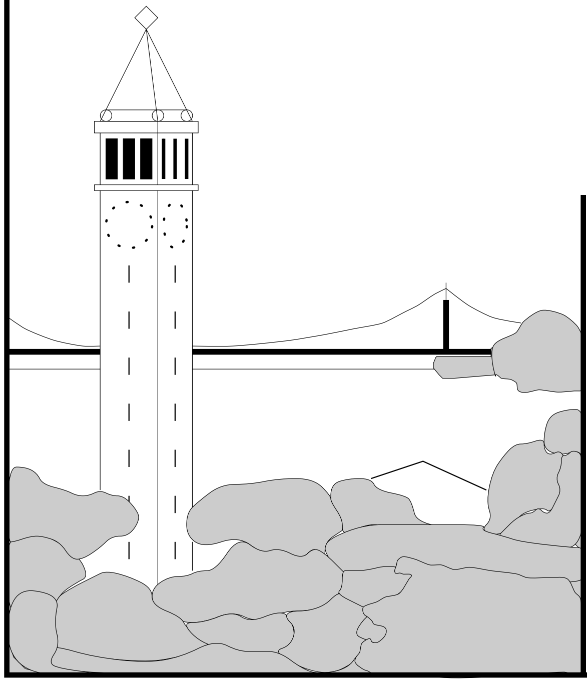
Efficient Reconstruction of Haplotype Structure via Perfect Phylogeny

Eleazar Eskin *Eran Halperin* *Richard M. Karp*



Report No. UCB/CSD-2-1196

August 2002

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Efficient Reconstruction of Haplotype Structure via Perfect Phylogeny

Eleazar Eskin[*] Eran Halperin[†] Richard M. Karp [‡]

August 2002

Abstract

Each person's genome contains two copies of each chromosome, one inherited from the father and the other from the mother. A person's *genotype* specifies the pair of bases at each site, but does not specify which base occurs on which chromosome. The sequence of each chromosome separately is called a *haplotype*. The determination of the haplotypes within a population is essential for understanding genetic variation and the inheritance of complex diseases. The haplotype mapping project, a successor to the human genome project, seeks to determine the common haplotypes in the human population.

Since experimental determination of a person's genotype is less expensive than determining its component haplotypes, algorithms are required for computing haplotypes from genotypes. Two observations aid in this process: first, the human genome contains short blocks within which only a few different haplotypes occur; second, as suggested by Gusfield, it is reasonable to assume that the haplotypes observed within a block have evolved according to a *perfect phylogeny*, in which at most one mutation event has occurred at any site.

We present a simple and efficient polynomial-time algorithm for inferring haplotypes from the genotypes of a set of individuals assuming a perfect phylogeny. Using a reduction to 2-SAT we extend this algorithm to handle constraints that apply when we have genotypes from both parents and child. We also present a hardness result for the problem of removing the minimum number of individuals from a population to ensure that the genotypes of the remaining individuals are consistent with a perfect phylogeny.

Our algorithms have been tested on real data and give biologically meaningful results.

1 Introduction

Critical to the understanding of the genetic basis for complex diseases is the modeling of human genetic variation. Most of this variation can be characterized by single nucleotide polymorphisms (SNPs) which are mutations at a single nucleotide position that occurred once in human history and been passed on through heredity. Although the two chromosomes of an individual can be separated and analyzed independently as in the study of [18], current technology suitable for large scale polymorphism screening obtains *genotype* information at each SNP. The genotype gives the bases at the SNP for both copies of the chromosome, but does not identify the chromosome on which each base appears. Consider a SNP where there are two common bases, A or G. There are four possible cases for the genotype. Two of the cases are where either both chromosomes contain A or both chromosomes contain G. We refer to these cases as *homozygous* genotypes. The other two cases are where the first chromosome contains A and the second contains G and vice versa. We refer to these cases as *heterozygous* genotypes. Thus, the genotype consists of the mutual information on the two chromosomes. The sequence of each chromosome separately is called the haplotype information. Consider a case where, at four successive SNPs, with possible values A or G, an individual has a genotype $AHHG$, where H represents a heterozygous site. In this case, the individual's haplotypes have two possibilities: either one chromosome contains $AAAG$ and the other contains $AGGG$ or one chromosome contains $AAGG$ and the other contains $AGAG$.

[*]Computer Science Department, Columbia University. E-mail: eeskin@cs.columbia.edu.
[†]CS Division, Soda Hall, University of California Berkeley, CA 94720-1776. E-mail: eran@eecs.berkeley.edu.
[‡]International Computer Science Institute, 1947 Center St., Berkeley, CA 94704. E-mail:karp@cs.berkeley.edu.

One of the first goals set by the NIH right after the completion of the human genome project is the haplotype mapping project. The goal of the human genome project was to find the consensus genotype sequence of humans. In order to achieve more information on genetic disease, one has to know not only the genotype data, but also the haplotype data, and not only the consensus, but which are the common haplotypes. Recent studies [5, 18] have shown that SNPs that are physically close to each other on the chromosome are usually correlated, and that our chromosomes can be partitioned into blocks, so that in each block there is a strong correlation between all the SNPs contained in it. These studies show that for each block, the number of possible variation is usually very small (3 or 4), while the number of SNPs in the block could be as large as 30. The goal of the haplotype mapping project is to gather all the haplotype information, that is, for each block, to list all possible combinations of SNPs that appear in the population. After having all this information, one could perform a more accurate case study to associate genes (and maybe blocks) with diseases, and furthermore, one could sequence the human chromosomes much faster with high accuracy. This paper takes the haplotype mapping project one step forward by finding an efficient way to infer the haplotype data of a population by observing only the genotypes of the population.

Given a set of n genotypes, each of length m, we address the problem of inferring the haplotype structure. Clearly, in the absence of additional information, one cannot infer the haplotype structure, since there are many possible solutions. Gusfield [10] suggested to add the constraint that the resulting set of haplotypes should correspond to a phylogeny model known as the coalescent model, or perfect phylogeny. In this model we assume perfect Mendelian heredity, that is, one of the chromosomes of each parent is transmitted to the child with some possible mutations. Furthermore, we assume that in each SNP site, there has only been one mutation throughout the history represented by the tree. We wish to find a directed phylogenetic tree that will correspond to these two assumptions, such that each of the resulting haplotypes will be found in one of the leaves of that tree. The problem is referred to as the Perfect Phylogeny Haplotype problem (PPH problem). We will formally state the problem in Section 2.1. Gusfield [10] introduces a polynomial time algorithm for the PPH problem. His algorithm uses as a black box an algorithm to recognize graphic matroids [21, 2]. This algorithm is very complicated and impractical. One of the open problems mentioned in Gusfield's paper is to find a simple and efficient algorithm to the PPH problem. In this paper we introduce an efficient and simple solution to the problem, using no heavy machinery. The simplicity of our algorithm sheds a new insight on the problem and allows us to cope with some extensions of the model. We show relations between the extensions of the problem and other combinatorial problems such as 2-satisfiability and minimum CNF deletion.

