Haplotype Inference Constrained by Plausible Haplotype Data

Michael R. Fellows[1], Tzvika Hartman, Danny Hermelin[⋆2], Gad M. Landau[2,3], Liat Leventhal[2], and Frances Rosamond[1]

[1] The University of Newcastle, Callaghan NSW 2308 - Australia
{mike.fellows,frances.rosamond}@cs.newcastle.edu.au
[2] Department of Computer Science, University of Haifa, Haifa - Israel.
{danny,liat}@cri.haifa.ac.il, landau@cs.haifa.ac.il
[3] Department of Computer and Information Science, Polytechnic University, New York - USA.

Abstract. The *haplotype inference problem* (HIP) asks to find a set of haplotypes which resolve a given set of genotypes. This problem is of enormous importance in many practical fields, such as the investigation of diseases, or other types of genetic mutations. In order to find the haplotypes that are as close as possible to the real set of haplotypes that comprise the genotypes, two models have been suggested which by now have become widely accepted: The *perfect phylogeny* model and the *pure parsimony* model. All known algorithms up till now for the above problem may find haplotypes that are not necessarily plausible, *i.e.* very rare haplotypes or haplotypes that were never observed in the population. In order to overcome this disadvantage we study in this paper, for the first time, a new constrained version of HIP under the above mentioned models. In this new version, a pool of plausible haplotypes $\widetilde{H}$ is given together with the set of genotypes G, and the goal is to find a subset $H \subseteq \widetilde{H}$ that resolves G. For the *constrained perfect phylogeny haplotyping* (CPPH) problem we provide initial insights and polynomial-time algorithms for some restricted cases that help understanding the complexity of that problem. We also prove that the *constrained parsimony haplotyping* (CPH) problem is fixed parameter tractable by providing a parameterized algorithm that applies an interesting dynamic programming technique for solving the problem.

1 Introduction

Genetic information in living organisms is encoded in DNA sequences that are organized into *chromosomes*. Diploid organisms such as humans have two copies of every chromosome, which are not necessarily identical, with each copy called a *haplotype*. Identifying the common genetic variations that occur in humans are valuable in understanding diseases [1]. The genetic sequences of the population are almost totally identical, except from some bases that differ from one person to another with a frequency of more than some threshold (1% for example). Those differences are the common genetic variations and they known as *single nucleotide polymorphisms* (SNPs).

The data described in each haplotype may be the full DNA, but it is more common to consider only the data of the SNPs since the other sites are assumed to be identical. A *genotype* is the description of the two copies (haplotypes) together. When the two haplotypes agree, the site in the genotype has the agreed base. Such a site is called a *homozygous* site. When the two haplotypes disagree, the genotype has both bases, yet it does not tell which base occur in which haplotype. This type of site is called *heterozygous* (see figure 1(a)).

Current biological technologies give us an easier and cheaper way to obtain genotype data in comparison to haplotype data. However, the haplotype information is the one of greater use [15]. For this reason, it is necessary to computationally infer the haplotype information from the genotype data. An important biological fact is that almost always there are only two bases at a SNP, which

⋆ Supported by the Adams Fellowship of the Israel Academy of Sciences and Humanities.

can be marked as 0 and 1. A genotype will have 0 or 1 if the two haplotypes both have 0 or 1 in the same site respectively, or 2 otherwise (see figure 1(b)). In view of that, a set of genotypes and a set of haplotypes can be represented as matrices. A *genotype matrix* is a matrix over $\{0, 1, 2\}$ where each row is a genotype and each column represents SNP, and a *haplotype matrix* is a matrix over $\{0, 1\}$, where each row is a haplotype and each column represents SNP.

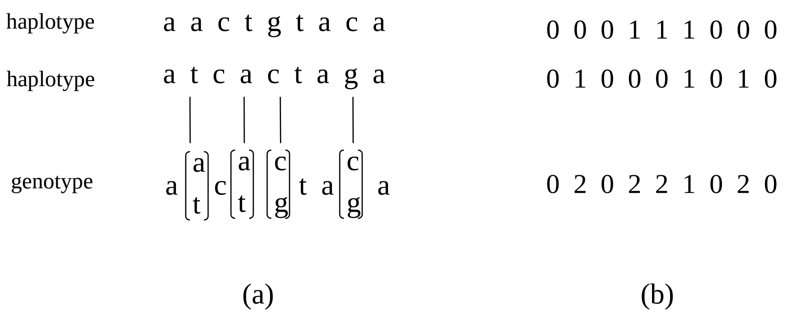


Fig. 1. (a) Example of two haplotypes and the corresponding genotype. (b) The same haplotypes and genotype using the 0,1,2 representation where a=0, t=1, c=0, g=1.

For the rest of the paper, let $g(i)$ represent the data at site i of genotype g, and $h(i)$ the data at site i of haplotype h.

Definition 1 (Resolution). *A pair of haplotypes $\{h, h'\}$ is said to* resolve *g if for each i: $g(i) = h(i)$ where $h(i) = h'(i)$, and $g(i) = 2$ otherwise. We extend this and say that a set of haplotypes H* resolves *a set of genotypes G, if for each $g \in G$, there is a pair $\{h, h'\} \in H$ which resolves g. The pair $\{h, h'\}$ is called a* resolution *of g, and H is* resolution *of G.*

Definition 2 (Haplotype Inference Problem (HIP)). *Given a set of n genotypes G, each of length m, find a resolution of G.*

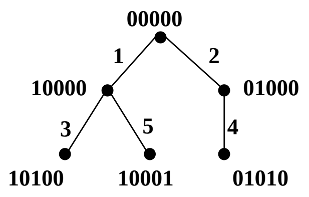
Note that if a genotype has $d \leq m$ heterozygous sites (sites marked with 2), then the number of possible resolving pairs is 2^{d-1}. The goal is to find the set of pairs which as close as possible to the real set of haplotypes that created the genotype. Currently, there are two models used in practice that give two different biologically motivated heuristics on how to determine this:

1. **Perfect Phylogeny:** The perfect phylogeny model is a coalescent model which assumes no recombination. This means that the history of the haplotypes is represented as a tree where two haplotypes from two individuals have at most one resent common ancestor [15] (see [13, 15, 20, 27] for further information). Formally, a set of haplotypes (binary sequences) of length m *defines a perfect phylogeny* if the haplotypes appear as labels of a rooted tree which obeys the following properties [10]:
   – Each vertex of the tree is labeled by a binary sequence of length m representing a possible haplotype;
   – Every edge (u, v) is marked with i, where the base at site i in sequence u is different from the one in sequence v. Every coordinate i labels at most one edge.

