Phylogeny- and Parsimony-Based Haplotype Inference with Constraints[1]

Michael Elberfeld*, Till Tantau

Institut für Theoretische Informatik, Universität zu Lübeck, 23538 Lübeck, Germany

Abstract

Haplotyping, also known as haplotype phase prediction, is the problem of predicting likely haplotypes based on genotype data. One fast computational haplotyping method is based on an evolutionary model where a perfect phylogenetic tree is sought that explains the observed data. An extension of this approach tries to incorporate prior knowledge in the form of a set of candidate haplotypes from which the right haplotypes must be chosen. The objective is to increase the accuracy of haplotyping methods, but it was conjectured that the resulting formal problem *constrained perfect phylogeny haplotyping* might be NP-complete. In the paper at hand we present a polynomial-time algorithm for it. Our algorithmic ideas also yield new fixed-parameter algorithms for related haplotyping problems based on the maximum parsimony assumption.

Key words: haplotype inference, perfect phylogeny, maximum parsimony, polynomial-time algorithms, fixed-parameter algorithms

1. Introduction

In large-scale studies of the relation between genomic variation and phenotypic traits, low-cost sequencing methods are used to read out the DNA sequences of many individuals. For each individual the bases present on the two chromosomes at a large number of SNP (single nucleotide polymorphism) sites are determined, yielding the individual's *genotype* for the different sites. In order to study phenotypic traits that are related to the bases present on multiple loci on a single DNA strand, it is important to determine *haplotypes* rather than genotypes. They describe how bases are assigned to chromosomes (this assignment of bases to haplotypes is also known as *phasing*), but are expensive to determine directly. *Haplotype inference* or just *haplotyping* methods aim at computationally predicting haplotypes from genotypes by using biological insights into the haplotype distribution in a population. They either use statistics, pioneered in [2], or combinatorics, the two most common approaches being the *perfect phylogeny method* (haplotype evolution is assumed to take place with unique point mutation and without recombination) and the *maximum parsimony method* (haplotype evolution is assumed to produce only few haplotypes).

Most combinatorial algorithms ignore prior knowledge that we might have on which haplotypes may be permissible to explain a given genotype. In some situations a pool of haplotypes from prior studies is already known and we should only pick haplotypes out of this pool. We may even have more specific information about the permissible haplotypes for the genotypes of the individuals, such as their ethnicity, allowing us to further narrow the pool of permissible haplotypes for groups of individuals. On the other hand, for some individuals no prior knowledge may be available.

In the present paper we study combinatorial haplotyping methods that take such *pool constraints* into account. For some or all genotypes we are given a pool of haplotypes that are allowed for this particular genotype. The task is to

[1]This is the accepted author manuscript of the article `http://dx.doi.org/10.1016/j.ic.2011.03.009`. A preliminary version of this paper was presented at the CPM 2010 conference [1].

*Corresponding author telephone number: +494515005312, *fax number:* +494515005301

Email addresses: `elberfeld@tcs.uni-luebeck.de` (Michael Elberfeld), `tantau@tcs.uni-luebeck.de` (Till Tantau)

URL: `http://www.tcs.uni-luebeck.de/mitarbeiter/elberfeld/` (Michael Elberfeld), `http://www.tcs.uni-luebeck.de/mitarbeiter/tantau/` (Till Tantau)

February 8, 2012

predict haplotypes for the genotypes such that all constraints are satisfied and the haplotypes form a perfect phylogeny or their number is minimal or both.

The above ideas lead to three mathematical problems, whose complexity we study in the present paper: c_{pools}PPH is the *constrained perfect phylogeny haplotyping problem*, c_{pools}MH is the *constrained maximum parsimony haplotyping problem*, and c_{pools}MPPH is the combined problem (see Section 2 for formal definitions). The two problems c_{pools}PPH and c_{pools}MH are generalizations of the two problems $c_{\text{one pool for all}}$PPH and $c_{\text{one pool for all}}$MH recently studied by Fellows et al. [3]; the difference is that Fellows et al. require a single pool of haplotypes to be used for all genotypes while we allow pools to be specified individually for each genotype. We remark that, since we also allow that no constraints are imposed at all, the standard problems PPH, MH, and MPPH (without any constraints) are special cases of their constrained counterparts and the algorithms we present also work for them.

Our Results. Our first main result is a polynomial-time algorithm for c_{pools}PPH. It is based on an initial partition of the genotypes into independent subinstances and a subsequent recursive decomposition of the pool constraints. Since this algorithm also solves the simpler problem $c_{\text{one pool for all}}$PPH, we settle the main open problem of Fellows et al. [3]: $c_{\text{one pool for all}}$PPH is polynomial-time solvable.

Our second set of results concerns maximum parsimony haplotyping. Both MH and $c_{\text{one pool for all}}$MH are known to be NP-complete, but fixed-parameter tractable with respect to the number of distinct haplotypes in the solution [3, 4]. We show that, in contrast, c_{pools}MH is hard for the class W[2] for the same parameter and, therefore, unlikely to have a fixed-parameter algorithm. We prove this by showing that $c_{\text{pools for all}}$MH, where some pool *must* be specified for each genotype, is W[2]-complete. On the positive side we present a fixed-parameter algorithm for c_{pools}MH where the parameter is the number of distinct haplotypes in the solution plus the number of times duplicated genotypes have incomparable pool constraints.

Our third main result is that the NP-complete problem c_{pools}MPPH *is* fixed-parameter tractable with respect to the number of distinct haplotypes in the solution. So, c_{pools}MPPH has the same complexity as $c_{\text{one pool for all}}$MH. As corollaries we obtain that MPPH and $c_{\text{one pool for all}}$MPPH are both fixed-parameter tractable, which was not known before. Our algorithm is a combination of the algorithmic ideas for c_{pools}PPH and c_{pools}MH.

Related Work. The study of the perfect-phylogeny haplotyping problem was initiated by the seminal paper of Gusfield [5], who showed that it is solvable in polynomial time. Subsequent papers presented conceptually simpler polynomial-time algorithms [6, 7], linear-time algorithms [8, 9, 10, 11], and fine-grained complexity-theoretic results [12, 13] for it.

The problem MH is NP-complete, as remarked in [14], and a later publication sharpens this lower bound by showing that MH remains NP-complete if every given genotype has at most three heterozygous sites [15]. On the positive site Sharan, Halldórsson, and Istrail [4] devised a fixed-parameter algorithm for MH, where the parameter is the number of distinct haplotypes in the solution. Fleischer et al. [16] presented general techniques to improve the parameter-dependent part of the runtime of fixed-parameter algorithms for parsimony-based haplotype inference. Moreover, algorithms based on linear programming [17], branch-and-bound algorithms [18], and a combination of both methods [19] are known for MH.

To increase the accuracy of the predicted haplotypes, the perfect phylogeny and the maximum parsimony assumptions have been combined, leading to the problem MPPH. It was shown to be NP-complete for instances with at most three heterozygous entries per genotype by Bafna et al. [20] and later studied by Iersel et al. [21].

Another direction to increase prediction accuracy is to constrain the set of solution haplotypes: Fellows et al. [3] proposed the $c_{\text{one pool for all}}$PPH problem and presented polynomial-time algorithms for some special cases like the number of heterozygous entries in the genotypes and in the sites being bounded by small constants. They left open the complexity of $c_{\text{one pool for all}}$PPH and leaned towards the conjecture that it is NP-complete. The problem $c_{\text{one pool for all}}$MH is NP-complete by a reduction from MH with at most three heterozygous entries per genotype (for each genotype put all its explaining haplotype pairs, of which there can be at most four, into the pool). Huang et al. [22] studied approximation algorithms for this problem, Fellows et al. [3] showed that it is fixed-parameter tractable with respect to the number of distinct haplotypes in the solution.

Organization of This Paper. We first give formal definitions of genotypes, haplotypes, and the computational problems we study in Section 2. Sections 3, 4, and 5 are devoted to the algorithmic and complexity-theoretic studies of c_{pools}PPH, c_{pools}MH, and c_{pools}MPPH, respectively.

2. Preliminaries

A *haplotype* describes the genetic information from a single chromosome at SNP sites. Since most SNP sites are biallelic, it is customary to encode a haplotype as a binary string $h \in \{0,1\}^n$, where 0 and 1 represent the two possible alleles. A genotype combines the genetic information of two haplotypes by joining their entries to a sequence of sets. Following common conventions, instead of sets we write a 0 or a 1 when both underlying haplotypes have this value (these entries are called *homozygous*) and use the value 2 when the underlying haplotypes have different entries (these entries are called *heterozygous*). A pair of haplotypes $\{h, h'\} \subseteq \{0,1\}^n$ *explains* a genotype $g \in \{0,1,2\}^n$ if for every site $s \in \{1,\ldots,n\}$ we have $g[s] = h[s] = h'[s]$ whenever $g[s] \in \{0,1\}$ and $h[s] \neq h'[s]$ whenever $g[s] = 2$. In a *genotype matrix* A each row is a genotype. If the matrix is clear from the context, we refer to the genotype in row i by g_i. Similarly, we arrange haplotypes in a *haplotype matrix* B and refer to the haplotype in row i by h_i. A $2n \times m$ haplotype matrix B *explains* an $n \times m$ genotype matrix A if every genotype g_i is explained by the haplotype pair $\{h_{2i-1}, h_{2i}\}$. We use the term *site* to refer to positions in genotypes and haplotypes and also to columns of genotype and haplotype matrices.

For a pair s and t of sites the *induced set* $\mathrm{ind}(B, s, t)$ contains all strings from $\{00, 01, 10, 11\}$ that appear in the sites s and t in the haplotype matrix B in some row. We say that these strings are *induced* by s and t. The notion of induces can be extended to genotype matrices A: for two sites s and t the set $\mathrm{ind}(A, s, t)$ contains a string $xy \in \{00, 01, 10, 11\}$ if A has a genotype g with either $g[s] = x \wedge g[t] = y$ or $g[s] = x \wedge g[t] = 2$ or $g[s] = 2 \wedge g[t] = y$. This implies $\mathrm{ind}(A, s, t) \subseteq \mathrm{ind}(B, s, t)$ for every haplotype matrix B explaining A.