We begin by presenting an extremely simple and elegant polynomial-time algorithm for the problem. Although this algorithm may be effective in practice its time bound is unreasonably high. Therefore we go on to present our main algorithm, which runs in $O(nm^2)$ time, and produces a simple linear size data structure which can be used to produce all possible solutions to the problem. Each possible solution can be implicitly enumerated in time $O(m)$ (clearly, to output the solution one needs $O(mn)$ time). We furthermore extend our main algorithm to handle the additional constraint that some of the individuals are related, and therefore, a parent must transmit one of its haplotypes to a child, and each child has one haplotype transmitted from its father, and the other from its mother. We use the data structure returned by our main algorithm for the PPH problem as a starting point for the algorithm, and reduce the resulting problem to the 2-SAT problem which can be solved in polynomial time [1, 4, 19, 16].

Finally, we address the problem of finding a minimal set of individuals such that by removing them from our data set, we will be able to find at least one solution to the PPH problem. We show that finding an α-approximation to this problem will imply an α-approximation algorithm for the Min UnCut problem, where a graph $G = (V, E)$ is given, and the goal is to remove the minimum number of edges in G such that the remaining graph is bipartite. This problem has a $\log n$-approximation algorithm by a reduction from the minimum 2-CNF deletion [15]. On the negative side it is only known that the problem is MAX-SNP hard, and therefore there is no PTAS for the problem [17] unless P=NP.

We evaluate our algorithm over the data collected in the study of a 500 kilobase region of chromosome 5p31 containing 103 SNPs from the study of [5]. In this study, genotypes are collected from 129 mother, father, child trios and the correct child haplotypes are inferred from these genotypes. In our experiments, we use our method to make

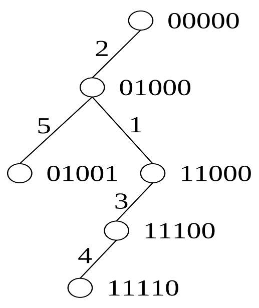


Figure 1: The coalescent tree corresponding to the haplotypes $00000, 01000, 01001, 11000, 11100, 11110$

predictions of the child's haplotypes from the child's genotypes and then check our predictions against the correct haplotypes inferred from the trios. Our results indicate that the algorithm is practical and efficient, and gives biologically meaningful results.

There are several previous approaches to determining the haplotype information from genotype data. These methods include the parsimony approach of Clark [3] and related approaches [8, 9, 13], maximum likelihood methods [6, 12, 14, 7] and statistical methods such as PHASE [20]. These approaches use heuristics, but in practice they do not scale to data that contains more than 30 sites, while our algorithm could cope with large data sets.

2 Preliminaries

We first formally describe the coalescent model (perfect phylogeny) and the PPH problem. We assume that at each polymorphism site there are two possible nucleotides that appear at any position in any one of the chromosomes. Let us denote these nucleotides by 0 and 1. We note that although the assumption that there are only two possibilities at any site seems artificial, it is the case in most polymorphism sites.

2.1 The Perfect Phylogeny Haplotype Problem

Given a $\{0, 1\}$ matrix $B = (b_{ij})$ of size $n \times m$ which represents a set of haplotypes, we say that B fits the coalescent model if there exists a rooted tree $T(B)$ such that the following holds:

1. Each vertex v of $T(B)$ is labeled by a row vector $l(v)$ of length m representing a possible haplotype that existed through history.
2. For each vertex v and its parent u, the set of sites where $l(u)$ and $l(v)$ differ is called the mutations from u to v.
3. In each of the m sites, there is at most one parent-child pair in the tree, where there is a mutation in this site from the parent to the child.
4. The set of rows of B is contained in the set of labels of $T(B)$.

An example for a coalescent tree is given in Figure 1. We first introduce some notations. Given a pair of chromosomes ch_1, ch_2, and a given site j in the chromosome, we say that the haplotype configuration in ch_1, ch_2 is (x, y), if the nucleotides present in site j in ch_1 and ch_2 are x and y respectively. Throughout the paper, for a given integer k, we denote the set $\{1, \ldots, k\}$ by $[k]$. For a matrix M, we denote the i-th row of M by $M(i, :)$.

We are now ready to define the Perfect Phylogeny Haplotype (PPH) problem.

The Perfect Phylogeny Haplotype Problem (PPH). An input to the PPH problem is an $n \times m$ matrix $A = (a_{ij})$, where $a_{ij} \in \{0, 1, 2\}$ for each $i \in [n], j \in [m]$. The goal is to construct a $\{0, 1\}$ matrix $B = (b_{ij})$ of dimension

$2n \times m$, such that for every $i \in [n], j \in [m]$, if $a_{ij} \neq 2$ then $b_{2i-1,j} = b_{2i,j} = a_{ij}$, if $a_{ij} = 2$ then $b_{2i-1,j} \neq b_{2i,j}$, and that B fits the coalescent model, or determine that such a matrix B does not exist.

The matrix A represents the genotypes of n individuals, where the length of each genotype is m. For any individual $r \in [n]$, and site $c \in [m]$, the four possible haplotype states are $(0,0),(0,1),(1,0),(1,1)$. The sequence observed can only distinguish between the three states $0,1$ and 2. States 0 and 1 stand for the haplotype states $(0,0)$, $(1,1)$ respectively. State 2 stands for either the haplotype state $(0,1)$ or $(1,0)$. We are interested in constructing the haplotype matrix B which will be consistent with A and with the coalescent model. We call such a matrix a legal extension of A.

Before introducing the algorithms we have to introduce some notations and definitions.

2.2 Some Useful Lemmas and Notations

Throughout the paper we will use the terms sites and columns alternatively to represent the columns of the matrix. Let $c_1, \ldots, c_m$ be the set of sites. For a site $c \in [m]$ and a value $x \in \{0,1,2\}$, let $c(A,x) = \{r \in [n] \mid a_{rc} = x\}$, and $c(B,x) = \{r \in [2n] \mid b_{r,c} = x\}$. For a matrix M and a column c, we denote by $M \setminus c$ the matrix generated by truncating column c from M.

We say that two sites c, c' are compatible in a matrix M if at least one of the following sets is empty:

- $(c(M,1) \cap c'(M,1)) \cup (c(M,2) \cap c'(M,1)) \cup (c(M,1) \cap c'(M,2))$.
- $(c(M,1) \cap c'(M,0)) \cup (c(M,2) \cap c'(M,0)) \cup (c(M,1) \cap c'(M,2))$.
- $(c(M,0) \cap c'(M,1)) \cup (c(M,0) \cap c'(M,2)) \cup (c(M,2) \cap c'(M,1))$.