A common way of checking whether a set of haplotypes defines a perfect phylogeny is to check whether it obeys the *four gamete test*, *i.e.* the corresponding haplotype matrix does not contain, in any two columns, the *forbidden gamete submatrix*

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

See Figure 2 for an example of a perfect phylogenetic tree, and the corresponding haplotype matrix. The *Perfect Phylogeny Haplotyping* (PPH) problem is the problem of finding for a given set of genotypes a resolution which defines a perfect phylogeny, if such resolution exists.



$$H = \begin{bmatrix} 01000 \\ 01010 \\ 10100 \\ 10001 \end{bmatrix}$$

Fig. 2. Example of a perfect phylogenetic tree for the haplotypes $h_1 = (01000), h_2 = (01010), h_3 = (10100), h_4 = (10001)$. H is the haplotype matrix of the above haplotypes, which obeys the four gamete test.

2. **Pure Parsimony:** The pure parsimony model seeks for the minimum set of haplotypes that resolves a given set of genotypes. The biological motivation behind this is the statistical observation that the number of distinct haplotypes in the population is vastly small [14, 15]. The *Parsimony Haplotyping* (PH) problem is the problem of finding a resolution of smallest size possible for a given set of genotypes.

In [13], Gusfield showed that the PPH problem is solvable in $O(nm\alpha(nm))$ time, where α is the inverse Ackerman function. Gusfield also showed a linear-time algorithm to build, once the first solution is found, a linear-space data structure that represents all PPH solutions. However, his work is based on complex graph-theoretic algorithms which are difficult to implement [15]. In [2, 10], algorithms fine-tuned to the actual combinatorial structure of the PPH problem were shown. These algorithms run in $O(nm^2)$ time and are easy to understand and implement. They also give a representation of all PPH solutions. More recent work developed $O(nm)$ time algorithms: In [8], the algorithm is graph-theoretic and uses a directed rooted graph called a "shadow tree", and in [25], the algorithm is based on interdependencies among the pairs of SNPs, and builds a data structure called "FlexTree" to represent all PPH solutions. Other works have researched different variations of PPH [3, 6, 7, 11, 12, 18, 19, 22].

The parsimony haplotyping (PH) problem was probably first suggested and proved to be NP-hard by Earl Hubell (unpublished). Gusfield formally introduced the problem in [14], and proposed an integer linear programming solution. More integer linear programming solutions following this were proposed in [4, 14, 17, 23]. Approximation algorithms for the problem were presented in [16, 23, 24], and in [28], a branch-and-bound algorithm was proposed. Other theoretical results were shown in [5, 21, 23, 26]. Most notably is the work of Sharan *et. al* [26], who characterized restricted instances of PH under the term $(\alpha, \beta) - \textit{bounded}$, where α and β stand for the maximum number of heterozygous sites per row and column of the genotype matrix. Sharan *et al.* also showed that the PH problem is *fixed parameter tractable* (see [9] for formal definition) when parameterized by the number of k haplotypes in the resolution of G.

All known algorithms up till now for haplotype inference under the perfect phylogeny model find resolutions for a given set of genotypes from the superset of all possible haplotypes (*i.e.* all m-length binary vectors). However, these algorithms may find resolutions that include binary vectors representing haplotypes that do not actually occur in the population, or are otherwise very rare. It is therefore biologically interesting to force the resolving haplotypes to be chosen only from a specific pool that contains only *plausible* haplotypes, *i.e.* haplotypes which have already been observed in relatively high frequencies at previous experiments. This pool can be determined by empirically setting up some statistical threshold, or by any other reasonable method.

In view of all this, we study here for the first time, a new constrained variant of the haplotype inference problem, in which a pool of plausible haplotypes $\widetilde{H}$ is given alongside the set of genotypes G, and the goal is to find a resolution of G which is a subset of $\widetilde{H}$.

Definition 3 (Constrained Haplotype Inference Problem (CHIP)). *Given a set of ℓ genotypes G, each of length m, and a pool of n plausible haplotypes $\widetilde{H}$ for G, each of length m, find a resolution $H \subseteq \widetilde{H}$ of G.*

The constrained perfect phylogeny haplotyping (CPPH) problem and the constrained parsimony haplotyping (CPH) problem are defined accordingly. Note that if $\ell > n(n-1)$ in the above definition, there is no solution automatically, since taking the entire pool of n plausible haplotypes we can resolve at most $n(n-1)$ genotypes. On the other hand, there is no inequality necessarily in the other direction. We therefore assume $\ell \leq n(n-1)$ throughout the paper.

Note that while the complexity of CPH can easily be deferred from the complexity of PH, this is not the case for CPPH. Indeed, while the PPH problem is, as stated, linear-time solvable, it is still open whether CPPH is even polynomial-time solvable. Most of the current techniques for solving PPH are based on dependencies and constrains between pairs of sites, and the property that these can be expressed as a set of linear equations over GF[2] (see Eskin *et al.* [10]). The new dependencies between sites that arise from the given pool of plausible haplotypes are of a different type, and there seems to be no easy way to express them over GF[2]. On the other hand, all attempts of showing that CPPH is NP-hard by standard straightforward reductions from PH or MPPH [10] (*Minimum Pure Parsimony Haplotyping* - the problem of finding a perfect phylogenic resolution of minimal size) eventually fail due to the fact that $\widetilde{H}$ is given alongside the input (and therefore is required to be polynomial), and since any solution (not necessarily minimal) is acceptable.

In this paper, we provide an initial insight to determining the complexity of CPPH. We present a framework which helps partially answering this question, and allows polynomial-time solutions for the CPPH (α, β) bounded cases of (*,1), (2,*), (5,2), and (3,3). As is the case for the PH problem [10, 26] these cases can be very useful for speeding up implementations of CPPH, but they also give a glimpse into the complexity of the problem. For example, while it was proved in [10] that MPPH and PH for (3,3)-bounded genotype matrices are both APX-hard, we show that CPPH is polynomial-time solvable in this case.

In the second part of the paper we turn to consider CPH. We show that like PH [10], CPH is fixed-parameter tractable when parameterized by the number of haplotypes k in a minimum resolution $H \subseteq \widetilde{H}$ of G. The parameterized algorithm for CPH, is however much more involved than the one for PH, and it applies an interesting dynamic programming technique for solving the problem.

2 Constrained Perfect Phylogeny Haplotyping

In this section we describe polynomial-time algorithms for CPPH with genotype matrices of specific structures. In [26], bounded cases of genotype matrices were introduced in order to explore the complexity of PH. The bounded cases were defined as follows:

Definition 4 ((α,β)-bounded [26]). *A genotype matrix G is* (α,β)-bounded *if it has at most α sites with 2 per row and at most β sites with 2 per column. α and β might be * which means there is no bound on the number of 2s per row or column, respectively.*

Here we use the same term of (α,β)-bounded to present polynomial-time algorithms for special cases of CPPH. We will present algorithms for the following cases: (*,1), (2,*), (5,2) and (3,3).