A haplotype matrix B *admits a perfect phylogeny* if there exists a tree T (an undirected acyclic graph), such that: (a) Each haplotype from B labels exactly one vertex of T; (b) each site $s \in \{1,\ldots,m\}$ labels exactly one edge of T and each edge is labeled by at least one site; and (c) for every two haplotypes h_i and h_j from B and every site $s \in \{1,\ldots,m\}$, we have $h_i[s] \neq h_j[s]$ if, and only if, s lies on the path from h_i to h_j in T. It is well-known that B admits a perfect phylogeny if, and only if, it satisfies the following *four gamete property: For every pair of sites s and t we have* $\{00, 01, 10, 11\} \neq \mathrm{ind}(B, s, t)$.

For the three problems PPH, MH, and MPPH the input is always a genotype matrix plus, for the last two problems, a number k. The questions are whether there exists a haplotype matrix B that explains A and admits a perfect phylogeny (PPH), has at most k different haplotypes (MH), or admits a perfect phylogeny and has at most k different haplotypes (MPPH).

Constrained Haplotyping Problems. For *constrained* haplotyping problems different kinds of *constraints* are specified along with the input genotype matrix. The first kind of constraints that we study are what we call *pool constraints*. Let A be an $n \times m$ genotype matrix. A pool constraint specifies that, in the output haplotype matrix, the two explaining haplotypes for some particular genotype g_i should both be drawn from a *pool* $H_i \subseteq \{0,1\}^n$ of allowed haplotypes. This constraint is written as $\mathrm{pool}(i, H_i)$. Clearly, it suffices to allow only one such constraint per genotype. Two pool constraints are *incomparable* if none of their pools is a subset of the other.

The second kind of constraints are restrictions on the phase of sites. For a genotype g with 2-entries at two sites s and t, the explaining haplotypes add either $\{00, 11\}$ or $\{01, 10\}$ to the induced set. If there is another genotype g' with 2-entries in the sites s and t, then, in order to satisfy the four gamete property, it must choose the same pair for its explaining haplotype. In the first case we say that s and t are *phased equally*, otherwise *phased unequally*. The *phase constraints* equal-phase(s, t) and unequal-phase(s, t) specify that a particular phasing must be chosen for the two sites s and t in a solution matrix. Formally, a haplotype matrix B satisfies equal-phase(s, t) if $\{01, 10\} \nsubseteq \mathrm{ind}(B, s, t)$; and unequal-phase$(s, t)$ if $\{00, 11\} \nsubseteq \mathrm{ind}(B, s, t)$.

We indicate constrained haplotyping problems by prefixing the haplotyping problems MH, PPH, and MPPH with a c whose index indicates which constraints are allowed to be specified as part of the input. The index "pools" means that arbitrary pool constraints are allowed; "pools for all" indicates that (possibly different) pools must be specified for all genotypes (and not only for some); and "one pool for all" indicates that, additionally, the same pool must be specified for all genotypes. The index "phase" indicates that phase constraints are permissible. For example, $\mathrm{c}_{\mathrm{pools,phase}}$MPPH is the MPPH problem where both haplotype and phase constraints are allowed as part of the input.

Haplotyping with phase constraints has not been defined formally in the literature, but many known algorithms implicitly handle phase constraints:

Fact 2.1 ([6, 7, 12])**.** *There exists an algorithm that, given an $n \times m$ genotype matrix with phase constraints, solves the problem* c_{phase}PPH *in time $O(nm^2)$.*

3. Constrained Perfect Phylogeny Haplotyping

In this section we prove the following theorem, which answers the main question of Fellows et al. [3] affirmatively: There is a polynomial-time algorithm for $c_{\text{one pool for all}}$PPH.

Theorem 3.1. *There exists an algorithm that, given an $n \times m$ genotype matrix with phase constraints and pool constraints, solves* $c_{\text{pools,phase}}$PPH *in time $O((p+1)(n+p)m^2)$, where p equals the sum of the sizes of all pools.*

The outline of the algorithm for $c_{\text{pools,phase}}$PPH, which we detail in the rest of this section, is as follows: Given an $n \times m$ genotype matrix A and a set K of pool and phase constraints, our algorithm uses procedure SOLVE-CPPH from Figure 1 to preprocess the input and to partition the genotypes into at most m matrices A_s that can be solved independently. Each matrix A_s has the property that there is a site s, called the *2-site of A_s*, that has only 2-entries in all genotypes from A_s. Each A_s along with its corresponding constraints is then solved by the procedure SOLVE-CPPH-2-SITE from Figure 1 via a recursive branch-and-reduce approach: For each of the two possible phasings between the 2-site and another site, it branches recursively, derives new phase constraints, and splits the pool constraints.

In the following we describe the four procedures that make up our algorithm: the two main procedures SOLVE-CPPH and SOLVE-CPPH-2-SITE, whose pseudo-codes are depicted in Figure 1, and the simpler procedures SANITIZE-POOL-CONSTRAINTS and DEDUCE-PHASE-CONSTRAINTS for which no pseudo-code is given. Since many of these procedures only simplify the input without directly deciding the problem, it is useful to introduce the following notion:

Definition 3.2 (Correctness Property)**.** A computational step *has the correctness property* if the following holds: There exists a haplotype matrix that explains the genotype matrix and satisfies the four gamete property and the constraints before the step if, and only, if this holds after the step for the modified instance. Furthermore, whenever the step outputs "no", no solution exists for the current instance.

Procedure SANITIZE-POOL-CONSTRAINTS. This procedure removes superfluous haplotypes from pool constraints. Let K be a set of constraints. First, for a constraint pool$(i, H_i) \in K$ and a genotype g_i, it removes all h from H_i for which there exists a site s such that $h[s] \neq g_i[s] \in \{0, 1\}$. Second, it deletes every haplotype h from H_i for which there exists no other haplotype $h' \in H_i$ such that $\{h, h'\}$ explains g_i. Third, it deletes haplotypes contradicting phase constraints: For two sites s and t with $g_i[s] = g_i[t] = 2$, it deletes h from H_i whenever $h[s] = h[t] \wedge$ unequal-phase$(s, t) \in K$ or $h[s] \neq h[t] \wedge$ equal-phase$(s, t) \in K$. Finally, if a pool constraint becomes empty, it outputs "no." Clearly, this step has the correctness property.

Procedure DEDUCE-PHASE-CONSTRAINTS. Let A be a genotype matrix and K a set of constraints. The procedure repeats the following rule as long as possible: Let s, t and u be three sites such that there is a genotype g_i with $g_i[s] = g_i[t] = g_i[u] = 2$ and there is no phase constraint for the pair t and u, but phase constraints for both pairs s and t, and s and u. If these phase constraints have the same type, we insert equal-phase(t, u) into K and, if their type is different, we insert unequal-phase(t, u) into K. Using graph representations for phase constraints and their dependencies, the result of this procedure can be computed in time $O(nm^2)$ [6, 7, 12].

Lemma 3.3. DEDUCE-PHASE-CONSTRAINTS *has the correctness property.*

Proof. Let s, t, and u be three sites such that there is a genotype g_i with $g_i[s] = g_i[t] = g_i[u] = 2$, no phase constraint for t and u, but equal-phase$(s, t) \in K$ and equal-phase$(s, u) \in K$ (for other phase constraints the following arguments are similar). Let $\{h, h'\}$ be the explaining haplotype pair from a solution haplotype matrix for g_i. Without loss of generality assume that $h[s] = 0$ and $h'[s] = 1$ which implies $h[t] = 0$, $h'[t] = 1$, $h[u] = 0$ and $h'[u] = 1$ by the phase constraints. Thus 00 and 11 are induced by the haplotype matrix in t and u and, therefore, 01 or 10 is not induced. Thus, it is safe to insert equal-phase(s, t) into K. □

Procedure SOLVE-CPPH(A, K).
Input: An $n \times m$ genotype matrix A and a set of haplotype and phase constraints K
Output: An explaining haplotype matrix B for A that satisfies the four gamete property and the constraints K, if it exists; or "no", otherwise
Preprocessing:
1 ensure that column pairs with different entries induce 00
2 sort columns decreasingly by leaf count
3 update phase constraints with induces
4 **call** DEDUCE-PHASE-CONSTRAINTS
5 **call** SANITIZE-POOL-CONSTRAINTS
Solve independent subinstances:
6 **for each** site $s \in \{1, \ldots, m\}$ **do**
7 $B_s \leftarrow$ **call** SOLVE-CPPH-2-SITE(A_s, K_s, s)
8 **if** B_s is "no" **then return** "no"
9 **return** combination of matrices B_s and genotypes without 2-entries

Procedure SOLVE-CPPH-2-SITE(A, K, s_2).
Input: An $n \times m$ genotype matrix A with a 2-site s_2 and a set of haplotype and phase constraints K
Output: An explaining haplotype matrix B for A that satisfies the four gamete property and the constraints K, if it exists; or "no", otherwise
Recursion break:
1 **if** for every pool$(i, H_i) \in K$ we have $|H_i| = 2$ **then**
2 replace all pool constraints by corresponding phase constraints
3 **return** solution for the resulting c_{phase}PPH instance
Recursive branch-and-reduce:
4 **else for each** component G' from G_{cover} with corresponding instance A', K' **do**
5 $s \leftarrow$ some site from G'
6 $B'_e \leftarrow$ **call** TRY-PHASE-CPPH$(A', K' \cup \{\text{equal-phase}(s_2, s)\}, s_2)$
7 $B'_u \leftarrow$ **call** TRY-PHASE-CPPH$(A', K' \cup \{\text{unequal-phase}(s_2, s)\}, s_2)$
8 **if** B'_e = "no" and B'_u = "no" **then return** "no" **else** add non-"no" B'_e or B'_u to solution
9 **return** solution

Sub-Procedure TRY-PHASE-CPPH(A, K, s_2).
1 **call** DEDUCE-PHASE-CONSTRAINTS
2 **call** SANITIZE-POOL-CONSTRAINTS
3 **if** pool$(i, \emptyset) \notin K$ for all i **then return** SOLVE-CPPH-2-SITE(A, K, s_2) **else return** "no"

Figure 1: The polynomial-time algorithm for $c_{\text{pools,phase}}$PPH.