If c and c' are not compatible then we say that c and c' have a conflict. The following lemma has been proven independently by several authors:

Lemma 2.1. *A $\{0,1\}$ matrix $M_{m \times n}$ corresponds to a perfect phylogeny if and only if every two sites are compatible in M. Furthermore, M corresponds to a perfect phylogenetic tree with root $(x_1, \ldots, x_m)$ if and only if for each pair of sites c_i, c_j, one of the sets $c_i(M, 1-x_i) \cap c_j(M, 1-x_j), c_i(M, x_i) \cap c_j(M, 1-x_j), c_i(M, 1-x_i) \cap c_j(M, x_j)$ is empty.*

By Lemma 2.1, in the PPH problem, we have to assign $\{0,1\}$ values to the 2-entries in B so that there is no conflict. Note that without loss of generality we may assume that the root of the tree is the all zeros vector for the following reason. Change the values of the entries of A such that in each site c, either $|c(A,0)| > |c(A,1)|$ or $|c(A,0)| = |c(A,1)|$ and the minimal index i_0 such that $a_{i_0,c} = 0$ is smaller then the minimal index i_1 such that $a_{i_1,c} = 1$. In this case, it is easy to see that unless there are two identical columns c, c' where $c(A,0) = \phi$, then every two sites c, c' satisfy that $c(A,0) \cap c'(A,0) \neq \phi$. Therefore, B must also satisfy that $c(B,0) \cap c'(B,0)$. Thus c and c' are compatible in B if and only if one of the sets $c(B,1) \cap c'(B,1), c(B,1) \cap c'(B,0)$ and $c(B,0) \cap c'(B,1)$ is empty.

Thus, from now on we assume that the root of the tree is the all zeros vector. If $c(B,1) \cap c'(B,1) \neq \phi$ then c and c' must lie on the same path from a leaf to the root, and thus, one must be an ancestor of the other. On the other hand, if $c(B,1) \cap c'(B,0) \neq \phi$, then there is a path from the leaf to the root, where c lies in the path while c' does not. This implies that c' cannot be an ancestor of c.

3 A Simple Algorithm for the PPH Problem

Let the $n \times m$ matrix A be an input to the PPH problem, and let the $2n \times m$ $\{0,1\}$ matrix B be a legal solution to the problem. Let c and c' be two columns such that $c(A,2) \cap c'(A,2) \neq \phi$. Let us say that B *resolves* the pair of columns (c, c') *unequally* if $c(B,1) \cap c'(B,1) = \phi$ and *equally* if $c(B,0) \cap c'(B,1) = \phi$ or $c(B,1) \cap c'(B,0) = \phi$. Then

B must resolve the pair (c, c') either equally or unequally, and cannot resolve the pair both equally and unequally. Solving the PPH problem is equivalent to deciding in a consistent way which pairs of columns to resolve equally and which to resolve unequally. These decisions essentially determine the matrix B. In order to determine the constraints on a consistent solution we classify the ordered pairs of columns.

Each ordered pair (c, c') of columns is of one of four types.

Type A: $c(A, 2) \cap c'(A, 2) = \phi$. In this case the pair (c, c') does not have to be resolved.

In the remaining three cases $c(A, 2) \cap c'(A, 2) \neq \phi$ and the pair (c, c') does have to be resolved. The cases are as follows:

Type 0 $(c(A, 1) \cap c'(A, 1)) \cup (c(A, 2) \cap c'(A, 1)) \cup (c(A, 1) \cap c'(A, 2)) \neq \phi$. In this case the pair (c, c') must be resolved equally.

Type 1 $(c(A, 0) \cap c'(A, 1)) \cup (c(A, 0) \cap c'(A, 2)) \neq \phi$ and $(c(A, 1) \cap c'(A, 0)) \cup (c(A, 2) \cap c'(A, 0)) \neq \phi$. In this case the pair (c, c') must be resolved unequally.

Type x (c, c') is neither of Type 1 nor of Type 2. In this case (c, c') may be resolved either equally or unequally.

Note that (c, c') and (c', c) are of the same type and must be resolved in the same way (either both equally or both unequally). In completing the description of the algorithm we work with unordered pairs of columns.

A resolution of the pairs of Types 0,1 and x can be represented by a symmetric *labeling function* $L(c, c')$ which is equal to 0 if (c, c') is resolved equally, and to 1 if (c, c') is resolved unequally. A labeling function is *legal* if it yields a legal solution to the PPH problem.

For any row r let $V_r = \{c \mid A(r, c) = 2\}$. The proof of the following theorem will be given in the full version of the paper.

Theorem 3.1. *A labeling function L is legal if and only if:*

1. *Every pair of columns is compatible in A.*
2. *If (c, c') is of Type 0 then $L(c, c') = 0$;*
3. *If (c, c') is of Type 1 then $L(c, c') = 1$*
4. *For each r, V_r is partitioned into two parts such that, for all c and c' in V_r, $L(c, c') = 0$ if and only if c and c' are in the same part.*

The last condition of this Theorem can be restated as follows. For each row r choose a *reference column* $c(r) \in V_r$. For every pair of columns $c_1 \in V_r$ and $c_2 \in V_r$ such that $c(r), c_1$ and c_2 are distinct, $L(c_1, c_2) = L(c_1, c_r) + L(c_r, c_2)$, where addition is modulo 2.

With this restatement we see that all the constraints on a legal labeling function can be expressed as linear equations over GF[2]. The number of variables is at most $\binom{m}{2}$ and the number of equations is at most $\frac{nm^2}{2}$. In polynomial time, using Gaussian elimination, one can either determine that no solution exists or characterize the set of legal solutions in terms of a set of variables that can be chosen freely, such that their values determine the values of the remaining variables.

The polynomial time bound implied by this description is quite high, but in practice many of the pairs will be of Type 1 or Type 0. The values of the corresponding variables are immediately determined, and further variables can be eliminated easily by a forcing process which eliminates a variable whenever it encounters an equation with one or two undetermined variables.

4 The Build-Tree algorithm

In this section we present an algorithm for the PPH problem which runs in $O(nm^2)$ time. In preparation for our main algorithm we require some definitions and lemmas. Throughout the algorithm we assume that B is an extension of

Strong domination $c \succ c'$	Siblings $c \sim c'$	Weak domination $c \succeq c'$
11	10	10
12	20	20
10	01	22
20	02	
22	22	

Table 1: The table of the relations. Each column in the table represents a different relation. Each column contains a list of pairs that are allowed to appear in B in this relation. Thus, we have (a) $c \succeq c'$ if and only if $c'(B,1) = \phi$ and $c'(B,2) \subseteq c(B,2)$. (b) $c \sim c'$ if and only if $c(B,1) \cap c'(B,1) = c(B,1) \cap c'(B,2) = c(B,2) \cap c'(B,1) = \phi$ and $c \not\succeq c'$. (c) $c \succ c'$ if and only if $c(B,2) \cap c'(B,1) = c(B,0) \cap c'(B,1) = c(B,0) \cap c'(B,2) = \phi$ and $c \not\succeq c'$.