We begin with the following lemma which lists ten matrices that we can assume G does not include, since including any one of them implies that all resolutions of G necessarily include the forbidden gamete submatrix. Its proof is left to the reader.

Lemma 1. *If G includes one of the following 2×3 submatrices:*

$$\begin{pmatrix} 2&0\\0&2\\1&1 \end{pmatrix}, \begin{pmatrix} 2&1\\1&2\\0&0 \end{pmatrix}, \begin{pmatrix} 2&0\\1&2\\0&1 \end{pmatrix}, \begin{pmatrix} 2&1\\0&2\\1&0 \end{pmatrix}, \begin{pmatrix} 2&1\\0&0\\1&0 \end{pmatrix}, \begin{pmatrix} 1&2\\0&0\\0&1 \end{pmatrix}, \begin{pmatrix} 2&0\\1&1\\0&1 \end{pmatrix},\ \textit{or}\ \begin{pmatrix} 0&2\\1&1\\1&0 \end{pmatrix},$$

or one of the following 2×2 submatrices:

$$\left(\begin{smallmatrix} 2&0\\2&1 \end{smallmatrix}\right)\ \textit{or}\ \left(\begin{smallmatrix} 0&2\\1&2 \end{smallmatrix}\right),$$

then G does not have a perfect phylogenic resolution.

As in Eskin *et al.* [10], we will be working with pairs of columns in G. Pairs of sites of a genotype can be split into two types, according to the data in those sites. Type I includes the pairs of sites that have only one possible resolution. Those pair of sites are (00), (01), (10), (11), (20), (21), (02) and (12). The resolutions of those sites are described in the following list:

1. (00) $\rightarrow \left(\begin{smallmatrix} 0&0\\0&0 \end{smallmatrix}\right)$ 2. (01) $\rightarrow \left(\begin{smallmatrix} 0&1\\0&1 \end{smallmatrix}\right)$ 3. (10) $\rightarrow \left(\begin{smallmatrix} 1&0\\1&0 \end{smallmatrix}\right)$
4. (11) $\rightarrow \left(\begin{smallmatrix} 1&1\\1&1 \end{smallmatrix}\right)$ 5. (20) $\rightarrow \left(\begin{smallmatrix} 0&0\\1&0 \end{smallmatrix}\right)$ 6. (21) $\rightarrow \left(\begin{smallmatrix} 0&1\\1&1 \end{smallmatrix}\right)$
7. (02) $\rightarrow \left(\begin{smallmatrix} 0&0\\0&1 \end{smallmatrix}\right)$ 8. (12) $\rightarrow \left(\begin{smallmatrix} 1&0\\1&1 \end{smallmatrix}\right)$

Type II includes pairs of sites with (22) (*22-columns*). A 22-columns have two potential resolutions: (22) $\rightarrow \left(\begin{smallmatrix} 0&0\\1&1 \end{smallmatrix}\right)$, which will be called *equal resolution*, or (22) $\rightarrow \left(\begin{smallmatrix} 0&1\\1&0 \end{smallmatrix}\right)$, which will be called *unequal resolution*.

Determining whether there is a perfect phylogenic resolution of G boils down to deciding the resolution type, equal or unequal, for any pair of 22-columns. For some 22-columns, the type of the resolution is determined by the given set of genotypes, for others it determined by the given set of haplotypes, and for the rest we need algorithms that will find the proper resolution.

2.1 Preprocessing

We next present a preprocessing stage which is performed before all algorithms regardless of the specific structure of the input sets of genotypes or haplotypes.

A 22-columns ij must be resolved equally if the given set of genotypes G includes at least one of the following submatrices in columns ij:

$$\left(\begin{smallmatrix} 0&0\\1&1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 2&0\\1&2 \end{smallmatrix}\right)\ \text{or}\ \left(\begin{smallmatrix} 2&1\\0&2 \end{smallmatrix}\right),$$

since any resolution of G must include the combinations "00" and "11" in this case (see Fig. 3). For the same reason columns ij must be resolved unequally when G includes at least one of the submatrices

$$\left(\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right), \left(\begin{smallmatrix}2&0\\0&2\end{smallmatrix}\right) \text{ or } \left(\begin{smallmatrix}2&1\\1&2\end{smallmatrix}\right).$$

We will call this type of constraints on the resolution type *genotypes constraints*. In addition, a 22-columns ij must be resolved equally (unequally) if the haplotypes set includes for some genotype only equal (unequal) resolutions. This type of constraints will be called *haplotypes constraints*.

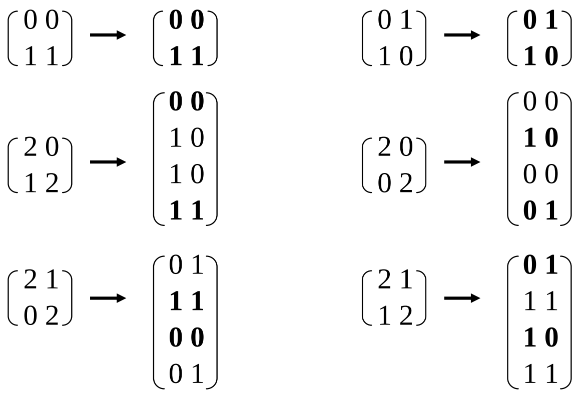


Fig. 3. Example of genotypes and their resolutions. Note that on the three cases on the left, the combinations "00" and "11" appear in the resolutions, and on the three cases on the right the combinations "01" and "10" are those who appear.

The preprocessing ensures that haplotype-pairs which violate the above constraints will not be chosen. For each genotype $g_i \in G$, we use $\widetilde{H}(g_i)$ to denote all possible resolutions of g_i in $\widetilde{H}$, *i.e.* $\widetilde{H}(g_i) = \{\{h, h'\} \mid h, h' \in \widetilde{H}$, h and h' resolve $g_i\}$. The preprocessing step includes the following four steps:

1. Check whether the genotype matrix G can be resolved in a perfect phylogenic way (use any algorithm from [2, 8, 10, 25]). If not, report there is no solution.
2. For each genotype $g_i \in G$, $1 \leq i \leq \ell$, go over all pairs of haplotypes from $\widetilde{H}$ and compute $\widetilde{H}(g_i)$.
3. For each genotype constraint, delete from the sets $\widetilde{H}(g_1), \ldots, \widetilde{H}(g_\ell)$ all haplotype pairs that violate the constraint, *i.e.* resolve the relevant sites in a different way than the constraint indicates.
4. For each haplotype constraint, delete from the sets $\widetilde{H}(g_1), \ldots, \widetilde{H}(g_\ell)$ all the haplotype pairs that violate it. Note that the deletion of haplotypes may create a new haplotype constraints. Repeat Step 4 until there is no change in the haplotype constraints.