Procedure SOLVE-CPPH. The pseudo-code of this procedure is shown in Figure 1. We go over this method line by line.

The first five lines preprocess the input. Line 1 extends an idea from Eskin, Halperin and Karp [7] to constraints. For every site s we iterate downwards through the genotypes and if a 1-entry appears before a 0-entry, we substitute all 1-entries by 0-entries and vice versa and adjust the constraints accordingly. As shown in [7], this step ensures that any two sites with at least one different entry induce 00. In line 2 the procedure first calculates the *leaf count* [5] of each column, which is the number of 2-entries of a column plus twice the number of its 1-entries. Then it sorts the columns decreasingly from left to right by this value. After this sorting we have $10 \in \text{ind}(A, s, t)$ for every two sites s and t with different entries and $s < t$. This holds since otherwise there is no genotype with $g[s] = 1 \wedge g[t] \in \{0, 2\}$ or $g[s] = 2 \wedge g[t] = 0$, but at least one genotype with $g[s] \in \{0, 2\} \wedge g[t] = 1$ or $g[s] = 0 \wedge g[t] = 2$. This would imply that the leaf count of site t should be greater than the leaf count of site s, a contradiction. In line 3 the algorithm considers all pairs of sites s and t and updates their phase constraints as follows: If $\{00, 11\} \subseteq \text{ind}(A, s, t)$, it inserts equal-phase$(s, t)$ into K; and if $\{01, 01\} \subseteq \text{ind}(A, s, t)$, it inserts unequal-phase(s, t). This step has the correctness

property since the new phase constraints reflect only induces that are already in the matrix. Finally, lines 4 and 5 deduce phase constraints and sanitize the pool constraints. In the following, we call a matrix that has undergone the preprocessing from lines 1 to 5 a *preprocessed genotype matrix*.

In lines 6 to 8 the genotype matrix A is partitioned genotype-wise into m submatrices $A_1, \ldots, A_m$, one matrix for every site. A genotype g belongs to the matrix A_s if $g[s] = 2$ and for every site $t < s$ we have $g[t] \neq 2$. Each A_s is passed along with the corresponding pool constraints and all phase constraints, stored in the set K_s, to a call of the procedure SOLVE-CPPH-2-SITE. The construction of A_s ensures that site s has 2-entries in all genotypes from A_s. The effect of the partition is stated by Lemma 3.4. An example of a preprocessed genotype matrix together with a description of its partition is given in Figure 2.

Genotypes A:	Pool and Phase Constraints K:
123456	
1 220002	pool(1, {000000, 110001})
2 122000	
3 122200	pool(3, {100100, 111000, 101000, 110100})
4 120200	pool(4, {110100, 100000, 100100, 110000})
5 120020	
	equal-phase(1, 3), equal-phase(1, 4), equal-phase(1, 5), unequal-phase(3, 4), unequal-phase(3, 5), unequal-phase(3, 6), unequal-phase(4, 5), unequal-phase(4, 6), unequal-phase(5, 6)

Figure 2: A preprocessed genotype matrix A with pool and phase constraints K is shown. In lines 6 to 8 the algorithm partitions this instance into six subinstances: Instance 1 is made up by the first genotype 220002 along with its pool constraint pool(1, {000000, 110001}) and all phase constraints. Instance 2 contains all other genotypes along with the corresponding pool constraints and all phase constraints. Instances 3 to 6 are empty.

Lemma 3.4. *Let A be a preprocessed $n \times m$ genotype matrix with constraints K. Then there exists an explaining haplotype matrix B for A that satisfies the four gamete property and the constraints K if, and only if, for every site $s \in \{1, \ldots, m\}$ there exists an explaining haplotype matrix B_s for A_s that satisfies the four gamete property and the constraints K_s.*

Proof. A solution for A and K can clearly be transformed into solutions B_s for every matrix A_s with constraints K_s. For the other direction, let B be the combination of haplotypes from B_s and haplotypes for genotypes that are not in any matrix A_s. Clearly, B explains A and satisfies the pool constraints. We show that it also satisfies the four gamete property and the phase constraints, which proves the lemma: Let t and u be two sites. Since the instance is preprocessed, we know $\{00, 01, 10, 11\} \neq \text{ind}(A, t, u)$. If there are no genotypes with 2-entries in both t and u, we are done, since the explaining haplotypes do not add new elements to the induced set. If there are genotypes with 2-entries in both t and u, we show that their explaining haplotype pairs do not add different elements to the induced set: If all these genotypes lie in the same matrix A_s, their explaining haplotype pairs add either $\{00, 11\}$ or $\{01, 10\}$ to the induced set and satisfy the phase constraints by assumption. Otherwise, let g be a genotype from A_s and g' a genotype from matrix $A_{s'}$ with $s < s'$ and 2-entries at sites t and u. Note that $t \neq s$ and $u \neq s$ since, otherwise, g and g' would lie in the same matrix A_s. We show that there exists a phase constraint for the sites t and u in K and, therefore, the induces for the sites t and u in the explaining haplotype pairs for g and g' are determined by the phase constraints: From the construction of the subinstances, we know that the entries of g and g' at sites s, s', t and u are

	s	s'	t	u
g	2	$g[s']$	2	2
g'	$g'[s]$	2	2	2

with $g'[s] \in \{0, 1\}$. If $g'[s] = 1$, both pairs of sites s and t and s and u induce $\{00, 11\}$ and the update of phase constraints with induces adds equal-phase(s, t) and equal-phase(s, u) to K. If $g'[s] = 0$, both pairs induce $\{01, 10\}$ and the same step adds unequal-phase(s, t) and unequal-phase(s, u). In both cases DEDUCE-PHASE-CONSTRAINTS adds the constraint equal-phase(t, u) whenever there is no phase constraint for t and u. □

Putting it altogether, we see that SOLVE-CPPH correctly solves $c_{\text{pools,phase}}$PPH, provided that the procedure SOLVE-CPPH-2-SITE is correct, which we prove next.

Procedure SOLVE-CPPH-2-SITE. This procedure recursively solves the instances that are produced by SOLVE-CPPH, each consisting of a genotype matrix A with a 2-site s_2 and constraints K. The recursion stops when all pool constraints contain only two haplotypes (they must contain at least two haplotypes because a 2-entry is present in the genotype). In such a case the phasing of the genotype is completely known. We remove the pool constraints and, instead, add phase constraints that describe this particular phasing: For each constraint pool$(i, \{h, h'\})$ and sites s and t add the phase constraint equal-phase(s, t) if $h[s] = h[t] \neq h'[s] = h'[t]$ and unequal-phase(s, t) if $h[s] = h'[t] \neq h[t] = h'[s]$. For example, the pool constraint pool$(i, \{000001, 110000\})$ of a genotype $g_i = 220002$ is replaced by the phase constraints equal-phase$(1, 2)$, unequal-phase$(1, 6)$, and unequal-phase$(2, 6)$. The phase constraints ensure that every solution explains g_i with the haplotypes 000001 and 110000. The resulting instance of c_{phase}PPH can be solved in polynomial time by Fact 2.1.

In order to describe the recursive step, we need some terminology. Let $\text{geno}_2(s)$ be the set of genotypes that have a 2-entry at site s. Let S_{free} be the set of sites s of A where $s \neq s_2$, $\text{geno}_2(s) \neq \emptyset$, and there is no phase constraint for s and s_2 in K. Let S_{cover} be the set of sites $s \in S_{\text{free}}$ for which there is no site $s' \in S_{\text{free}}$ with $\text{geno}_2(s) \subseteq \text{geno}_2(s')$; in the case that sites from S_{free} have the same set of 2-entries, we choose exactly one of them to be contained in S_{cover}. Note that when a genotype from A has a 2-entry in a site $s \in S_{\text{free}}$, then it also has a 2-entry in one of the sites in S_{cover}. Let G_{cover} be the graph that has S_{cover} as its vertex set and an edge between sites s and s' if $\text{geno}_2(s) \cap \text{geno}_2(s') \neq \emptyset$. Whenever there is an edge between sites in G_{cover}, then there exists a phase constraint for them. Figure 3 shows an example of the construction of G_{cover}.

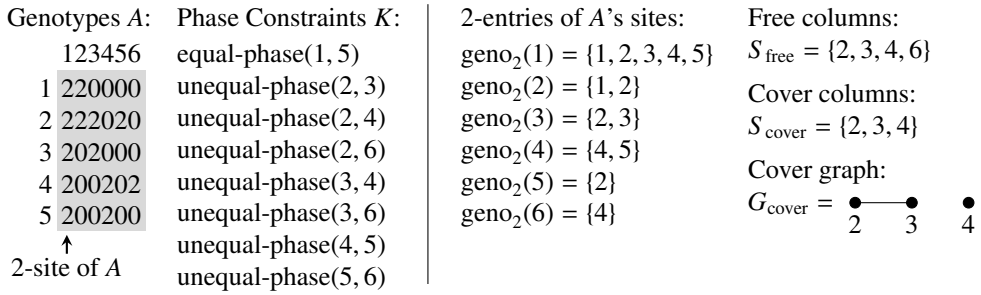

Figure 3: The construction of the graph G_{cover} is shown for a genotype matrix A with 2-site 1 and phase constraints K.