A which is partially assigned, that is, for every $r \in [n], c \in [m]$, if $a_{rc} \leq 1$, then $b_{2r-1,c} = b_{2r,c} = a_{rc}$, and if $a_{rc} = 2$, then either $b_{2r-1,c} = b_{2r,c} = 2$, or $b_{2r-1,c} \neq b_{2r,c}$ and $b_{2r-1,c}, b_{2r,c} \in \{0,1\}$.

We now define relations on the set of columns. The definitions of the relations strong domination, weak domination and siblings are given by table 1. If c and c' are identical, and $c(B,1) \cap c'(B,1) \neq \phi$, we must equally resolve c and c', and thus we can remove one of them from the matrix. It is easy to verify that for any pair of columns c, c', one of the relation holds.

Note that if $c \succ c'$, then c and c' must be equally resolved, and in the phylogenetic tree, c must be an ancestor of c'. If $c \sim c'$, then c and c' must be unequally resolved, and they must be siblings in the phylogenetic tree. The only ambiguous case is when $c \succeq c'$. Furthermore, note that if B can be legally extended, then strong domination must be a transitive and asymmetric relation, and thus it must induce a partial order on the columns. Weak domination must also induce a partial order on the columns.

For any site c, let W_c, S_c and D_c be the set of sites that are weakly dominated by c, siblings to c or strongly dominated by c (and therefore must be descendants of c) respectively, that is $W_c = \{c' \mid c \succeq c'\}$, $S_c = \{c' \mid c \sim c'\}$ and $D_c = \{c' \mid c \succ c'\}$.

4.1 The Main Algorithm

We begin by removing any identical columns c, c' in A, where $c(A,1) \cap c'(A,1) \neq \phi$. We then compute all the pairwise relations between the sites, and we determine that there is no legal extension to A if one of the properties in Lemma 5.1 is violated. and verify that there is no conflict in A. We then generate B from A by duplicating each row of A twice. Next, we assign values to B using the algorithm Build-Tree given in Figure 2. The input to Build-Tree is the matrix $B_{2n \times m}$ which is assumed to be a partial extension of A. The algorithm is recursive. It either returns an assignment for the 2 values in B such that the resulting matrix corresponds to a perfect phylogeny, or it determines that there is no such assignment.

Let $M = \{c | \forall c' \text{ either } c \succ c', c \gtrapprox c' \text{ or } c \sim c'\}$ be the set of maximal columns. Note that if there is no maximal element, there must be a conflict and no legal extension. Choose $\hat{c} \in M$. The algorithm proceeds by considering separately the sets $W_{\hat{c}}, D_{\hat{c}}$, and $S_{\hat{c}}$. Note that there can only be a conflict between a site c and $\hat{c}$ if the values assigned to c and $\hat{c}$ in $c(B,2)$ are assigned incorrectly. When considering the rows of $\hat{c}(B,2)$, it is easy to see that the entries of $D_{\hat{c}}$ and $S_{\hat{c}}$ are uniquely determined assuming B can be legally extended. Each site in $W_{\hat{c}}$ may either be a sibling or a descendant of $\hat{c}$. In order to determine which are the siblings and which are the descendants, we construct a graph $G_{\hat{c}}$, with the set of sites as the set of vertices excluding $\hat{c}$, and the set of edges as the set of pairs of sibling sites c, c', for which $\hat{c}(B,2) \cap c(B,2) \cap c'(B,2) \neq \phi$. It is easy to see that (c, c') share an edge if they cannot both be either descendants or siblings of $\hat{c}$. We check if this graph is bipartite. In Lemma 4.1 we prove that if the graph is not bipartite, then there is no legal extension to A. If it is bipartite, we color it using two colors, where one color corresponds to sites that are siblings to $\hat{c}$, and the other one to sites that are descendants of $\hat{c}$. We check that the

```
ALGORITHM Build-Tree(B)
Input: A {0,1,2} matrix $B$ which is a partial extension to $A$.
Output: A legal assignment to the 2 values of $B$.
  1. Let $M = \{c \mid \forall c'$ either $c \succ c', c \gtrapprox c'$ or $c \sim c'\}$ be the set of maximal columns. If $M = \phi$, then
     there is no legal extension. Otherwise choose $\hat{c} \in M$.
  2. Let $G_{\hat{c}} = (V, E_{\hat{c}})$ be a graph with $W_{\hat{c}} \cup D_{\hat{c}} \cup S_{\hat{c}}$ as the vertices, and the set of edges is $E_{\hat{c}} =$
     $\{(c, c') \mid c \sim c', \hat{c}(B,2) \cap c(B,2) \cap c'(B,2) \neq \phi\}$.
  3. For each row $r \in \hat{c}(B,2)$, assign $b_{2r-1,\hat{c}} = 1$, $b_{2r,\hat{c}} = 0$.
  4. If $G_{\hat{c}}$ is not bipartite report infeasibility and exit.
  5. For each connected component $C$ of $G_{\hat{c}}$ do:
      (a) Color $C$ in two colors, $C = C_0 \cup C_1$.
      (b) If for some $j \in \{0,1\}$, $D_{\hat{c}} \cap C_j \neq \phi$, then set $L(C_j, \hat{c}) = 0$ (equally resolved), and
          $L(C_{1-j}, \hat{c}) = 1$ (unequally resolved).
      (c) If for some $j \in \{0,1\}$, $S_{\hat{c}} \cap C_j \neq \phi$, then set $L(C_j, \hat{c}) = 1$ and $L(C_{1-j}, \hat{c}) = 0$.
      (d) If $L(C_0, \hat{c})$ is assigned both 1 and 0, report infeasibility and exit.
      (e) If $L(C_0, \hat{c})$ is not set at all, arbitrarily set $L(C_0, \hat{c}) = 0$, and $L(C_1, \hat{c}) = 1$.
  6. For each $r \in \hat{c}(B,2), c \in [m]$, $c \neq \hat{c}$ we do:
      (a) If $a_{rc} = 2$, and $L(c, \hat{c}) = 1$ then set $b_{2r-1,c} = 0$, $b_{2r,c} = 1$.
      (b) If $a_{rc} = 2$, and $L(c, \hat{c}) = 0$ then set $b_{2r-1,c} = 1$, $b_{2r,c} = 0$.
  7. Update the relations $\succ, \sim, \succeq$ that were changed in step 6, and remove identical columns $c, c'$ such
     that $c(B,1) \cap c'(B,1) \neq \phi$.
  8. Solve Build-Tree($B \setminus (W_{\hat{c}} \cup \{\hat{c}\})$).
```

Figure 2: Algorithm Build-Tree

colors are consistent with $S_{\hat{c}}$ and $D_{\hat{c}}$. If they are not, then there is no legal extension, and if they are, we assign the values induced by $\hat{c}$ to the entries in the rows of $\hat{c}(B,2)$, and remove $\hat{c}$ and $W_{\hat{c}}$ from the matrix, recursively calling to the algorithm with the rest of the sites. Note that by doing that, we assign values to all the entries of B, since each column that is removed was fully assigned (the nonzero values of the sites in $W_{\hat{c}}$ are only in the rows of $\hat{c}(B,2)$).