After each step of steps 2 to 4 in the preprocessing stage, if one of the haplotypes sets $\widetilde{H}(g_1), \ldots, \widetilde{H}(g_\ell)$ remains empty, it means there is no solution and we done. Once the preprocessing is complete, our goal is to find a resolution $H \subseteq \widetilde{H}$ of G, by selecting one pair of haplotypes from each $\widetilde{H}(g)$, $g \in G$. From here on out, we will only be concerned with resolutions of this type. Note that even after the preprocessing stage, not all resolutions of this type will define a perfect phylogeny. This is because we are still left with 22-columns that have yet been resolved, as there might be two different genotypes g and g' which share a common pair of 22-columns ij, and $\widetilde{H}(g)$

and $\widetilde{H}(g')$ includes both resolutions for ij (*i.e.* equally and unequally). Such a pair of resolution is said to be *conflicting*, and more generally, any pair of resolutions $\{h_1, h'_1\}$ and $\{h_2, h'_2\}$ are *conflicting* if $\{h_1, h'_1, h_2, h'_2\}$ does not define a perfect phylogeny. We have the following important lemma:

Lemma 2. *After the preprocessing stage, $\{h_1, h'_1\}, \ldots, \{h_r, h'_r\}$ are pairwise non-conflicting resolutions of r genotypes in G iff $H = \bigcup_{1 \leq i \leq r} \{h_i, h'_i\}$ defines a perfect phylogeny.*

Proof. Let $G' \subseteq G$ denote the subset of r genotypes which H resolves. If H defines a perfect phylogeny, then clearly $\{h_1, h'_1\}, \ldots, \{h_r, h'_r\}$ are pairwise non-conflicting. To prove the other direction of the lemma, it suffices to show that any four haplotypes in H define a perfect phylogeny. Suppose in way of contraction that this is not the case. Then there are four haplotypes h_a,h_b,h_c, and h_d in H, such that $\{h_a, h_b, h_c, h_d\}$ do not define a perfect phylogeny. This means that there is a pair of sites $i, j \in \{1, \ldots, m\}$ such that $\{h_a, h_b, h_c, h_d\}$ will have the forbidden gamete matrix at ij. There are three possible cases:

– The four haplotypes belong to two different resolutions in $\{h_1, h'_1\}, \ldots, \{h_r, h'_r\}$. But this contradicts the assumption that $\{h_1, h'_1\}, \ldots, \{h_r, h'_r\}$ are pairwise non-conflicting.

– The four haplotypes belong to three different resolutions in $\{h_1, h'_1\}, \ldots, \{h_r, h'_r\}$. Then, w.l.o.g., $\{h_a, h_b\}$ is resolution of some genotype $g \in G'$, and ij is a pair of 22-columns in g, since $h_a(i) \neq h_b(i)$ and $h_a(j) \neq h_b(j)$. Suppose w.l.o.g. (the other case is symmetric) that h_a and h_b resolve ij equally, *i.e.* $h_a(i) = h_a(j)$ and $h_b(i) = h_b(j)$, and let $g', g'' \in G'$ be the two genotypes that h_c and h_d resolve. Then it is not hard to verify that G must include one of the following five submatrices at rows $g'g''$ and columns ij:

$$\left(\begin{smallmatrix} 0 & 0 \\ 1 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 2 & 0 \\ 1 & 2 \end{smallmatrix}\right), \left(\begin{smallmatrix} 2 & 1 \\ 0 & 2 \end{smallmatrix}\right), \left(\begin{smallmatrix} 2 & 0 \\ 2 & 1 \end{smallmatrix}\right), \text{ or } \left(\begin{smallmatrix} 0 & 2 \\ 1 & 2 \end{smallmatrix}\right).$$

If G includes the last two submatrices, then G does not have a perfect phylogenic resolution in the first place (Lemma 1), and so the preprocessing stage would have reported “no solution”. If G includes the first two submatrices, then there is a genotype constraint on ij stating that it must be resolved unequally, and so $\{h_a, h_b\}$ would have been removed from $\widetilde{H}$ at step 3 of the preprocessing stage. In both cases we reach a contradiction.

– The four haplotypes belong to four different resolutions in $\{h_1, h'_1\}, \ldots, \{h_r, h'_r\}$. Consider the four genotypes $g_a, g_b, g_c, g_d \in G'$ that h_a,h_b,h_c, and h_d resolve. If ij is a pair of 22-columns in one of these genotypes, then this case is similar to the previous case. Otherwise, it is not difficult to verify that G must include one of the forbidden submatrices of Lemma 1, and so the preprocessing stage would have halted at its first step – contradiction.

In all three cases we have reached a contradiction, and so the lemma is proven. □

Lemma 3. *The preprocessing stage takes $O(m^4n^2 + m^2n^4)$ time.*

Proof. The first step of the preprocessing takes $O(\ell m^2)$-time by using one of the polynomial-time algorithms for PPH. The second step takes $O(\ell m n^2)$-time, since checking whether a specific haplotype pair resolves a specific genotype takes $O(m)$-time, and there are ℓ genotypes, and $O(n^2)$ haplotype pairs. The third step takes $O(m^2n^2)$-time, since finding one genotype constraint and deleting all violating haplotype pairs takes $O(\ell + n^2) = O(n^2)$-time ($\ell \leq n^2$) and it done for all pairs of columns. The last step takes $O(m^2n^2 \cdot min(m^2, n^2))$, since finding all haplotype constraints and delete all violating haplotype pairs take $O(m^2n^2)$-time and we repeat it at most m^2 times since each pair of columns may create a new constraint only once, or at most n^2 times since we cannot delete more than n^2 haplotype pairs. In summary, the total running time of the preprocessing stage is $O(m^2n^2 \cdot min(m^2, n^2) + \ell mn^2) = O(m^4n^2 + m^2n^4)$. $\square$

2.2 The dependency graph

This brings us to the notion independency and dependency between genotypes. Loosely speaking, a dependency between two genotypes $g, g' \in G$ arises when the decision on how to resolve g affects the decision on how to resolve g'. This obviously happens when there is a resolution $\{h_1, h'_1\} \in \widetilde{H}(g)$ conflicting with a resolution $\{h_2, h'_2\} \in \widetilde{H}(g')$. In this case we say that g and g' are *directly dependent*. If there is no resolution in $\widetilde{H}(g)$ conflicting with a solution in $\widetilde{H}(g')$, we say that g and g' are *independent*.