In the recursive step the algorithm iterates over the components G' of G_{cover}. For each G' it considers the sub-matrix A' of A made up by all genotypes with 2-entries in sites of G' along with a constraints set K', consisting of the pool constraints for the genotypes from A' and all phase constraints. It chooses a site s from G' and adds once the constraint equal-phase(s_2, s) and once unequal-phase(s_2, s) to the set of constraints. In each case, it checks which additional phase constraints are now triggered using the sub-procedure TRY-PHASE-CPPH. This sub-procedure calls DEDUCE-PHASE-CONSTRAINTS followed by SANITIZE-POOL-CONSTRAINTS and tries to solve the resulting instance recursively by calling SOLVE-CPPH-2-SITE. If for all components a recursive call returns a solution, the procedure combines them along with haplotypes for genotypes that are not in any matrix A' to a solution for the whole instance. The following lemma states the correctness of SOLVE-CPPH-2-SITE:

Lemma 3.5. *Let A be a preprocessed $n \times m$ genotype matrix with 2-site s_2 and constraints K. Then* SOLVE-CPPH-2-SITE *returns a haplotype matrix B that explains A and satisfies the four gamete property and the constraints K, if it exists, or "no", otherwise.*

Proof. We prove the lemma by induction over the size of S_{free}. If $|S_{\text{free}}| = 0$, then, since A is preprocessed, the phase constraints with s_2 completely determine the phase constraints for all sites with 2-entries. The sanitation of the pool constraints ensures that, then, every pool constraint contains exactly two haplotypes that explain the corresponding genotype. Thus, SOLVE-CPPH-2-SITE works correctly for $|S_{\text{free}}| = 0$.

Now we assume $|S_{\text{free}}| > 0$. If there is a solution haplotype matrix B and the procedure reaches lines 2 and 3, then B witnesses a positive answer. We are left to look at the case that the procedure iterates over matrices A' with constraints K' and branches recursively. The solution B for the whole matrix A with K gives a solution matrix B' for every submatrix A' with constraints K'. If B' does not induce both 01 and 10 in s_2 and s, then the call of TRY-PHASE-CPPH($A', K' \cup$ equal-phase(s_2, s), s_2) produces a positive answer by the induction hypothesis. If s_2 and s do not induce both 00 and 11, the call of TRY-PHASE-CPPH($A', K' \cup$ unequal-phase(s_2, s), s_2) succeeds.

For the completion of the correctness proof, we show that the algorithm only outputs correct solutions for $|S_{\text{free}}| > 0$. If the procedure stops with a solution in line 3, this is clear. Assume that the procedure outputs a solution B at the end, which is combined from the solutions of recursive calls (and the haplotypes for genotypes that are not in any submatrix). By induction, B explains A and satisfies the pool constraints from K. We are left to show that A satisfies the four gamete property and the phase constraints. Since SOLVE-CPPH-2-SITE is called by SOLVE-CPPH, we can assume that the genotype matrix itself does not induce four elements in any column pair. If there is a phase constraint for two sites s and t, then all non-"no" matrices B'_e and B'_u satisfy it and the combined haplotype matrix does not contain four elements in these sites. If there is no phase constraint for s and t, observe that there are not two genotypes from different submatrices with 2-entries in both s and t. Different submatrices only share 2-entries in s_2. If a matrix A' has a genotype g with 2-entries in s and t and there exists another genotype that is not part of any submatrix, also with 2-entries in s and t, then there exists a phase constraint for this pair of sites. Overall, this proves that the combined solution satisfies both the four gamete property and the phase constraints. □

Runtime. The input to the algorithm consists of a genotype matrix of dimension $n \times m$, phase constraints and pool constraints. Let p denote the sum of the sizes of all pool constraints. We show that the algorithms runs in time $O((p + 1)(n + p)m^2)$, as claimed in Theorem 3.1. All individual operations of the algorithm take time at most $O((n + p)m^2)$. Thus, it suffice to show that the tree of recursive calls of procedure SOLVE-CPPH-2-SITE has at most $p + 1$ leafs: The procedure partitions its input matrix into submatrices with constraints. For every submatrix A' with constraints K' it may branch into two possible phasings for the sites s_2 and s. The call of DEDUCE-PHASE-CONSTRAINTS ensures that there are phase constraints between s_2 and all sites from G': when two sites are connected via an edge in G_{cover}, we know that there is a phase constraint for them and a genotype that contains 2-entries in these sites and s_2. Note that the phases between s_2 and the sites from G' for the case equal-phase(s_2, s) are exactly opposite to the phases for the case unequal-phase(s_2, s). This implies that, since all genotypes in A' have a 2-entry in s_2 and a site from G', every haplotype from the pool constraints *is passed to at most one recursive call.* This yields a partition of sets of haplotypes from the pool constraints among all recursive calls. Since the procedure stops when the sizes of the pools drop to two (or zero), the number of leafs of the recursive tree of the procedure SOLVE-CPPH-2-SITE is bounded by $p + 1$.

Implementation and Experiments. We implemented the polynomial-time algorithm for constrained perfect phylogeny haplotyping and measured, for different kinds of constraints, its performance in terms of both runtime and accuracy. Our implementation is in Java and no runtime optimizations were performed.

For our experiments, we used a data set of real haplotypes from Andrés et al. [23] that contains haplotype pairs of 20 African American (AA) and 19 European American (EA) individuals from a 48kb genomic region of Chromosome 19 that spans the genes Kalekrein 13 and 14. From this data set, which contains 401 sites (biallelic SNPs and indels), we used all 210 sites without missing entries, yielding the data set HAPLO-DATA. To obtain genotypes as an input for the algorithm, we combined the haplotype pairs of the individuals to genotypes, stored in the data set GENO-DATA.

We applied the algorithm with different pool constraints to study how more and more specific pools affect its haplotyping accuracy: In the STANDARD approach we used perfect phylogeny haplotyping without any pools, working only on the genotype data. For the POOL approach we used a single pool for all genotypes that contained all haplotypes from HAPLO-DATA plus the same number of random haplotypes to model rare haplotypes. In the POPULATION approach we used different pools for the AA genotypes and the EA genotypes, which consist of the AA haplotypes and the EA haplotypes, respectively, again each time together with a corresponding number of random haplotypes. Since GENO-DATA is not explainable by a single perfect phylogenetic tree, we considered every maximal interval of sites that allows a solution for the most restrictive approach, POPULATION, and, therefore, also allows solutions for the less restrictive approaches POOL and STANDARD. For each interval we have calculated the number of solutions together with the average Haplotype error rate (HE), the single-site error rate (SSE) and the switch error (SWE) over all solutions for this interval (see Andrés

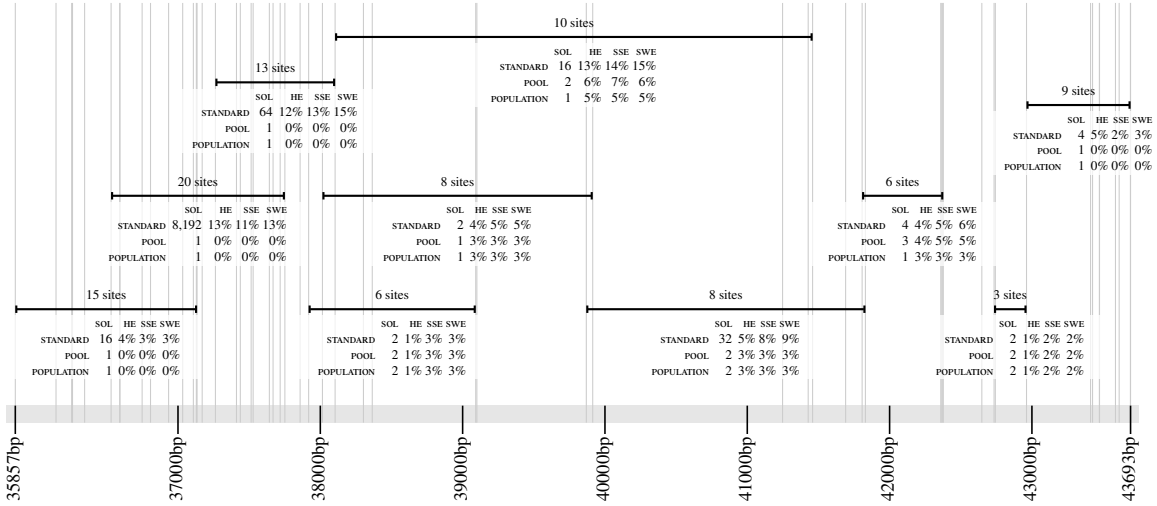

Figure 4: The region between 35857bp and 43693bp from the 48kb data set of Andrés et al. [23]. Vertical lines indicate sites without missing entries, horizontal bars are the maximal intervals of at least 3 sites that satisfy the POPULATION approach. Every interval is labeled by the number of spanned sites together with the number of solutions (SOL) and average error rates for all approaches.

et al. [23] for detailed definitions). Figure 4 shows the intervals and measures between the sites 35857 and 43693 of the 48kb region for one run of the program.

The first observation from our experiments is that the number of solutions that the algorithm produces reduces for more restrictive approaches. While the STANDARD approach may yield many different solutions (up to 8192 in an interval of the example), pools help to shrink the number of solutions and, therefore, uniquely predict haplotypes. There are also cases with a difference between the POOL and the POPULATION approaches, with POPULATION finding less solutions. The second observation is that the error rates are usually smaller for more restrictive approaches and, thus, for the observed data, pools help to find more accurate solutions. The prototype Java implementation handles the inputs from our experiments in a matter of seconds on a standard machine.

4. Constrained Maximum Parsimony Haplotyping

In this section, we present two results on the fixed-parameter tractability of the constrained maximum parsimony haplotyping problem. (We refer the reader to [24] for an introduction to parametrized complexity theory). First, we prove that $c_{\text{pools for all}}$MH is W[2]-complete when parametrized by the minimum number of distinct haplotypes in an explaining haplotype matrix. In sharp contrast, MH and $c_{\text{one pool for all}}$MH are fixed-parameter tractable for this parameter, as shown in [4] and [3], respectively. This means that the possibility to specify pool constraints on a per-genotype basis vastly increases the complexity of the problem. Second, we show that a fixed-parameter algorithm is possible even for c_{pools}MH when we extend the parameter to the number of distinct haplotypes plus the number of duplicated genotypes that have incomparable pools.

The algorithms for MH and $c_{\text{one pool for all}}$MH from the literature use data structures that describe how haplotypes are shared among genotypes. We formalize these data structures as follows:

Definition 4.1 (Haplotype Sharing Plan)**.** Given an $n \times m$ genotype matrix A, we define a *haplotype sharing plan P for A of size k* as a multigraph $G = (V, E)$ (a graph where multiple edges between two vertices are allowed) with $|V| = k$ and $|E| = n$ where

1. edges are labeled bijectively by genotypes from A,
2. some vertices are labeled by haplotypes, and
3. if an edge e labeled g connects two vertices v_1 and v_2 labeled h_1 and h_2, respectively, then g is explained by $\{h_1, h_2\}$.