We now prove that the algorithm finds an extension if one exists. Clearly, for every $c' \in S_{\hat{c}} \cup D_{\hat{c}}$, there is a unique way to assign the values to c' in the rows of $c(B,2)$, since we are certain whether c and c' have to be equally or unequally resolved. after assigning values to the entries of B in column $\hat{c}$, the assignment to the entries in every site c where $\hat{c} \succ c$ is uniquely determined, since c must be a descendent of c' in the tree, and therefore B must satisfy that for every row r such that $b_{rc} = 1$, then also $b_{r\hat{c}} = 1$. From the same reason, if $r \in S_{\hat{c}}$ (that is $a_{r\hat{c}} = 2$), and $c \sim \hat{c}$, then the assignment to $b_{2r-1,c}, b_{2r,c}$ is uniquely determined.

Note that if $c \in W_{\hat{c}}$, then $c(B,1) = \phi$ for the following reason. Since $\hat{c} \gtrapprox c$, then $\hat{c}$ is either identical to c in which case $c(B,1) = \phi$ since otherwise we remove c, or $\hat{c} \succeq c$. In the latter, if $c(B,1) \neq \phi$, then either $c \sim \hat{c}$ or c and $\hat{c}$ are identical. The following lemma captures the essence of the algorithm.

Lemma 4.1. *If B has a legal extension then $G_{\hat{c}}$ is bipartite.*

Proof. By definition $(c_1, c_2) \in E_{\hat{c}}$, c_1 and c_2 must be unequally resolved. Thus, $G_{\hat{c}}$ cannot contain an odd cycle, and is therefore bipartite. □

By similar arguments to the ones given in Lemma 4.1, we show that the following lemma holds:

Lemma 4.2. *If the algorithm reports infeasibility then there is no legal extension to B.*

Proof. Assume by contradiction that B has a legal extension B'. By Lemma 4.1, $G_{\hat{c}}$ must be bipartite. In step 5 we first color the connected component C using two colors, inferring a partition of C into two sets C_1 and C_2. By the same argument given in the proof of Lemma 4.1, two sites c_1, c_2 have different colors if and only if one of them is a descendant of $\hat{c}$ and the other is sibling to $\hat{c}$ in the phylogenetic tree. Since all sites in $S_{\hat{c}}$ must be siblings of $\hat{c}$ in the tree, and all sites in $D_{\hat{c}}$ must be descendants of $\hat{c}$, it follows that $S_{\hat{c}} \cap C$ must be assigned one color, while $D_{\hat{c}} \cap C$ must be assigned a different color. The algorithm outputs that there is no extension if this is not the case. □

In order to complete the analysis of the algorithm, we show that the algorithm assigns a legal assignment to B.

Lemma 4.3. *If B can be legally extended, then algorithm* Build-Tree *legally extends B into a matrix B'.*

Proof. Assume by contradiction that we have a pair of sites c, c' which have a conflict. By induction, we may assume that either $c = \hat{c}$, or $c \in W_{\hat{c}}, c' \in W_{\hat{c}}$, or $c \in W_{\hat{c}}, c' \in D_{\hat{c}} \cup S_{\hat{c}}$. We note first that for any of these cases, if c and c' satisfy that $B(c, 2) \cap B(c', 2) = \phi$, then c and c' are compatible in any extension of B (assuming that c and c' are compatible in A). We therefore assume that $B(c, 2) \cap B(c', 2) \neq \phi$. Clearly, if $c = \hat{c}$, then c and c' are compatible in B'. Assume now that $c \in W_{\hat{c}}$. Note that $c(B, 1) = \phi$. Therefore, if $r \in c(B', 1) \cap c'(B', 1)$, then one of two cases can be true:

1. $r \in c(B, 2) \cap c'(B, 2)$. In this case, we must have that c and c' has the same color, and therefore cannot share an edge. Thus, c and c' cannot be sibling. It is then easy to see that c and c' are compatible in B'.
2. $r \in c(B, 2) \cap c'(B, 1)$. In this case we get that $r \in \hat{c}(B, 1)$, and therefore, $r \in c(B, 0)$ which is a contradiction.

□

In order to implement the algorithm efficiently, we first pre-compute all the relations between every pair of sites, which takes $O(nm^2)$ time. After the preprocessing, each time we call the algorithm, we have to compute the edges of the graph and to update the relations, which takes $O(|\hat{c}(B, 2)|m^2)$ time, but since the sets $\hat{c}(B, 2)$ are disjoint, the sum of these is at most $O(nm^2)$. Furthermore, we have to find the maximal site $\hat{c}$, and color the graph using two colors. This may take $O(|E_{\hat{c}}|)$ time, which might be as large as $O(m^2)$, but since we only do that at most m times, the total running time is $O(m^2(m + n))$.

In order to get all the possible extensions, we can change the algorithm as follows. Note that the only times where the assignment is not uniquely determined is when we have a connected component C which is fully contained in $W_{\hat{c}}$. We introduce a boolean variable x_C for any connected component. Let C_0, C_1 be the two color sets of C. Let $c \in C_0, c' \in C_1$. For every $r \in \hat{c}(B, 2) \cap c(B, 2)$, set $b_{2r-1,c} = x, b_{2r,c} = \bar{x}$, and every $r \in \hat{c}(B, 2) \cap c'(B, 2)$ set $b_{2r-1,c'} = \bar{x}, b_{2r,c'} = x$. It is easy to see that every possible solution corresponds to a $\{0, 1\}$ assignment to the boolean variables x_C, for every connected component C.This gives a non-trivial bound of 2^{m-1} to the number of possible solutions. This bound is tight since when all the entries of A equal 2, we have exactly 2^{m-1} possible solutions (up to symmetry).

5 Adding Families to the Data

In many cases, the experimental studies are done on related individuals, such as a set of trios of mother, father and a child. In this case, in addition to the coalescent model, we have the additional constraint that each of the parents transmits exactly one of its haplotypes to the child. In this section we show that this extension to the PPH problem can be solved in polynomial time. Formally, we solve the following problem:

The Trios PPH problem (TPPH). The input is a $\{0,1,2\}$ matrix $A = (a_{ij})$ of size $n \times m$ and a set of triplets $T \subseteq [n]^3$. Every triplet $(r_1, r_2, r_3) \in T$ represents a mother father and child trio. We need to determine if there is a legal extension $B = (b_{ij})$ to A such that for every triplet $(r_1, r_2, r_3) \in T$, one of the rows $B(2r_3 - 1, :), B(2r_3, :)$ is a duplicate of one of the rows $B(2r_1 - 1, :), B(2r_1, :)$, and the other is a duplicate of one of the rows $B(2r_2 - 1, :), B(2r_2, :)$. This correspond to the fact that one of the haplotypes is transmitted from the mother and the other from the father.