We next introduce the *dependency graph* $DG(G)$ of our given set of genotypes G, after they have been preprocessed by the algorithm in the previous section. Later on, we will use the properties of the dependency graph in our polynomial algorithms.

Definition 5 (dependency graph). *The* dependency graph $DG(G)$ *of a set of genotypes G is a graph which has a vertex for each genotype $g \in G$, and edge between vertices representing directly dependent genotypes (see Fig. 4).*

g_1 = 1 2 2 0

g_2 = 2 2 2 0

g_3 = 2 1 2 0

g_4 = 1 2 2 2

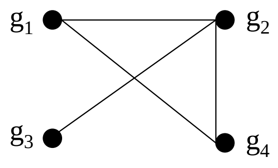


Fig. 4. Example of four genotypes and their corresponding dependency graph

Lemma 4. *After the preprocessing stage, two genotypes that do not have any pair of 22-columns in common are independent.*

Proof. Consider two genotypes $g, g' \in G$ that don't have any common pair of 22-columns. Suppose in way of contradiction that there is a resolution $\{h_x, h_y\} \in \widetilde{H}(g)$ conflicting with a $\{h_{x'}, h_{y'}\} \in \widetilde{H}(g')$. This means that there is a pair of columns ij of $\{h_x, h_y, h_{x'}, h_{y'}\}$ that has the forbidden gamete matrix. But this can only happen when G includes (in the rows gg', and in columns ij) one of the two 2×2 forbidden submatrices of Lemma 1. $\square$

Lemma 5. *Let G_1 and G_2 be two connected components in $D(G)$. If $H_1 \subseteq \widetilde{H}$ and $H_2 \subseteq \widetilde{H}$ are perfect phylogenic resolutions of G_1 and G_2 respectively, then $H_1 \cup H_2$ is a perfect phylogenic resolution of $G_1 \cup G_2$.*

Proof. Consider any pair of resolutions in $H_1 \cup H_2$ of two genotypes $g, g' \in G_1 \cup G_2$. If g and g' are not both in G_1, nor in G_2, then there is no edge between them in $DG(G)$, meaning that they are independent. Hence, the pair of resolutions is non-conflicting by definition. If $g, g' \in G_1$ or $g, g' \in G_2$, then by Lemma 2, the pair of resolutions is non-conflicting as both H_1 and H_2 define a perfect phylogeny. It follows that all resolutions in $H_1 \cup H_2$ are pairwise non-conflicting, and so again by Lemma 2, $H_1 \cup H_2$ defines a perfect phylogeny. □

In view of lemma 5, every connected component can be resolved individually, and the union of the chosen haplotypes will give the desirable resolution of G.

2.3 (*,1)- and (2,*)-bounded cases

We next turn to show how to use the properties of the dependency graph $DG(G)$ to solve various bounded-cases of CPPH. We begin with the simple cases of (1,*)-bounded and (*,2)-bounded genotype matrices.

Lemma 6. *Any pair of genotypes g_1 and g_2 in a (*,1)-bounded genotype matrix are not directly dependent.*

Proof. Since there is at most one 2 per column, g_1 and g_2 cannot have a 2 in the same column. Due to this, g_1 and g_2 can never share a pair of 22-columns, which according to Lemma 4, means they cannot be directly dependent. □

Lemma 7. *If g_1 and g_2 are two genotypes in a (2,*)-bounded genotype matrix, then w.l.o.g. they are not directly dependent.*

Proof. According to Lemma 4, two genotypes are directly dependent only if they have a pair of 22-columns in common. If g_1 and g_2 are different genotypes in a (2,*)-bounded genotype matrix, and they have a pair of 22-columns in common, it means that G contains at least one of the two 2×2 forbidden submatrices of Lemma 1, and so at the first step of the preprocessing stage we will determine that there is no solution. □

Theorem 1. CPPH *with (*,1)- or (2,*)-bounded genotype matrices is polynomial-time solvable.*

2.4 (5,2)-bounded case

It is convenient to mark every edge $\{g, g'\}$ in the dependency graph $DG(G)$ with the indices of the 2-columns g and g' share. Observe that in the (*,2)-bounded case, a specific index will appear only on one edge of the dependency graph, since there are no more than two genotypes that have 2 in the same column. Thus, for the (5,2)-bounded case, we have the following important property:

Lemma 8. *If G is a (5,2)-bounded genotype matrix then $DG(G)$ has maximum degree 2.*

Proof. Consider a genotype $g \in G$ with five 2s. According to Lemma 4, if g is directly dependent with any other genotype then they share a pair of 22-columns. There are two cases of direct dependency with g (see Figure 5):

1. g shares four columns with another genotype $g' \in G$. In this case, neither g nor g' can be directly dependent with other genotypes in G, as they both have at most one column left with a 2 that can be shared with other genotypes.
2. g shares three or two columns with another genotype $g' \in G$. In this case, g can be directly dependent with no more than one other genotype, since it does not have four columns that can be directly dependent with other genotypes in G.

$$\begin{array}{ll} g & 2\,2\,2\,2\,2\,0 \\ g' & 0\,2\,2\,2\,2\,2 \\ & \\ & \textbf{(1)} \end{array} \qquad \begin{array}{ll} g & 2\,2\,2\,2\,2 \\ g' & 2\,2\,2\,0\,0 \\ g'' & 0\,0\,0\,2\,2 \\ & \textbf{(2)} \end{array}$$

Fig. 5. Example of the two possible dependencies in a (5,2)-bounded genotype matrix.

In both cases g has degree of at most 2 in $DG(G)$, and so the lemma is proven. □

The lemma above implies, that if G is a (5,2)-bounded genotype matrix then every connected component in $DG(G)$ is either a path or a cycle. Furthermore, in the first case of the lemma, the connected component is of size 2, and all solutions for it can be determined trivially. We therefore focus on the second case of the lemma, where each edge in a given component is marked with at most 3 indices. We will initially assume that the component is a path with all edges labeled by two indices, and then show how to easily extend our ideas to the case of edges labeled by three indices, and to the case of a cycle.

Consider a connected component G' in $DG(G)$ which is a path comprised of r genotypes $g_1, \ldots, g_r$, where g_1 and g_r are vertices of degree 1, and g_i is connected to g_{i+1}, $1 \leq i < r$, by an edge which is marked with two indices. We partition the internal genotypes of the path G' into three types, depending on the number of possible resolutions available for them. Let $g_i \in G'$ be genotype in G' for some $1 < i < r$, where $\{g_{i-1}, g_i\}$ is labeled with two indices ab in $DG(G)$, $1 \leq a < b \leq m$, and $\{g_i, g_{i+1}\}$ is labeled with two indices cd, $1 \leq c < d \leq m$:

– Genotypes of type I have all four possible resolutions of ab and cd in $\widetilde{H}(g_i)$.
– Genotypes of type II have only three out of four possible resolutions of ab and cd in $\widetilde{H}(g_i)$.
– Genotypes of type III have only two out of four possible resolutions of ab and cd in $\widetilde{H}(g_i)$.