We call a plan *complete* if all vertices are labeled and *empty* if no vertex is labeled. A plan P *extends* a plan P' if P arises from P' by labeling previously unlabeled vertices. A haplotype sharing plan P *satisfies a pool constraint* $\text{pool}(i, H_i)$ if the incident haplotypes of g_i lie in H_i. With this definition, constructing haplotype matrices with at most k distinct haplotypes is equivalent to constructing complete plans of size at most k. An example of a genotype matrix and a haplotype sharing plan is shown in Figure 5.

Genotypes A:

	12345
1	02022
2	20200
3	00222
4	22222
5	02211

Pool constraints K:

$\text{pool}(1, \{00000, 01011, 01000\})$

$\text{pool}(2, \{00000, 10100, 10000\})$

$\text{pool}(5, \{00111, 01011\})$

A partial haplotype sharing plan for A that satisfies K:

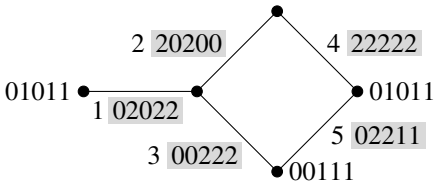

Figure 5: A genotype matrix A together with a haplotype sharing plan of size 5 for it. The haplotype sharing plan that is shown in the figure is neither empty nor complete. This means that some, but not all, vertices are labeled with haplotypes. The plan satisfies the pool constraints K: every genotype that has two incident haplotypes is explained by them.

Given a budget k for the number of distinct haplotypes in the solution, the known fixed-parameter algorithms for MH and $\text{c}_{\text{one pool for all}}\text{MH}$ consider all possible empty haplotype sharing plans of size k and check whether they can be extended to complete ones in polynomial time [4, 3, 16]. To bound the number of edges of the plan they use a preprocessing step that deletes duplicated genotypes and retains only one of them. Since k haplotypes can explain at most $k(k+1)/2$ different genotypes, these algorithms consider at most $O(k^{2n}) \leq O(k^{k^2+k})$ different empty plans.

These ideas cannot be extended to a fixed-parameter algorithm when genotype-specific pool constraints are given since we cannot delete duplicated genotypes in a preprocessing step. This is due to the fact that genotypes might have the same entries, but incomparable pools, which we cannot merge directly. Strong evidence that no slightly variation of the standard approaches will work is given by Theorem 4.2.

Theorem 4.2. $\text{c}_{\text{pools for all}}\text{MH}$*, parametrized by the number of distinct haplotypes in the solution, is* W[2]*-complete. Consequently,* $\text{c}_{\text{pools}}\text{MH}$ *is* W[2]*-hard for the same parametrization.*

Proof. We present a fixed-parameter reduction from the W[2]-hard [24] parametrized problem HITTING-SET to the problem $\text{c}_{\text{pools for all}}\text{MH}$. The HITTING-SET problem is defined as follows: Given a hypergraph G that consists of a vertex set V and a set of hyperedges $E = \{e_1, \ldots, e_n\}$ with $e_i \subseteq V$ and a parameter k, we want to find a *hitting set* $S \subseteq V$ for G, which means that $e \cap S \neq \emptyset$ for all $e \in E$, of size at most k.

Given a hypergraph G with vertices $V = \{v_1, \ldots, v_m\}$ and edges $E = \{e_1, \ldots, e_n\}$, our reduction constructs an instance for $\text{c}_{\text{pools for all}}\text{MH}$ as follows: The genotype matrix A contains n genotypes of length $m+1$ with only 2-entries. For $i \in \{1, \ldots, m\}$ let h_i be the haplotype of length $m+1$ with exactly one 1-entry, namely at site i, and let h'_i be its bitwise complement. For every genotype g_j there is a constraint $\text{pool}(j, \bigcup_{v_i \in e_j} \{h_i, h'_i\})$. The budget (the allowed number of distinct haplotypes to explain A) is set to $2k$. The construction is clearly computable in polynomial time.

For the correctness of the reduction, first let $S \subseteq V$ be a hitting set of size at most k for G. We choose an explaining haplotype pair for every genotype g_j as follows: Let $v_i \in S$ be a vertex with $v_i \in e_j$, then we use the haplotype pair $\{h_i, h'_i\}$ to explain g_j. Since S contains at most k elements, this gives an explaining haplotype matrix with at most $2k$ distinct haplotypes for A that satisfies the pool constraints by construction. For the opposite direction let B be a haplotype matrix with at most $2k$ distinct haplotypes explaining A. The matrix B can be seen as a sequence of n haplotype pairs, where the jth pair explains genotype g_j. Since there are at most $2k$ distinct haplotypes in the solution and the constructed haplotype pairs do not share any haplotypes, this sequence contains at most k different haplotype pairs. For every genotype g_j consider its explaining haplotype pair $\{h_i, h'_i\}$ from $\text{pool}(j, H_j)$ and insert the corresponding vertex v_i, for which $v_i \in e_j$ holds, into S. This gives a hitting set S of size at most k.

To prove $\text{c}_{\text{pools for all}}\text{MH} \in$ W[2], we present a fixed-parameter reduction to the W[2]-complete problem of finding a satisfying assignment for Boolean circuits of weft 2 and depth 3 with at most k inputs set to 1, denoted $\text{wcs}(2, 3)$ in [24]. The circuit's input layer has a different gate, called a *haplotype gate,* for every haplotype from the union of all

constraint pools. The next layer consists of $\wedge$-gates, called *genotype gates,* one for each possible pair of haplotypes. Each of these genotype gates is connected to a different pair of haplotype gates. Next, there is a layer of wide $\vee$-gates, called *row gates*. The ith row gate is connected to all genotype gates that are, in turn, connected to two haplotype gates h and h' such that (a) h and h' are both elements of the constraint pool for the ith row of the genotype matrix and (b) h and h' together explain this ith row. Finally, a wide $\wedge$-gate connected to all row gates assures that all genotypes are explained by haplotype pairs. Clearly, a satisfying assignment of this circuit with k inputs set to 1 corresponds to a set of k haplotypes that explain the genotype matrix and satisfy all pool constraints. □

The instances constructed in the W[2]-hardness proof of $c_{\text{pools for all}}$MH contain only identical genotypes, namely completely heterozygous genotypes, while pools might be highly incomparable. Since such a worst case instance is unlikely to be present in practice, we propose to additionally parametrize the problem by the maximum number l of duplicated genotypes with pairwise incomparable pool constraints. When parametrized by the number k of distinct haplotypes and at the same time by l, the problem c_{pools}MH becomes fixed-parameter tractable.

Theorem 4.3. c_{pools}MH *is fixed-parameter tractable with respect to the number of distinct haplotypes in the solution plus the maximum number of duplicated genotypes with pairwise incomparable pool constraints.*

Proof. Our fixed-parameter algorithm SOLVE-CMH is shown in Figure 6. After sanitizing the pool constraints, which does not change the size of a smallest solution, in lines 2 to 4 the algorithm conflates genotypes and their pools. This means that instead of simply deleting duplicated genotypes, which would not give a correct algorithm, it repeatedly finds genotypes g_i and g_j with $g_i = g_j$ and $\text{pool}(i, H_i), \text{pool}(j, H_j) \in K$ and $H_i \subseteq H_j$. Each time such genotypes are found, we can delete g_j. We also delete g_j if there is no pool constraint for it, but there is another genotype g_i with $g_i = g_j$. After this conflation, there can be at most l copies of each genotype by definition of l. In particular, if more than $lk(k+1)/2$ genotypes remain after line 4, they cannot be explained using only k haplotypes.

Next, the algorithm considers all empty haplotype sharing plans P of size at most k for the current matrix and tests, by using the below claim, whether there exists a complete extension for it (first without considering any pool constraints) and skips the current plan whenever the claim does not hold.

Let $G = (V, E)$ be the underlying graph of a haplotype plan P for A and let s be a site. Let $V_0^s \subseteq V$ be the set of all vertices from G that have a labeling haplotype with a 0-entry at site s or that have an incident genotype that has a 0-entry at site s. Define $V_1^s \subseteq V$ similarly, with "0" replaced by "1." Let G_2^s arise from G by deleting every edge whose genotype does not have a 2-entry at site s.

Claim. *Let A be an $n \times m$ genotype matrix and P be a haplotype sharing plan for A. There exists a complete haplotype sharing plan P' for A that extends P if, and only if, for every site s the following properties hold:*

1. *there is no odd-length cycle in G_2^s,*
2. *there is no even-length path between vertices $v \in V_0^s$ and $v' \in V_1^s$ in G_2^s, and*
3. *there is no odd-length path between vertices $v, v' \in V_0^s$ and also no odd-length path between vertices $v, v' \in V_1^s$ in G_2^s.*

Proof. First, we prove that the above properties are necessary. Consider a genotype matrix A and a complete haplotype sharing plan P for A with graph $G = (V, E)$. For the sake of contradiction, assume that there exists a site s such that G_2^s contains an odd-length cycle. Let $h_1, g_1, \ldots, h_r, g_r, h_1$ be the sequence of haplotypes and genotypes on this cycle. We know that $h_1[s] = a \in \{0, 1\}$ and, since $g_1[s] = 2$, we have $h_2[s] = 1 - a$. If we transfer this fact to all haplotypes in the cycle, we known that $h_i[s] = a$ whenever i is odd and $h_i[s] = 1 - a$ whenever i is even. Since r is odd, we have $h_r[s] = a$ and, therefore, $h_1[s] = 1 - a$ holds, a contradiction. The arguments for the second and third properties are similar, only the values of the first and last haplotypes on the path are known.