5.1 Solving TPPH

Recall that the output of algorithm *Build-Tree* is a matrix B which corresponds to all possible solutions to the coalescent model. For each $i \in [2n], j \in [m]$, either $b_{ij} \in \{0,1\}$, or $b_{ij} \in \{x_1, \ldots, x_k, \bar{x_1}, \ldots, \bar{x_k}\}$, where x_i is a boolean variable for each $i \in [k]$. We have to set the values of the variables, so that the solution will be consistent with the trios.

For ease of notation, for every triplet $t = (r_1, r_2, r_3) \in T$, and every column i, we say that the (t, i) configuration is $(z_1, z_2, z_3, z_4, z_5, z_6)$ if $z_1 = b_{2r_1-1,i}, z_2 = b_{2r_1,i}, z_3 = b_{2r_2-1,i}, z_4 = b_{2r_2,i}, z_5 = b_{2r_3-1,i}, z_6 = b_{2r_3,i}$, that is, the values z_1, z_2, z_3, z_4, z_5 and z_6 represent the values of the mother, father and child's haplotypes. We first need the following lemma. Its proof will be given in the final version of the paper.

Lemma 5.1. *For every triplet $t \in T$ and every column $c \in [m]$, the following (t, c) configurations never appear in B:*

1. $(1, 1, *, *, x, \bar{x})$ *for $x \in \{x_1, \ldots, x_k, \bar{x_1}, \ldots, \bar{x_k}\}$, where $*$ represent an arbitrary value. Any permutations of the values between the mother, father and child does not appear either.*
2. $(x, \bar{x}, y, \bar{y}, *, *)$ *where $x \in \{x_1, \ldots, x_k, \bar{x_1}, \ldots, \bar{x_k}\}$, and $x \neq y$. Any permutations of the values between the mother, father and child does not appear either.*

Proof. 1. If the (t, i) configuration is $(1, 1, *, *, x, \bar{x})$, then $c(1) \neq \phi, c(2) \neq \phi$. Assume that $x = x_C$ for some connected component, where $\hat{c}$ is the maximal site at that point in the algorithm. Then, $\hat{c} \gtrapprox c$, and thus, $c(1) = \phi$, which is a contradiction.

2. Since each variable correspond to a color in a connected component, and every site belongs only to one color in one connected component throughout, then if $y \in \{x_1, \ldots, x_k, \bar{x_1}, \ldots, \bar{x_k}\}$, we must have that $x = y$. Since all the 2-values of i are assigned at the time when i is weakly dominated by $\hat{c}$, then it is impossible that $y \in \{0, 1\}$.

$\square$

For each triplet $t = (r_1, r_2, r_3) \in T$ we introduce three additional boolean variables $p(t), y_1(t), y_2(y)$ which have the following values:

- $p(t) = 1$ if and only if the haplotype $B(2r_3 - 1, :) = B(2r_1, :)$ or $B(2r_3 - 1, :) = B(2r_1 - 1, :)$. In other words, $p(t) = 1$ if and only if the first haplotype of r_3 is transmitted from the mother.

- $y_1(t) = 1$ if and only if $B(2r_3 - 1, :) = B(2r_1 - 1, :)$ or $B(2r_3 - 1, :) = B(2r_2 - 1, :)$, that is, if the first haplotype of the child is transmitted from a first haplotype of one of the parents.

- $y_2(t) = 1$ if and only if $B(2r_3, :) = B(2r_1 - 1, :)$ or $B(2r_3, :) = B(2r_2 - 1, :)$, that is, if the second haplotype of the child is transmitted from a first haplotype of one of the parents.

Whenever it is clear from the context, we will omit t, and just write p, y_1, y_2 instead of $p(t), y_1(t), y_2(t)$. It is easy to see that any $\{0,1\}$ assignment to the variables $p(t), y_1(t), y_2(t)$ corresponds to one of the eight possible transmissions of the haplotypes from the parents to the child. We thus get that the TPPH problem is equivalent to assigning $\{0,1\}$ values to the variables $x_1, \ldots, x_k$, and the variables $p(t), y_1(t), y_2(t)$ for $t \in T$. Every triplet $t \in T$, and every column $j \in [m]$, impose a constraint on the variables. For example, if the mother has 0 values in

1	(z, z, z, z, z, z)	No constraint
2	$(0, 0, x, \bar{x}, 0, 0)$	$(\bar{x} \vee p \vee y_2) \wedge (\bar{x} \vee \bar{p} \vee y_1) \wedge (x \vee p \vee \bar{y_2}) \wedge (x \vee \bar{p} \vee \bar{y_1})$
3	$(x, \bar{x}, 0, 0, 0, 0)$	$(\bar{x} \vee \bar{p} \vee y_2) \wedge (\bar{x} \vee p \vee y_1) \wedge (x \vee \bar{p} \vee \bar{y_2}) \wedge (x \vee p \vee \bar{y_1})$
4	$(z, \bar{z}, z, z, z, z)$	$(\bar{p} \vee y_1) \wedge (p \vee y_2)$
5	$(z, \bar{z}, z, \bar{z}, z, z)$	$y_1 = y_2 = 1$
6	$(x, \bar{x}, x, \bar{x}, 0, 0)$	$y_1 = y_2 = \bar{x}$
7	$(z, \bar{z}, \bar{z}, z, z, z)$	$p = y_1 = \bar{y_2}$
8	$(z, z, \bar{z}, \bar{z}, z, \bar{z})$	$p = 1$
9	$(x, \bar{x}, 0, 0, x, \bar{x})$	$(x \wedge p \wedge y_1) \vee (\bar{x} \wedge \bar{p} \wedge \bar{y_2})$
10	$(z, z, z, \bar{z}, z, \bar{z})$	$p \wedge \bar{y_2}$
11	$(z, \bar{z}, z, \bar{z}, z, \bar{z})$	$y_1 \wedge \bar{y_2}$
12	$(x, \bar{x}, x, \bar{x}, x, \bar{x})$	$y_1 \wedge \bar{y_2}$
13	$(\bar{z}, z, z, \bar{z}, z, \bar{z})$	$y_1 = y_2 = \bar{p}$

Table 2: The possible constraints up to symmetry. z represents any value in $\{0, 1\}$. x represents a literal, that is, $x \in \{x_1, \ldots, x_k, \bar{x_1}, \ldots, \bar{x_k}\}$.

both haplotypes, the father has 1 in both haplotypes, and the child has 1 in the first haplotype, and 0 in the second haplotype, then clearly, $p(t) = 1$. We will now show that almost all these constraints can be represented as a 2-CNF formula, and thus, can be solved using any polynomial algorithm known for 2-SAT. The cases where the constraints cannot be represented by a 2-CNF formula will be considered separately. In Table 5.1 we list all possible constraints which follow from the (t, c) configurations. It is easy to verify by Lemma 5.1, that all the possible configurations are listed in the table, up to symmetry. Note that apart from constraints 2 and 3, all other constraints can be expressed by a 2-CNF formula, that is, the logical and of clauses, where each clause is the logical or of two literals.