Note that these are the only possible cases, since we assume that any pair of columns label an edge can be resolved either equally or unequally, otherwise there is only one way to resolve it and we can remove this edge from the graph.

We next show how to find a perfect phylogenic resolution in $\widetilde{H}$ for the case of a path. Furthermore, we show that the path represents all legal perfect phylogenic resolutions.

Lemma 9. *In the case of a component which is a path with all edges labeled by two different indices there is always a perfect phylogenic resolution in $\widetilde{H}$.*

Proof. In order to prove the lemma we show how to obtain an actual solution: Consider a path $g_1, ..., g_r$ where the edge $\{g_1, g_2\}$ is labeled with the indices ab, and the edge $\{g_2, g_3\}$ is labeled with the indices cd. Start with g_1 and choose any resolution for columns ab, for example an equal resolution. Continue resolving the next pair of columns cd according to g_2's type:

– If g_2 is of type I then cd can be resolved either equally or unequally. Choose any resolution (equally, for example) and continue to the next edge.

– If g_2 is of type II then if ab was resolved in a way that leaves the two options to resolve cd, choose any one of them and continue. Otherwise, choose the only possible resolution and continue.
– If g_2 is of type III then there is only one way to resolve cd, choose this resolution and continue.

Note that in any of the three mentioned cases there is always a way of resolving the next pair of columns which proves that the above algorithm finds a perfect phylogenic resolution. □

We next show how to extend the above path algorithm to the case where some edges are labeled with three indices in the (5,2)-bounded case. Consider an edge $\{g, g'\}$ labeled with three indices abc, $1 \leq a < b < c \leq m$ in $DG(G)$. First we check that $\widetilde{H}(g)$ includes only pairs of haplotypes that have a non-conflicting resolution in $\widetilde{H}(g')$ and vice versa. We do so by checking all possible resolutions for columns abc in $\widetilde{H}(g)$ and $\widetilde{H}(g')$ and deleting the resolutions that appear only in one of those sets. This step ensures that if neither of the sets remains empty (which means there is no solution), then the algorithm will choose for abc a perfect phylogenic resolution. Note that the deletion of haplotype pairs may require deletion of other haplotype pairs in $\widetilde{H}(g_1), .., \widetilde{H}(g_\ell)$. Since that we will repeat step 4 of the preprocessing described in section 2.1. Observe that in the (5,2)-bounded case, the other edges connected to g and g' are labeled with at most two indices (since g and g' have at most five 2s each). According to that there are at most eight possible resolutions available for g and g', which means we can use the same path algorithm from lemma 9 while this time the algorithm may have more resolution options to choose from for the three indices edges.

We are left to show how to extend the path algorithm to the case of a cycle. Consider a cycle of r genotypes. The extension can be easily done by pulling out one edge from the cycle, for example, w.l.o.g., the edge $\{g_r, g_1\}$ what leaves us with a path comprises of r genotypes $g_1, g_2, ..., g_{r-1}, g_r$. We now check what are the possible resolutions for that path according to the way they resolve the columns label the edges $\{g_1, g_2\}$ and $\{g_{r-1}, g_r\}$. Note that there are exactly four types of possible resolutions according to this definition. The way of checking whether there exist any resolution of a specific type, for example the type that resolves the columns label $\{g_1, g_2\}$ and $\{g_{r-1}, g_r\}$ equally, is to run the path algorithm twice, in the first time starting from g_1 by choosing an equal resolution to the columns label $\{g_1, g_2\}$ and continue until reaching an edge with two possible resolutions. The second run will start from g_r and do the same on the opposite direction. If any of those runs reach the end of the path with the wrong resolution that means there is no resolution of the specific type. After knowing what types of resolutions exists, it is only left to check whether the columns label the removed edge $\{g_r, g_1\}$ have a resolution that does not conflict with any of those types.

Theorem 2. CPPH *with (5,2)-bounded genotype matrices is polynomial-time solvable.*

2.5 (3,3)-bounded case

In a (3,3)-bounded genotype matrix there are three cases of direct dependency for every genotype g (see figure 6).

1. g is directly dependent with two other genotypes sharing the same 22-columns. In this case any genotype of the three cannot be dependent with any other genotype since it left with at most one 2 to share, and so the corresponding connected component in $DG(G)$ is of size 3.
2. g is directly dependent with three other genotypes. In this case all four genotypes cannot be dependent with any other genotypes since they have at most one 2 to share, and so the corresponding connected component is of size 4.

$$\begin{array}{ll} g & \mathbf{222}10 \\ & 0\mathbf{222}0 \\ & 0\mathbf{22}1\mathbf{2} \end{array} \qquad \begin{array}{ll} g & 0\mathbf{222}0 \\ & 00\mathbf{222} \\ & \mathbf{222}10 \\ & 0\mathbf{2}1\mathbf{22} \end{array} \qquad \begin{array}{ll} g & \mathbf{222}10 \\ g' & 0\mathbf{222}0 \\ g'' & 01\mathbf{222} \\ & \ldots\ldots \end{array}$$

(1) **(2)** **(3)**

Fig. 6. Example of the three possible dependencies in a (3,3)-bounded genotype matrix.

3. g is directly dependent with exactly one other genotype g', and they have exactly one pair of 22-columns in common. In this case, g' can be directly dependent with another genotype g'', and so forth. The corresponding connected component in this case is a path, where each edge is labeled with two indices.

In the first two cases, all perfect phylogeny solutions can be determine whether the connected component has a prefect phylogeny resolution in $\widetilde{H}$ by simple exhaustive search. In the last case, we know by lemma 9 that the connected component necessarily has a prefect phylogenic resolution in $\widetilde{H}$, and we can use the algorithm described in that lemma for finding an actual solution. Observe that the algorithm will work correctly since in the third case described above, any pair of columns labels at most one edge across the path.

