branch limit set of 300. Even expanding very few haplotypes, greatly improves the runtimes and the number of branches needed over a set of size 0. Each non zero values tested were able to solve at least fourteen out of fifteen instances. When at least 32 haplotypes were expanded, all the instances were solved within 300 branches.

These results show that the HybridIP improves greatly over the PolyIP and can achieve runtimes much closer to those of the RTIP. With more care in selecting how the set H_e is populated and the choice of the size of H_e, it is likely better results could be achieved.

3.3 Variations on HIPP

The HIPP problem can be expanded to accomodate 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 [10] presented a number of these variations, and showed how their semidefinite programming formulation of HIPP could be extended to accomodate them. Here, we show that the HybridIP can easily be expanded to include these, as well.

Table 2: The effect of number of haplotypes expanded on performance of the HybridIP. This table shows the number of instances of length 30 solved within 300 branches in each time interval. Small values of K show great improvement over not expanding any haplotypes, which is equivalent to the PolyIP.

# of haplotypes	Instances solved in time:				
expanded	$<$ 30 sec	30 - 90 sec	90 sec - 5 min	$>$ 5 min	Total Solved
0 (the PolyIP)	0	0	0	5	5/15
16	4	2	5	3	14/15
24	6	4	2	2	14/15
32	6	6	1	2	15/15
48	6	3	4	2	15/15
64	6	6	1	2	15/15

Various forms of error The input genotype matrix G may not be accurate. It could be incomplete, with missing entries. It could have small numbers of mistakes in it. Or, there might be a small number of genotypes with many errors. All three of these scenarios can be adapted to our IP. 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 constraint (11) 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 constraint (11) to:

$$y_{2i-1,k} + y_{2i,k} = g_i[k] + e_{i,k} \quad \text{for each } i \text{ and } k. \tag{21}$$

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:

$$e'_{i,k} \geq e_{i,k}/2 \text{ and } e'_{i,k} \geq -e_{i,k}/2, \qquad \text{for each } i \text{ and } k \tag{22}$$

$$\textstyle\sum_{i=1}^{n} \sum_{k=1}^{m} e'_{i,k} \leq E. \tag{23}$$

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:

$$x'_j \geq 1 - d'_{i,j}, \quad \text{for each } i \text{ and } j. \tag{24}$$

In the third scenario, where there is a bound E on the number of genotypes that posess 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 constraint (21); then, we limit the sum of the e_i to be at most E. We must again add constraint (24) to ensure we pay for all chosen haplotypes.

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:

$$d_{2i-1,2j-1} + d_{2i,2j-1} + d_{2i-1,2j} + d_{2i,2j} = 3. \tag{25}$$

XOR genotypes Barzuza *et al.* [2] have proposed the study of haplotype inference on a new type of data called XOR-genotypes. Their genotype vectors are based on the observation that 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 $g = h_1 + h_2 \mod 2$.

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 $g - h \mod 2$. Still, the PolyIP is appropriate; we replace constraint (4) with the constraints:

$$y_{2i-1,k} + y_{2i,k} = 1, \qquad \text{if } g_i[k] \text{ is heterozygous} \tag{26}$$
$$y_{2i-1,k} = y_{2i,k}, \qquad \text{if } g_i[k] \text{ is homozygous} \tag{27}$$

Perfect phylogeny A final variation is to require that the solution satisfy a perfect phylogeny. Space limitations prevent our giving this a full explanation, but we this requires that the solution be compatible with a single phylogenetic tree, where each column of the input genotypes arose from a single mutation; see Bafna *et al.* [1] for more about this problem. Our IP can be adapted to this domain with the addition of $O(m^2)$ new variables to represent possible edges of a graph discussed in Bafna *et al.*'s paper, and $O(m^3)$ new variables to represent the existence of paths of length 2 in that graph. The details are omitted in this extended abstract, partially because they are unlikely to give practical runtimes.

4 Conclusion

We have presented a new integer programming formulation for the Haplotype Inference by Pure Parsimony problem. The two previous IP formulations for this problem both suffered limitations: the RTIP formulation of Gusfield [7] was potentially of exponential size, while the polynomial-sized formulation of Halldórsson *et al.* [8], which we independently discovered [3], experienced much slower runtimes. Our new formulation is a hybrid between these two approaches. It enumerates a small number of possible explaining haplotypes, but also explicitly encodes all of the haplotypes used in a particular solution. Using our hybrid IP, we achieve runtimes that are worse than for the set of problems where Gusfield's formulation is small enough to formulate, but which are still tolerable for a much wider range of problems.

Acknowledgements This research has been supported by the Natural Science and Engineering Research Council of Canada, through a Discovery Grant to the first author and a Postgraduate Scholarship to the second author.

References

[1] 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.

[2] T. Barzuza, J. Beckmann, R. Shamir, and I. Pe'er. Computational Problems in Perfect Phylogeny Haplotyping: Xor-Genotypes and Tag SNPs. In *Proceedings of CPM 2004*, volume 3109 of *LNCS*, pages 14–31. Springer-Verlag, 2004.

[3] D. G. Brown and I. M. Harrower. A new integer programming formulation for the pure parsimony problem in haplotype analysis. In *Proceedings of WABI 2004*, volume 3240 of *Lecture Notes in Bioinformatics*, pages 254–265. Springer, 2004.

[4] A. G. Clark. Inference of haplotypes from PCR-amplified samples of diploid populations. *Molecular Biology and Evolution*, 7(2):111–112, 1990.

[5] 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*, 29(2):229–232, 2001.

[6] D. Gusfield. Inference of haplotypes from samples of diploid populations: complexity and algorithms. *Journal of Computational Biology*, 8(3):305–313, 2001.

[7] D. Gusfield. Haplotype inference by pure parsimony. In *Proceedings of CPM 2003*, volume 2676 of *LNCS*, pages 144–155. Springer-Verlag, 2003.

[8] B. V. Halldórsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph, and Sorin Istrail. A survey of computational methods for determining haplotypes. In *Computational Methods for SNPs and Haplotype Inference: DIMACS/RECOMB Satellite Workshop*, volume 2983 of *LNCS*, pages 26–47. Springer-Verlag, 2004.

[9] J. He and A. Zelikovsky. Linear reduction for haplotype inference. In *Proceedings of WABI 2004*, volume 3240 of *Lecture Notes in Bioinformatics*, pages 242–253. Springer, 2004.

[10] K. Kalpakis and P. Namjoshi. Haplotype phasing using semidefinite programming. Techical report TR CS-04-10, University of Maryland Baltimore County, 2004.

[11] G. Lancia, C. M. Pinotti, and R. Rizzi. Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms. *INFORMS Journal on Computing*, 16(4):348–359, 2004.

[12] N. Patil, A. J. Berno, D. A. Hinds, W. A. Barrett, J. M. Doshi, et al. Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. *Science*, 294(5547):1719–1723, 2001.

[13] L. Wang and Y. Xu. Haplotype inference by maximum parsimony. *Bioinformatics*, 19(14):1773–1780, 2003.