To show that the properties are sufficient, consider a haplotype sharing plan P for A. We label previously unlabeled vertices in a stepwise fashion while maintaining the properties. Pick an unlabeled vertex v from P and assign a value to its haplotype h at every sites s as follows: When there is an even-length path between v and a vertex from V_a^s, $a \in \{0, 1\}$, in G_2^s, assign $h[s] = a$ and, when there is an odd length path between v and a vertex from V_a^s, $a \in \{0, 1\}$, in G_2^s, assign $h[s] = 1 - a$. If there are no vertices from $V_0^s \cup V_1^s$ in the component of v, which means that all genotypes in the component of v have a 2-entry at site s and all vertices in the component are unlabeled, choose a value for $h[s]$ arbitrarily. The haplotypes that can be assigned to v in this way are called the *permissible haplotypes for v in P*. An

assignment of a permissible haplotype does not change the claimed properties 1 to 3. Moreover, whenever at least one vertex in every component of P is labeled by a haplotype, the permissible haplotypes for the vertices of the component are uniquely determined. □

Procedure SOLVE-CMH(A, K, k).
Input: An $n \times m$ genotype matrix A, pool constraints K and a budget k.
Output: An explaining haplotype matrix B for A with at most k distinct haplotypes that satisfies the constraints K, if it exists; or "no", otherwise.
Preprocessing:
1 **call** SANITIZE-POOL-CONSTRAINTS
2 **for each** g_i and g_j with $g_i = g_j$ **do**
3 **if** there is no pool constraint for g_j or if pool$(i, H_i) \in K$, pool$(j, H_j) \in K$, $H_i \subseteq H_j$ **then**
4 delete g_j
5 **if** there are more than $lk(k+1)/2$ genotypes **then output** "no"
Try to extend empty haplotype sharing plans:
6 **for each** empty haplotype sharing plan P of size k **do**
7 **if** P cannot be extended to a complete plan (without constraints) **then** skip P
8 **for each** component P' of P **do**
9 **if** there is a genotype g_i in P' with pool$(i, H_i) \in K$ **then**
10 $v \leftarrow$ some vertex incident to g_i
11 **for each** haplotype $h \in H_i$ that is permissible for v in P **do**
12 $P'' \leftarrow P'$; in P'' label v with h and calculate haplotypes for all vertices in P''
13 **if** P'' is a haplotype sharing plan satisfying its pool constraints **then**
14 store P'' as a solution for P' and **continue** with next P'
15 **skip** P
16 **else** choose a permissible haplotype for one vertex from P',
calculate haplotypes for all other vertices, and store the solution P''
17 combine all P'' to a complete plan for A and K and **return** combined plan
18 **output** "no"

Figure 6: The fixed-parameter algorithm for c_{pools}MH.

We remark at this point that in the absence of pool constraints, the above claim can be used to solve the problem MH, replacing the usage of a GF[2] equation system solver [4].

The algorithm proceeds to look at every component P' of P and distinguishes whether P' contains a genotype with a pool constraint or not. If there is a genotype g_i with a pool constraint pool(i, H_i), it picks one of g_i's incident vertices and iterates over all haplotypes in H_i. In line 12 it then tries to assign the haplotype to the vertex and to determine – as discussed in the proof of the claim – haplotypes for all other vertices in the component. If this succeeds in the sense that we get a complete extension of P' that satisfies the pool constraints, we save the copy and later incorporate it into the whole plan. However, if no haplotype h yields a valid extension for P', we skip the current plan P. If P' has only unconstrained genotypes, any permissible haplotype can be chosen for one vertex, which directly determines haplotypes for all other vertices in the component. If all components of a plan P can be completed, the algorithm outputs the combined haplotype sharing plan. If no plan can be completed, the algorithm outputs "no". Figure 7 shows an example of how algorithm SOLVE-CMH processes an input instance.

To bound the runtime of the algorithm, first note that after line 5 there can be at most $lk(k+1)/2$ different genotypes. The main loop iterates over all possible empty haplotype sharing plans P, of which there are at most $O(k^{2n}) \leq O(k^{lk^2+lk})$ plans. Since the inner part of the algorithm runs in polynomial time, this gives the desired fixed-parameter runtime. □

We remark the that, in practice, the runtime of SOLVE-CMH will be prohibitively high for larger k. This is mainly due to the quickly increasing number of possible empty haplotype sharing plans that the algorithm has to take into account. A promising approach to lower the number of sharing plans may be a transfer of the algorithmic ideas for

(a) Input instance (A, K, k) with $k = 5$ and

Genotypes A:	Pool constraints K:
12345	
1 21122	pool(1, {01111, 11100})
2 21122	pool(2, {01101, 11110})
3 22222	
4 02211	pool(4, {01111, 00011})
5 02211	
6 21122	pool(6, {01111, 11100, 01101, 11110})

(b) After preprocessing in lines 1 to 5

Genotypes A:	Pool constraints K:
12345	
1 21122	pool(1, {01111, 11100})
2 21122	pool(2, {01101, 11110})
3 22222	
4 02211	pool(4, {01111, 00011})

(c) Iteration over empty haplotype sharing plans of size 5 (lines 6 to 17)

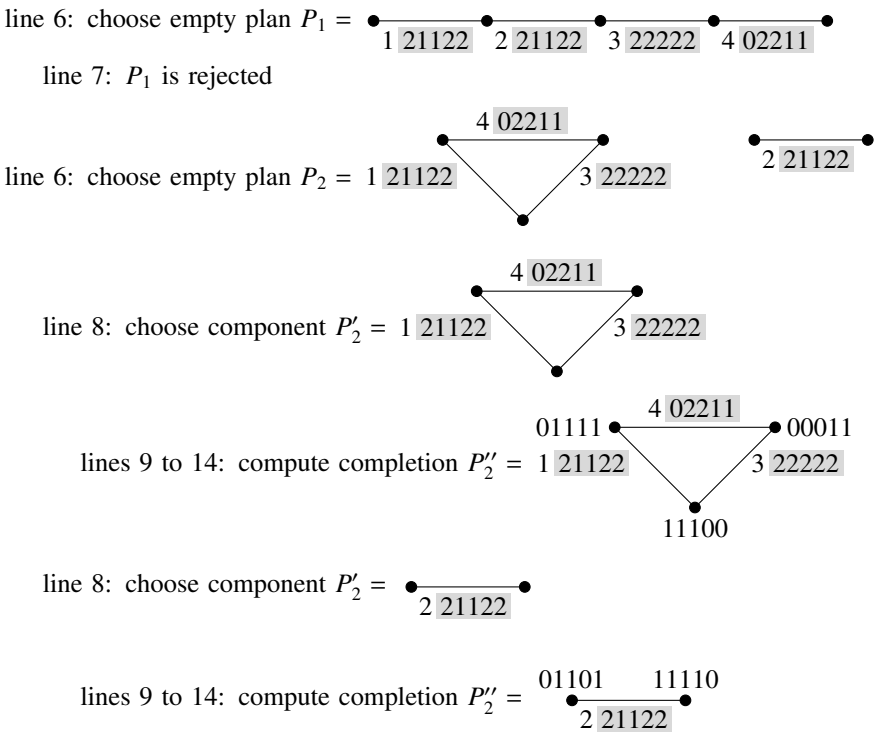

Figure 7: The figure shows an example input for the problem c_{pools}MH and how it is processed by algorithm SOLVE-CMH. (a) The example input instance (A, K, k), (b) the instance after preprocessing. Genotypes 5 and 6 are deleted in lines 2 to 4 of the algorithm. (c) The algorithm tries to extend all empty haplotype sharing plans of size 5 until a plan admitting a completion is found. The empty plan P_1 admits no completion and is rejected in line 7. The empty plan P_2 admits a completion that is returned in line 17

$c_{\text{one pool for all}}$MH from Fleischer et al. [16], who narrow the size of the search space of haplotype sharing plans from $O(k^{k^2+k})$ to $O(k^{4k})$.

5. Constrained Maximum Parsimony Perfect Phylogeny Haplotyping

In this section we show that $c_{pools,phase}$MPPH and, therefore, also MPPH and $c_{\text{one pool for all}}$MPPH, are fixed-parameter tractable with respect to the number of distinct haplotypes in the solution.

Theorem 5.1. $c_{pools,phase}$MPPH *is fixed-parameter tractable with respect to the number of distinct haplotypes in the solution.*

Similar to the algorithm for c_{pools}MH, in order to solve $c_{pools,phase}$MPPH we can equivalently search for a complete haplotype sharing plan that satisfies the constraints and whose haplotypes satisfy the four gamete property – in this context we say that the *haplotype sharing plan satisfies the four gamete property*. Pseudo-code of our algorithm is shown in Figure 8. Similar to the algorithm for $c_{pools,phase}$PPH, procedure SOLVE-CMPPH first applies some preprocessing steps and calculates a decomposition into independent submatrices with a 2-site. Procedure SOLVE-CMPPH-2-SITE then solves these matrices along with their pools via a recursive branch-and-reduce approach. In addition, to produce only solutions with a certain number of distinct haplotypes, we use haplotype sharing plans that we decompose during the recursive calls.

Procedure SOLVE-CMPPH. Given a genotype matrix A, a set of constraints K, and a budget value k, the procedure applies the same lines 1 to 5 as in the procedure from SOLVE-CPPH, which satisfy the correctness property (Section 3) and, moreover, do not change the solution size. Then it assures that there are no identical genotypes in the input: As long as possible it considers two genotypes g_i and g_j with $g_i = g_j$. If both have pool constraints pool(j, H_j), it deletes g_j and replaces pool(i, H_i) by pool$(i, H_i \cap H_j)$ and, otherwise, it deletes one of the two genotypes that has no pool constraint. This step satisfies the correctness property since, in order to satisfy the four gamete property, identical genotypes must be explained by the same haplotypes.

Next, the algorithm considers every empty haplotype sharing plan P of size at most k and calculates the partition of A into matrices $A_1, \ldots, A_m$ as in procedure SOLVE-CPPH. In order to decompose the plan P, the method labels some vertices with haplotypes: In line 8, the procedure considers every genotype without 2-entries and labels its incident vertices with this genotype. In lines 9 and 10 the procedure considers every pair of genotypes g and g' from matrices A_s and $A_{s'}$, respectively, where $s < s'$ and g and g' are incident to a common vertex v in P. The haplotype of this vertex is determined by the current instance and is set as follows: The construction of the submatrices ensures that for every site $t < s'$, we have $g'[t] \in \{0, 1\}$. Thus at these sites t we must set $h[t] = g'[t]$. For every site $t \geq s'$, we can set the entry in h directly if $g[t] \in \{0, 1\}$ or $g'[t] \in \{0, 1\}$. Whenever we have $g[t] = g'[t] = 2$, we can use phase constraints to determine the entry of h: The deduction of phase constraints in the preprocessing ensures that there is a phase constraint for s and t and, therefore, we must set $h[t] = h[s]$ whenever equal-phase$(s, t) \in K$ and $h[t] = 1 - h[s]$ whenever unequal-phase$(s, t) \in K$. Overall, every entry of h is determined.