Theorem 3. CPPH *with (3,3)-bounded genotype matrices is polynomial-time solvable.*

3 Constrained Parsimony Haplotyping

We next consider the CPH problem. In [26], Sharan *et al.* showed that PH is fixed parameter tractable (see [9] for a formal definition) when parameterized by the size k of the resolution of G. Here, we show an analogous result for CPH. Our algorithm will perform relatively efficiently (in comparison to brute-force type algorithms) in cases of $\ell << n$. Our approach involves solving a dynamic program to determine whether there is any $H' \subseteq H$ of size $\kappa \leq k$ which resolves G. Throughout the section we use G_i, $1 \leq i \leq \ell$, to denote the subset of genotypes $\{g_1, \ldots, g_i\} \subseteq G$.

Probably the first dynamic-programming solution to come to mind, is to compute all possible resolutions $H' \subseteq H$ of G_i from the resolutions of G_{i-1}. However, the number of κ-subsets resolving G_i might be $\Omega(n^k)$, which is too much. We therefore take an alternative route. Instead of computing the actual subsets which resolve G_i, we will compute abstract "blueprints" of these subsets, formally defined in the following definition:

Definition 6 (κ-plan). *Let κ be an integer in $\{1, \ldots, k\}$. A* κ-plan *is a string of length $i \leq \ell$ over the alphabet $\{\{x, y\} \,|\, 1 \leq i \leq j \leq \kappa\}$.*

Let $H' = \{h_1, \ldots, h_\kappa\}$ be a resolution of $G_i = \{g_1, \ldots, g_i\}$, with some of the haplotypes in H' possibly equal. A κ-plan p is *associated with* H' if when $\{x, y\}$ is the j'th letter in p, $1 \leq x \leq y \leq \kappa$ and $1 \leq j \leq i$, then h_x and h_y resolve g_j. We will say that p is *valid* for G_i if there is a resolution of G_i associated with p. In this way, a valid κ-plan does not describe the actual resolution of G_i, but it does provide all relevant information concerning which genotypes are resolved using the same haplotypes.

Definition 7 ($\boldsymbol{\mathcal{P}[\kappa, i], P[\kappa, i]}$). *Let κ be an integer in $\{1, \ldots, k\}$, and i be an integer in $\{1, \ldots, \ell\}$. We denote by $\mathcal{P}[\kappa, i]$ the set of all κ-plans of length i, and by $P[\kappa, i] \subseteq \mathcal{P}[\kappa, i]$ the set of all valid κ-plans for $G_i = \{g_1, \ldots, g_i\}$.*

Lemma 10. $|P[\kappa, i]| \leq |\mathcal{P}[\kappa, i]| \leq k^{O(k^2)}$ *for any $\kappa \leq k$ and $i \leq \ell$.*

Proof. To prove the lemma, recall that $\ell \leq k^2$. The number of distinct strings of length at most k^2 over an alphabet of size at most $\binom{\kappa}{2} < k^2$ is bounded by $k^{O(k^2)}$. □

Our algorithm proceeds by computing $P[\kappa, i]$ in increasing values of κ and i. The base-cases of this computation are

1. $P[\kappa, 1] = \mathcal{P}[\kappa, 1]$ for all $1 \leq \kappa \leq k$, and
2. $P[1, i] = P[2, i] = \emptyset$ for all $2 \leq i \leq \ell$.

Clearly, G can be resolved using $\kappa \leq k$ haplotypes if and only if $G_\ell = G$ has at least one valid κ-plan. Hence, assuming we can correctly compute $P[\kappa, i]$ for all $1 \leq \kappa \leq k$ and $1 \leq i \leq \ell$, the correctness of our algorithm is immediate. What remains to be described is the dynamic-programming step for computing $P[\kappa, i]$.

For this, we will first need to introduce some terminology. Let p be some κ-plan which is valid for G_i, and let $h \in H$ be some haplotype. For a given $x \in \{1, \ldots, \kappa\}$, we say that *the assignment of $h_x = h$ is compatible with p* if there is a resolution $H' = \{h_1, \ldots, h_\kappa\}$ of G_i associated with p such that $h_x = h$. We extend this terminology also for assignments of pairs of haplotypes $h_x = h$ and $h_y = h'$, $h, h' \in H$ and $x \neq y \in \{1, \ldots, \kappa\}$. The dynamic-programming step for computing $P[\kappa, i]$ is as follows:

1. $P[\kappa, i] \leftarrow P[\kappa - 1, i]$.
2. For each $p \in P[\kappa - 2, i - 1]$:
   - Concatenate $\{\kappa, \kappa - 1\}$ to the end of p, and add this new κ-plan to $P[\kappa, i]$.
3. For each $h, h' \in H$ resolving g_i, for each $p \in P[\kappa - 1, i - 1]$, and for each $x \in \{1, \ldots, \kappa - 1\}$:
   - Check whether the assignment of $h_x = h$ is compatible with p. If so, concatenate $\{x, \kappa\}$ to the end of p, and add this new κ-plan to $P[\kappa, i]$.
4. For each $h, h' \in H$ resolving g_i, for each $p \in P[\kappa, i - 1]$, and for each $x \neq y \in \{1, \ldots, \kappa\}$:
   - Check whether the assignment of $h_x = h$ and $h_y = h'$ is compatible with p. If so, concatenate $\{x, y\}$ to the end of p, and add this new κ-plan to $P[\kappa, i]$.

Correctness of the dynamic programming step is straightforward. Indeed, any valid κ-plan p_0 of G_i is either a κ'-plan of G_i for some $\kappa' \leq \kappa$ (Line 1), or it can be decomposed into either:

- A valid $(\kappa - 2)$-plan p of G_{i-1} concatenated to a new letter $\{\kappa, \kappa - 1\}$ (Line 2). In this case, in any resolution $H' = \{h_1, \ldots, h_\kappa\}$ associated with p_0, we know that h_κ and $h_{\kappa-1}$ resolve g_i, and that $H' \setminus \{h_\kappa, h_{\kappa-1}\}$ resolves G_{i-1}.
- A valid $(\kappa - 1)$-plan p of G_{i-1} concatenated to a new letter $\{\kappa, x\}$ for some $1 \leq x \leq \kappa - 1$ (Line 3). In this case, in any resolution $H' = \{h_1, \ldots, h_\kappa\}$ associated with p_0, h_κ and h_x resolve g_i and $H' \setminus \{h_\kappa\}$ resolves G_{i-1}.
- A valid κ-plan p of G_{i-1} concatenated to a letter $\{x, y\}$ for some $1 \leq x < y \leq \kappa$ (Line 4). In this case, in any resolution $H' = \{h_1, \ldots, h_\kappa\}$ associated with p_0, h_x and h_y resolve g_i and H' also resolves G_{i-1}.