For a triplet $t \in T$ and a column $i \in [m]$, we say that the configuration (t, i) is of type j for $j \leq 15$ if the configuration corresponds to the j-th row in Table 5.1. We denote it by $type(t, i) = j$. The following claim is proved by case analysis:

Claim 5.2. *Let $t \in T$, and let $i, j \in [m]$. If $type(t, i) \in \{2, 3\}$, and $type(t, j) > 3$. then one can express an equivalent constraint to the constraint inferred by (t, i) using a 2-CNF formula.*

We need the following definition.

Definition 5.3. *We call a triplet $t \in T$* special *if one of the following hold:*

1. *For every $i \in [m]$, $type(t, i) \in \{1, 2, 4, 6, 8\}$. In this case t is special of type 1.*
2. *For every $i \in [m]$, $type(t, i) \in \{1, 3, 5, 7, 9\}$. In this case t is special of type 2.*
3. *For every $i \in [m]$, $type(t, i) \in \{1, 2, 3\}$. In this case t is special of type 3.*

We now describe an algorithm for solving the TPPH problem. We first use algorithm Build-Tree to get all possible solutions to the PPH problem. We call a triplet $t \in T$ *special* if $type(t, i) \in \{1, 2, 3\}$ for every $i \in [m]$. We now construct a set C of constraints in the following way. We initialize C to be the set of constraints induced by all the triplets that are not special. We then consider each special triplet and add constraints to C. Let $t \in T$ be special triplet. First note that if $i, j \in [m], x, y \in \{x_1, \ldots, x_k\}$ such that the configuration in (t, i) is $(x, \bar{x}, 0, 0, 0, 0)$ and the configuration in (t, j) is $(y, \bar{y}, 0, 0, 0, 0)$ then necessarily $x = y$. Therefore, in this case we add the constraint $x = y$ to C.

By Claim 5.2 we can express all the constraints in C by a 2-CNF formula. We can therefore find a legal assignment to the subproblem induced by C by using any of the algorithms known for 2-SAT [1, 4, 19, 16]. If there is no feasible solution, we report that the problem is infeasible. Otherwise, the solution induces an assignment for a subset X of the variables $\{x_1, \dots, x_k\}$. We arbitrarily set the values of the set $\{x_1, \dots, x_k\} \setminus X$ to zero.

We now show that we can extend this solution to a feasible solution on the special triplets. Let $t \in T$ be a special triplet. It is easy to see that if we find a feasible solution to C, then the set of possible configurations in t contains at most four possible configurations out of $(1,1,1,1,1,1), (0,0,0,0,0,0), (0,0,z,\bar{z},0,0)$ and $(z', \bar{z'}, 0,0,0,0)$ for fixed $z, z' \in \{0,1\}$. It is easy to verify that for each such four configurations one could find a valid assignment to $p(t), y_1(t)$ and $y_2(t)$.

6 Minimum Genotype Removal

In practice, the biological data does not exactly fit the coalescent model. We therefore pose the problem of removing the minimal number of individuals from our data set so that the remaining data fits the coalescent model. Formally, we introduce the following problem:

The Minimum Genotype Removal Problem . The input is a $\{0,1,2\}$ matrix $A = (a_{ij})$ of dimensions $n \times m$. The goal is to remove the minimal number of rows from A such that the remaining rows fit the coalescent model, with the all zeros root.

Clearly, algorithm Build-Tree shows that one can determine in polynomial time if the minimal number of rows is zero. In fact, if the input to the problem is a $\{0,1\}$ matrix A, then the problem can be approximated within a factor of 3 by a local ratio argument. Note that since we know that the root is the all zeros vector, then the matrix fits the model if and only if there is no sub-matrix defined by three rows and a pair of columns such that the rows of the sub-matrix contain the pairs $(1,1), (1,0), (0,1)$. While there is such a sub-matrix, we simply remove the corresponding three rows. Eventually we will be left with a matrix with no conflict. Since for every three rows that we removed at least one of them should be removed by the optimal solution, we get that this is a 3-approximation to the problem.

6.1 The Hardness Result

In this section we show that the minimum genotype removal problem is at least as hard to approximate as the min UnCut problem, which is a well studied optimization problem. In the minimum UnCut problem we are given a graph $G = (V, E)$, and we wish to find a minimum size set of edges such that by their removal we are left with a bipartite graph. This problems has a $\log n$ approximation algorithm [15], where n is the number of vertices in G. It is only known to be MAX-SNP hard [17]. We now prove the following theorem:

Theorem 6.1. *If the minimum genotype removal problem has an α-approximation algorithm, then one can find an α-approximation algorithm for the minimum UnCut problem.*

Proof. If $\alpha > \log n$, then the theorem trivially holds. We thus assume that $\alpha \le \log n$. Let $\mathcal{A}$ be an α-approximation algorithm for the minimum genotype removal problem. Let $G = (V, E)$ be a graph, where $V = \{1, \dots, n\}$, $E = \{e_1, \dots, e_m\}$. We construct the following matrix $A = (a_{ij})$ as an input to $\mathcal{A}$. A will be of dimension $m(n+1)\alpha \times (n+1)$. The entries of A will have the following values:

1. For every $i \in [m(n+1)\alpha]$, $a_{i,n+1} = 2$. (The last column contains only values of 2).
2. For every $i \in [n], j \in [m\alpha - 1]$, $a_{i \cdot m \cdot \alpha - j, i} = 2$.
3. For every $i \in [m]$, if $e_i = (j,k)$, then $a_{i+mn,j} = 2, a_{i+mn\alpha,j} = 2$.
4. Any other entry of A is zero.

By property 2, for every $i \neq j$, $i, j \in [n]$, we have that i and j are siblings. By property 1, column $n+1$ is the maximal column. It is easy to see that the graph G_{n+1} constructed in algorithm Build-Tree is isomorphic to G. From that, it is easy to see that the solution returned by $\mathcal{A}$ corresponds to an α-approximation to the Min UnCut problem.

A more detailed proof will be given in the final version of the paper. there is a one to one correspondence between the Each of the last m rows of A corresponds to an edge in G. Furthermore, by removing a subset of the rows of A so that G_{n+1} becomes bipartite, we are left with a matrix which fits the coalescent model.

Let R be the set of rows of A. Let $OPT \subseteq E$ be the optimal solution to the min UnCut problem, and let $OPT' \subseteq R$ be the optimal solution for the minimum haplotype removal problem. Clearly, given a solution $S \subseteq E$ to the min UnCut problem on G one can construct a solution to the minimum haplotype removal simply by removing the rows in R that correspond to S. Thus, $|OPT'| \leq |OPT| \leq m$. $\mathcal{A}$ returns a set $S' \subseteq R$ of size at most $|OPT'|\alpha \leq m \log n$. Thus, since every row in R apart from the last m rows appears $m \log n$ times, we can assume that S' is a subset of the last m rows of A. We now remove from G every edge which corresponds to a row in S', and it is easy to see that the resulting graph is bipartite, and that the number of edges removed is $|S'| \leq |OPT'|\alpha \leq |OPT|\alpha$, and thus we get an α-approximation algorithm to the minimum haplotype removal problem. $\square$

7 Experimental Results

We performed our experiments over the data presented in Daly et al., 2001 [5]. In their study they predicted 11 blocks over the 103 SNPs they examined. They used a set of 129 mother-father child trios, and thus, most of the haplotype bases are uniquely determined by these relations under the assumption of perfect Mendelian heredity. We examined the predictions of algorithm Build-Tree over their blocks by only using the children genotypes (see Table 3). We used the methods in [11] to fit the model to the coalescent model whenever needed. Our results show an extremely small error rate. This experiment proves that using our algorithm, the study presented in [5] could have been done using the children alone. A more concise experimental work based on our algorithms and some extensions can be found in [11].