Every matrix A_s together with its constraints K_s (the corresponding pool constraints and all phase constraints), its plan P_s (the part of P that is made up by all edges with genotypes from A_s), and its 2-site s are passed to the procedure SOLVE-CMPPH-2-SITE. Due to the preceding labeling of vertices that lie in different plans P_s and the calculated phase constraints, we can combine complete haplotype sharing plans for the subinstances into a complete haplotype sharing plan for the whole instance. Thus, provided SOLVE-CMPPH-2-SITE is correct, the whole algorithm is correct.

Procedure SOLVE-CMPPH-2-SITE. Given a genotype matrix A with a 2-site s_2, constraints K, and a haplotype sharing plan P, the procedure SOLVE-CMPPH-2-SITE recursively checks whether P can be extended to a complete plan that satisfies the constraints and the four gamete property. If all pools contain exactly two haplotypes, the phase information from these haplotypes is transferred into phase constraints (see procedure SOLVE-CPPH-2-SITE for details) and the remaining instance is solved in polynomial time using the following lemma:

Lemma 5.2. *Let A be a genotype matrix with phase constraints K and let P be a haplotype sharing plan for A. Then one can compute in polynomial time an equation system* PP-COMPLETION-EXTENSION(A, P, K) *with the following property: There exists a haplotype sharing plan P' for A that extends P and satisfies the constraints and the four gamete property if, and only if, the equation system* PP-COMPLETION-EXTENSION(A, P, K) *has a solution.*

Proof. The equation system is constructed as follows: Let A be an $n \times m$ genotype matrix, K a set of phase constraints, and P a partial haplotype sharing plan for A of size k. Let B be the haplotype labels present in P. Then the GF[2] equation system PP-COMPLETION-EQUATION(A, P, K) consists of $m(m-1)/2$ variables $p_{s,t}$ and km variables $h_{i,s}$, one for every vertex with index i and site s. The variable $p_{s,t}$ encodes the phase of sites s and t. For every pair of sites s and t with $\{00, 11\} \subseteq \text{ind}(A, s, t) \cup \text{ind}(B, s, t)$ or equal-phase$(s, t) \in K$ we introduce the equation $p_{s,t} = 0$; and for every pair of sites with $\{01, 10\} \subseteq \text{ind}(A, s, t) \cup \text{ind}(B, s, t)$ or unequal-phase$(s, t) \in K$ we introduce the equation $p_{s,t} = 1$. For every vertex v_i that is labeled by a haplotype h and every site s we introduce $h_{i,s} = h[s]$. For every genotype g_i with incident vertices v_j and v_k and site s, we introduce the equations $h_{j,s} = g_i[s]$ and $h_{k,s} = g_i[s]$ whenever $g_i[s] \in \{0, 1\}$

Procedure SOLVE-CMPPH(A, K, k).
Input: An $n \times m$ genotype matrix A, a set K of haplotype and phase constraints, and a budget k
Output: An explaining haplotype matrix B for A with at most k distinct haplotypes
that satisfies the four gamete property and the constraints K, if it exists; or "no", otherwise
Preprocessing:
1 lines 1 to 5 from procedure SOLVE-CPPH
2 **for each** g_i and g_j with $g_i = g_j$ **do**
3 **if** pool$(i, H_i) \in K$ and pool$(j, H_j) \in K$ **then**
4 replace pool(i, H_i) by pool$(i, H_i \cap H_j)$ and delete g_j
5 **else** delete one of the genotypes g_i and g_j that has no pool constraint
6 **if** there are more than $k(k+1)/2$ genotypes **then output** "no"
Try to extend empty haplotype sharing plans via decomposition:
7 **for each** empty haplotype sharing plan P of size k **do**
8 label every vertex in P that is incident to a genotype without 2-entries
9 **for each** genotype $g \in A_s$ and $g \in A_{s'}$ with $s \neq s'$ **do**
10 **if** there is a vertex v in P incident to g and g' **then** label v with the correct haplotype
11 **for each** site $s \in \{1, \ldots, m\}$ **do**
12 $P'_s \leftarrow$ SOLVE-CMPPH-2-SITE(A_s, K_s, P_s, s)
13 **if** P'_s is "no" **then** skip P
14 **return** combination of haplotypes from plans P'_s and genotypes without 2-entries.
15 **return** "no"

Procedure SOLVE-CMPPH-2-SITE(A, K, P, s_2).
Input: An $n \times m$ genotype matrix A with a 2-site s_2, constraints K, and a haplotype sharing plan P.
Output: A complete plan P' for A that extends P and satisfies the four gamete property
and the constraints if it exists; or "no", otherwise
Recursion break:
1 **if** for every pool$(i, H_i) \in K$ we have $|H_i| = 2$ **then**
2 replace all pool constraints by corresponding phase constraints
3 **return** solution for the remaining instance from PP-COMPLETION-EXTENSION(A, K, P)
Recursive branch-and-reduce
4 **else for each** group R with constraints K^R and plan P^R **do**
5 $s \leftarrow$ a site from a component of G_{cover} that corresponds to a matrix from R.
6 $P''_e \leftarrow$ **call** TRY-PHASE-CMPPH$(R, K^R \cup \{\text{equal-phase}(s_2, s)\}, P^R, s_2)$
7 $P''_u \leftarrow$ **call** TRY-PHASE-CMPPH$(R, K^R \cup \{\text{unequal-phase}(s_2, s)\}, P^R, s_2)$
8 **if** $P''_e =$ "no" and $P''_u =$ "no" **then return** "no" **else** add non-"no" P''_e or P''_u to solution
9 **output** solution

Sub-Procedure TRY-PHASE-CMPPH(R, K, P, s_2).
1 **while** the instance is modified **do**
2 **call** DEDUCE-PHASE-CONSTRAINTS for R, K
3 **call** LABEL-INTERJACENT-VERTICES for R, K, P
4 **call** SANITIZE-POOL-CONSTRAINTS for R, K
5 **for each** matrix A' from R with constraints $K' \subseteq K$ and plan $P' \subseteq P$ **do**
6 $P'' \leftarrow$ **call** SOLVE-CMPPH-2-SITE(A', K', P', s_2)
7 **if** P'' is"no" **then return** "no" **else** add P'' to solution
8 **return** solution

Figure 8: The fixed-parameter algorithm for $c_{\text{pools,phase}}$MPPH

and $h_{j,s} \oplus h_{k,s} = 1$ whenever $g_i[s] = 2$. For every genotype g_i, an incident vertex v_j, and pairs of sites s and t with $g_i[s] = g_i[t] = 2$, we introduce the equation $p_{s,t} = h_{j,s} \oplus h_{j,t}$.

The claimed equivalence now follows because the system directly models the conditions imposed by P and the solutions of the system correspond directly to complete haplotype sharing plans that satisfy the four gamete property. □

If there are pools with more than two haplotypes, the procedure SOLVE-CMPPH-2-SITE partitions the genotypes into groups R according to the following rule: Two genotypes lie in the same group whenever they are in the same submatrix with respect to the decomposition from SOLVE-CPPH-2-SITE, which we call the G_{cover} *decomposition*, or incident to a common vertex in P.

The algorithm considers each group R together with its constraints K^R (consisting of all phase and pool constraints for genotypes in R) and its subplan P^R (the part of P that is made up by all edges with genotypes from R) and picks a site s from a component of G_{cover} whose corresponding submatrix from the G_{cover} decomposition lies in R. Then, as in SOLVE-CPPH-2-SITE, it considers both possible phasings of the columns s_2 and s by calling the sub-procedure TRY-PHASE-CMPPH. This subprocedure first applies DEDUCE-PHASE-CONSTRAINTS and the procedure LABEL-INTERJACENT-VERTICES, described in the next paragraph, repeatedly until there is no further change.

The procedure LABEL-INTERJACENT-VERTICES labels previously unlabeled vertices in P for which the haplotypes are completely determined by the current instance. It iterates over all genotypes g_i and g_j from R that are incident to a common vertex v in P and labels v in the following three cases:

(1) Genotypes g_i and g_j come from different matrices A_i and A_j of the G_{cover} decomposition with components G_i and G_j, respectively, such that there are phase constraints between s_2 and all sites from G_i. At all sites t with $g_i[t] \in \{0, 1\}$ or $g_j[t] \in \{0, 1\}$, the entry of h is known. Let s_i be a site from G_i with $g_i[s_i] = 2$ and $g_j[s_j] \in \{0, 1\}$ (such a site exists due to the decomposition). At this site the value of h is determined which, together with the phase constraint for s_2 and s_i, determines the value at site s_2. Since there are phase constraints between s_2 and all sites that have 2-entries in both g_i and g_j, the entries at these positions are also known. Altogether, this gives a haplotype for g_j and forces the phase between s_2 and a site $s_j \in G_j$ with $g_j[s_j] = 2$. We introduce equal-phase(s_2, s_j) whenever $h[s_2] = h[s_j]$ and unequal-phase(s_2, s_j) whenever $h[s_2] \neq h[s_j]$.

(2) Genotype g_i comes from a matrix A_i of the G_{cover} decomposition, such that there are phase constraints between s_2 and all sites from G_i, and g_j is not part of any matrix of the decomposition. Due to the decomposition there exists a site s_i from G_i with $g_i[s_i] = 2$ and $g_j[s_i] \in \{0, 1\}$. Thus, similar to the previous case, the value at site s_i determines the value at site s_2 and, indirectly, values for all sites where both genotypes have a 2-entry. Altogether, h is completely determined.