Note that as we know that p is associated with some resolution of G_{i-1}, we can determine in polynomial-time whether assignments are compatible with p. This can be done as follows: Suppose we want to determine whether $h_x = h$ is compatible with p. We mark all positions j, $1 \leq j \leq i-1$, with a letter $\{x, y\}$ in p. For each such position j, we compute $h_y = h'$ from g_j and h. Here, there are three possible outcomes – (i) we have reached a contradiction with a previous assignment, or (ii) we have discovered a new haplotype, or (iii) none of the previous two happens. In the first case we determine incompatibility. In the second case we continue with the checking process. In the third case, since we know that p is a κ-plan for G_{i-1}, we can safely determine compatibility. Checking whether $h_x = h$ and $h_y = y$ is compatible with p is done similarly. The entire process is performed in $O(\kappa)$ rounds, with each round requiring $O(\ell m)$ time, and so its total time complexity is $O(\kappa \ell m) = O(k^3 m)$.

Time-bounding the algorithm described above can be done as follows. Checking whether an assignment is compatible with a κ-plan requires $O(k^3 m)$ time. The number of assignments checked for each κ-plan is $O(\binom{\kappa}{2} \cdot n^2) = O(k^2 n^2)$. Since the number of distinct κ-plans is bounded by $k^{O(k^2)}$ (Lemma 3), each dynamic-programming step requires at most $O(k^{O(k^2)} k^5 n^2 m) = O(k^{O(k^2)} n^2 m)$ time. As the number of dynamic programming steps is $O(k\ell) = O(k^3)$, the total time complexity of our algorithm is $O(k^{O(k^2)} n^2 m)$.

Theorem 4. CPH *parameterized by* $k = |H|$ *is fixed-parameter tractable.*

References

1. The international hapmap project. *Nature*, 426:789–796, 2003.
2. V. Bafna, D. Gusfield, G. Lancia, and S. Yooseph. Haplotyping as perfect phylogeny: A direct approach. *Journal of Computational Biolegy*, 10:323–340, 2003.
3. T. Barzuza, J.S. Beckmann, R. Shamir, and I. Peer. Computational problems in perfect phylogeny haplotyping: Xor-genotypes and tag snps. In *15th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 14–31, 2004.
4. D. Brown and I.M. Harrower. A new integer programming formulation for the pure parsimony problem in haplotype analysis. In *Proceedings of Wrokshop on Algorithms in Bioinformatics (WABI)*, pages 254–265, 2004.
5. D. Brown and I.M. Harrower. Toward an algebric understanding of haplotype inference by pure parsimony. In *Computational Systems Bioinformatics Conference (CSB)*, pages 211–222, 2006.
6. P. Damaschke. Fast perfect phylogeny haplotype inference. In *14th Symposium on Fundamentals of Computation Theory (FCT)*, pages 183–194, 2003.
7. P. Damaschke. Incremental haplotype inference, phylogeny and almost bipartite graphs. In *Proceedings of RECOMB Satellite Workshop on Computational Methods for SNPs and Haplotypes*, pages 1–11, 2004.
8. Z. Ding, V. Filkov, and D. Gusfield. A linear-time algorithm for the perfect phylogeny haplotyping (pph) problem. *Journal of Computational Biolegy*, 13:522–553, 2006.
9. R. Downey and M. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.
10. E. Eskin, E. Halperin, and R. Karp. Efficient reconstruction of haplotype structure via perfect phylogeny. *Journal of Bioinformatics and Computational Biology*, 1:1–20, 2003.
11. J. Gramm, T. Nierhoff, R. Sharan, and T. Tantau. On the complexity of haplotyping via perfect phylogeny. In *Proceedings of RECOMB Satellite Workshop on Computational Methods for SNPs and Haplotypes*, 2004.
12. J. Gramm, T. Nierhoff, R. Sharan, and T. Tantau. Haplotyping with missing data via perfect path phylogenies. *Discrete Applied Mathematics*, 155:788–805, 2007.
13. D. Gusfield. Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions (extended abstract). In *Proceedings of RECOMB*, pages 166–175, 2002.
14. D. Gusfield. Haplotype inference by pure parsimony. In *14th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 144–155, 2003.
15. D. Gusfield and S.H. Orzack. Haplotype inference. *In CRC Handbook on Bioinformatics (Editor S. Aluru)*, 2005.
16. M.T. Hajiaghayi, K. Jain, K. Konwar, L.C. Lau, I.I. Mandoiu, A. Russell, A. Shvartsman, and V.V. Vazirani. The minimum k-colored subgraph problem in haplotyping and dna primer slection. In *Proceedings of International Workshop of Bioinformatics Research and Applications (IWBRA)*, pages 758–766, 2006.

17. B. Halldorsson, V. Bafna, N. Edwards, R. Lipert, S. Yooseph, and S. Istrail. A survey of computational methods for determining haplotypes. In *Proceedings of RECOMB Satellite on Computational Methods for SNPs and Haplotype Inference*, pages 26–47, 2003.
18. E. Halperin and E. Eskin. Haplotype reconstruction from genotype data using imperfect phylogeny. *Bioinformatics*, 20:1842–1849, 2004.
19. E. Halperin and R.M. Karp. Perfect phylogeny and haplotype assignment. In *Proceedings of RECOMB*, pages 10–19, 2004.
20. R. Hudson. Gene genealogies and the coalescent process. *Oxsford Survey of Evolutionary Biology*, 7:1–44, 1990.
21. L. Van Iersel, J. Keijsper, S. Kelk, and L.Stougie. Beaches os islands of tractability: Algorithms for parsimony and minimum perfect phylogeny haplotyping problems. In *Proceedings of Workshop on Algorithms in Bioinformatics (WABI)*, pages 80–91, 2006.
22. G. Kimmel and R. Shamir. The incomplete perfect phylogeny haplotype problem. *Journal of Bioinformatics and Comutational Biology*, 3:359–384, 2005.
23. G. Lancia, C. Pinotti, and R. Rizzi. Haplotyping population by pure parsimony: Complexity, exact and approximation algorithms. *INFORMS Journal on Computing, special issue on Comutational Biology*, 16:348–359, 2004.
24. G. Lancia and R. Rizzi. A polynomial case of the parsimony haplotyping problem. *Operations Research Letters*, 34:289–295, 2006.
25. R.V. Satya and A. Mukherjee. An optimal algorithm for perfect phylogeny haplotyping. *Journal of Computational Biolegy*, 13(4):897–928, 2006.
26. R. Sharan, B. Halldorsson, and S. Istrail. Islands of tractability for parsimony haplotyping. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3:303–311, 2006.
27. S. Tavare. Calibrating the clock: Using stochastic process to measure the rate of evolution. *In E. Lander and M. Waterman, editors. Calculating the Secrets of Life*, 1995.
28. L. Wang and L. Xu. Haplotype inference by maximum parsimony. *Bioinformatics*, 19:1773–1780, 2003.