SNPs	Actual Common Haplotypes	Predicted Common Haplotypes	Frequency	Error Rate
1-8	GGACAACC	GGACAACC	215	0
	AATTCGTG	AATTCGTG	38	
10-14	TTACG	TTACG	217	0
	CCCAA	CCCAA	35	
16-24	CGGAGACGA	CGGAGACGA	139	0.005780
	GACTGGTCG	GACTGGTCG	52	
	CGCAGACGA	CGCAGACGA	34	
		CGGATACGA	15	
25-35	CGCGCCCGGAT	CGCGCCCGGAT	142	0
	CTGCTATAACC	CTGCTATAACC	39	
	TTGCCCCGGCT*	CTGCCCCGGCT	35	
	CTGCCCCAACC*	TTGCCCCAACC	25	
36-40	CCAGC	CCAGC	146	0
	CCACC	CCACC	51	
	GCGCT	GCGCT	30	
	CAACC	CAACC	12	
41-45	CCGAT	CCGAT	152	0.011561
	CTGAC	CTGAC	63	
	ATACT	ATACT	31	
78-84	CGTTTAG	CGTTTAG	142	0
	TGTT*GA	TGTTTGA	53	
	TGATTAG	TGATTAG	20	
	CGTCTAG	CGTCTAG	12	
		TGTTGGA	10	
86-91	ACAACA	ACAACA	145	0.007353
	GCGGTG	GCGGTG	71	
	ACGGTG	ACGGTG	14	
	GTGACG	GTGACG	13	
92-98	GTTCTGA	GTTCTGA	142	0.004132
	TGTGTAA	TGTGTAA	49	
	TG*GCGG	TGTGCGG	32	
		TGCGTAA	15	
99-103	CGGCG	CGGCG	112	0.003448
	TATAG	TATAG	105	
	TATCA	TATCA	35	

Table 3: Predictions over data from Daly et al. 2001, [5]. The second column shows the common haplotypes as presented in Daly et al. 2001 as well as their frequencies. The third column shows the predictions and the fourth gives their frequencies. The fifth column shows the error ratio in our predictions.

8 Concluding Remarks

We presented a practical and efficient algorithm to infer haplotype structure from genotype data. We furthermore extended our algorithm to cope with further constraints. We presented relations between classical combinatorial problems and the suggested biological problem. Our experiments show that our algorithms are practical, and could save time and money in biological experiments. We believe that further extensions to this problem could lead to even more accurate haplotype reconstruction on a larger scale. Specifically, coping with errors in the data and with missing data is left as an open problem.

References

[1] M.F. Plass B. Aspval and R.E. Tarjan. A linear time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letter*, 8:121–123, 1979.

[2] R. E. Bixby and D. K. Wagner. An almost linear time algorithm for graph realization. *Mathematics of Operations Research*, 13:99–123, 1988.

[3] AG Clark. Inference of haplotypes from pcr-amplified samples of diploid populations. *Journal of Molecular Biology and Evolution*, 7(2):111–22, Mar 1990.

[4] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[5] MJ Daly, JD Rioux, SF Schaffner, TJ Hudson, and ES Lander. High-resolution haplotype structure in the human genome. *Nature Genetics*, 29(2):229–32, Oct 2001.

[6] L Excoffier and M Slatkin. Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. *Molecular Biology and Evolution*, 12(5):921–7, Sept 1995.

[7] D Fallin and NJ Schork. Accuracy of haplotype frequency estimation for biallelic loci, via the expectation-maximization algorithm for unphased diploid genotype data. *American Journal of Human Genetics*, 67(4):947–59, Oct 2000.

[8] D. Gusfield. A practical algorithm for optimal inference of haplotypes from diploid populations. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, 2000.

[9] D. Gusfield. nference of haplotypes from samples of diploid populations: complexity and algorithms. *Journal of Computational Biology*, 8(3):305–23, 2001.

[10] D. Gusfield. Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions (extended abstract). In *Proceedings of the 6th International Conference on Computational Molecular Biology (RECOMB 2002)*, 2002.

[11] E. Halperin and E. Eskin. Large scale recovery of haplotypes from genotype data using imperfect phylogeny. *Unpublished Manuscript*, 2002.

[12] ME Hawley and KK Kidd. Haplo: a program using the em algorithm to estimate the frequencies of multi-site haplotypes. *Journal of Heredity*, 86(5):409–11, Sep-Oct 1995.

[13] G. Lancia, V. Bafna, S. Istrail, R. Lippert, and R. Schwartz. Snps problems, algorithms and complexity, european symposium on algorithms. In Springer-Verlag, editor, *Proceedings of the European Symposium on Algorithms (ESA-2001), Lecture Notes in Computer Science*, volume 2161, pages 182–193, 2001.

[14] J.C. Long, R.C. Williams, and M. Urbanek. An e-m algorithm and testing strategy for multiple-locus haplotypes. *American Journal of Human Genetics*, 56(3):799–810, Mar 1995.

[15] V.V. Vazirani N. Garg and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM J. Comp*, 25:235–251, 1996.

[16] C.H. Papadimitriou. On selecting a satisfying truth assignment. *FOCS*, pages 163–169, 1991.

[17] C.H. Papadimitriou and M.Yannakakis. Optimization, approximation and complexity classes. *JCSS*, 43:425–440, 1991.

[18] N Patil, AJ Berno, DA Hinds, WA Barrett, JM Doshi, CR Hacker, CR Kautzer, DH Lee, C Marjoribanks, DP McDonough, BT Nguyen, MC Norris, JB Sheehan, N Shen, D Stern, RP Stokowski, DJ Thomas, MO Trulson, KR Vyas, KA Frazer, SP Fodor, and DR Cox. Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. *Science*, 294(5547):1719–23, Nov 23 2001.

[19] A. Itai S. Even and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SICOMP*, 5:691–703, 1976.

[20] M. Stephens, N. Smith, , and P. Donnelly. A new statistical method for haplotype reconstruction from population data. *American Journal of Human Genetics*, 68:978–989, 2001.

[21] W.T. Tutte. An algorithm for determining whether a given binary matroid is graphic. *Proc. of Amer. Math. Soc.*, 11:905–917, 1960.