(3) Genotypes g_i and g_j are not part of any matrix of the decomposition. Let s_i be a site where the genotypes differ. One of the genotypes, say g_i, must have a 2-entry at site s_i since, otherwise, the sites s_2 and s_i induce all four gametes. Again, we know the value of the haplotype h labeling v at position s_i which determines the value at site s_2 and all other sites where both genotypes have a 2-entry.

The effect of the while-loop in lines 1 to 3 of TRY-PHASE-CMPPH is that there are new phase constraints between s_2 and all sites from G_{cover} components whose matrices lie in R and all vertices in P that are incident to genotypes from different components are labeled by haplotypes. Then we apply SANITIZE-POOL-CONSTRAINTS and pass every matrix of the G_{cover} decomposition along with its plan and constraints to a recursive call. If this leads to solutions for all matrices, the sub-procedure TRY-PHASE-CMPPH combines them to a solution for R.

In line 8 of SOLVE-CMPPH-2-SITE, if for every group at least one possibility for the phase of s_2 and s gives a solution for all matrices from the group, the algorithm adds them to an extension for the whole plan P. Otherwise, the algorithm outputs "no".

Runtime. The runtime of the algorithm is dominated by the iteration over at most $O(k^{2n}) \leq O(k^{k^2+k})$ empty plans. For every possible plan the algorithm recursively tries new phase constraints and partitions the genotypes, the plan, and the haplotypes from the pools. Similar to SOLVE-CPPH-2-SITE, the corresponding tree of recursive calls has only polynomial size, which proves the desired fixed-parameter runtime.

6. Conclusion

We studied phylogeny- and parsimony-based haplotype inference in the presence of pool and phase constraints. Our main result is that $c_{\text{pools,phase}}$PPH is polynomial-time solvable by a new recursive decomposition techniques for genotypes and pools. This solves the open problem from [3] whether $c_{\text{one pool for all}}$PPH is polynomial-time solvable. We showed that c_{pools}MH is W[2]-hard by proving that $c_{\text{pools for all}}$MH is W[2]-complete when parametrized by the number of distinct haplotypes in the solution. Both problems are fixed-parameter tractable when we also use the comparability of the pools as a parameter. For c_{pools}MH we presented an algorithm that extends the recursive decomposition of genotypes and pools by a decomposition of haplotype sharing plans, yielding a fixed-parameter algorithm for $c_{\text{pools,phase}}$MPPH with respect to the number of distinct haplotypes in the solution.

There are several research directions for future work: (1) *Work with incomplete genotype data:* A large body of work has investigated the more difficult – and more realistic – variant of PPH where some input data may be missing, the *incomplete perfect phylogeny haplotyping problem* (IPPH). This problem is NP-complete [25], also if a haplotype without missing entries is known beforehand [26] and the phylogenetic tree has the topology of a path [27], but there is a fixed-parameter algorithm for trees with a bounded number of branches [28]. A natural direction would be to adjust our ideas to algorithms that work on the constrained variants of IPPH. (2) *More general constraints:* Another direction is to incorporate $*$-constraints where some entries in the haplotypes can be chosen freely. Or we may also try to allow a few additional rare haplotypes to be used that are not in any pool. (3) *Other biological objectives:* The perfect phylogeny assumption is applicable to regions of the genome whose evolutionary history can be described by using at most one mutation per site. If we also want to cover genomic regions that are subject to back mutations or recombinations, we need to use other phylogenetic models. One research direction would be to pick up recent work on haplotype inference problems that are based on extensions of the perfect phylogeny model [29, 30, 31] and study the constrained versions of these problems.

References

[1] M. Elberfeld, T. Tantau, Phylogeny- and parsimony-based haplotype inference with constraints, in: Proceedings of the 21st Annual Symposium on Combinatorial Pattern Matching (CPM 2010), Vol. 6129 of Lecture Notes in Computer Science, 2010, pp. 177–189.

[2] L. Excoffier, M. Slatkin, Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population, Molecular Biology and Evolution 12 (5) (1995) 921–7.
URL http://mbe.oxfordjournals.org/cgi/content/abstract/12/5/921

[3] M. R. Fellows, T. Hartman, D. Hermelin, G. M. Landau, F. A. Rosamond, L. Rozenberg, Haplotype inference constrained by plausible haplotype data, in: Proceedings of the 20th Annual Symposium on Combinatorial Pattern Matching (CPM 2009), Vol. 5577 of Lecture Notes in Computer Science, Springer, 2009, pp. 339–352.
URL http://dx.doi.org/10.1007/978-3-642-02441-2_30

[4] R. Sharan, B. V. Halldórsson, S. Istrail, Islands of tractability for parsimony haplotyping, IEEE/ACM Transactions on Computational Biology and Bioinformatics 3 (3) (2006) 303–311.
URL http://dx.doi.org/10.1109/TCBB.2006.40

[5] D. Gusfield, Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions, in: Proceedings of the 6th Annual International Conference on Computational Biology (RECOMB 2002), ACM Press, 2002, pp. 166–175.
URL http://dx.doi.org/10.1145/565196.565218

[6] V. Bafna, D. Gusfield, G. Lancia, S. Yooseph, Haplotyping as perfect phylogeny: A direct approach, Journal of Computational Biology 10 (3–4) (2003) 323–340.
URL http://dx.doi.org/10.1089/10665270360688048

[7] E. Eskin, E. Halperin, R. M. Karp, Efficient reconstruction of haplotype structure via perfect phylogeny, Journal of Bioinformatics and Computational Biology 1 (1) (2003) 1–20.
URL http://dx.doi.org/10.1142/S0219720003000174

[8] Z. Ding, V. Filkov, D. Gusfield, A linear-time algorithm for the perfect phylogeny haplotyping (PPH) problem, Journal of Computational Biology 13 (2) (2006) 522–553.
URL http://dx.doi.org/10.1089/cmb.2006.13.522

[9] P. Bonizzoni, A linear-time algorithm for the perfect phylogeny haplotype problem, Algorithmica 48 (3) (2007) 267–285.
URL http://dx.doi.org/10.1007/s00453-007-0094-3

[10] Y. Liu, C.-Q. Zhang, A linear solution for haplotype perfect phylogeny problem, in: Proceedings of the International Conference on Bioinformatics and its Applications, World Scientific, 2005, pp. 173–184.
URL http://dx.doi.org/10.1142/9789812702098_0016

[11] R. V. Satya, A. Mukherjee, An optimal algorithm for perfect phylogeny haplotyping, Journal of Computational Biology 13 (4) (2006) 897–928.
URL http://dx.doi.org/10.1089/cmb.2006.13.897

[12] M. Elberfeld, Perfect phylogeny haplotyping is complete for logspace, Computing Research Repository (CoRR) abs/0905.0602.
URL http://arxiv.org/abs/0905.0602
[13] M. Elberfeld, T. Tantau, Computational complexity of perfect-phylogeny-related haplotyping problems, in: Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2008), Vol. 5162 of Lecture Notes in Computer Science, Springer, 2008, pp. 299–310.
URL http://dx.doi.org/10.1007/978-3-540-85238-4_24
[14] D. Gusfield, Haplotype inference by pure parsimony, in: Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003), Vol. 2676 of Lecture Notes in Computer Science, Springer, 2003, pp. 144–155.
URL http://dx.doi.org/10.1007/3-540-44888-8_11
[15] G. Lancia, M. C. Pinotti, R. Rizzi, Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms, INFORMS Journal on Computing 16 (4) (2004) 348–359.
URL http://dx.doi.org/10.1287/ijoc.1040.0085
[16] R. Fleischer, J. Guo, R. Niedermeier, J. Uhlmann, Y. Wang, M. Weller, X. Wu, Extended islands of tractability for parsimony haplotyping, in: Proceedings of the 21st Annual Symposium on Combinatorial Pattern Matching (CPM 2010), Vol. 6129 of Lecture Notes in Computer Science, Springer, 2010, pp. 214–226.
URL http://dx.doi.org/10.1007/978-3-642-13509-5_20
[17] D. G. Brown, I. M. Harrower, Integer programming approaches to haplotype inference by pure parsimony, IEEE/ACM Transactions on Computational Biology and Bioinformatics 3 (2) (2006) 141–154.
URL http://dx.doi.org/10.1109/TCBB.2006.24
[18] L. Wang, Y. Xu, Haplotype inference by maximum parsimony, Bioinformatics 19 (14) (2003) 1773–1780.
URL http://dx.doi.org/10.1093/bioinformatics/btg239
[19] G. Jäger, S. Climer, W. Zhang, Complete parsimony haplotype inference problem and algorithms, in: Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009), Vol. 5757 of Lecture Notes in Computer Science, Springer, 2009, pp. 337–348.
URL http://dx.doi.org/10.1007/978-3-642-04128-0_31
[20] V. Bafna, D. Gusfield, S. Hannenhalli, S. Yooseph, A note on efficient computation of haplotypes via perfect phylogeny, Journal of Computational Biology 11 (5) (2004) 858–866.
URL http://dx.doi.org/10.1089/cmb.2004.11.858
[21] L. van Iersel, J. Keijsper, S. Kelk, L. Stougie, Shorelines of islands of tractability: Algorithms for parsimony and minimum perfect phylogeny haplotyping problems, IEEE/ACM Transactions on Computational Biology and Bioinformatics 5 (2) (2008) 301–312.
URL http://dx.doi.org/10.1109/TCBB.2007.70232
[22] Y.-T. Huang, K.-M. Chao, T. Chen, An approximation algorithm for haplotype inference by maximum parsimony, Journal of Computational Biology 12 (10) (2005) 1261–1274.
URL http://dx.doi.org/10.1089/cmb.2005.12.1261
[23] A. M. Andrés, A. G. Clark, L. Shimmin, E. Boerwinkle, C. F. Sing, J. E. Hixson, Understanding the accuracy of statistical haplotype inference with sequence data of known phase, Genetic Epidemiology 31 (7) (2007) 659–671.
URL http://dx.doi.org/10.1002/gepi.20185
[24] R. G. Downey, M. R. Fellows, Parameterized Complexity, Springer-Verlag, New York, 1999.
[25] M. Steel, The complexity of reconstructing trees from qualitative characters and subtrees, Journal of Classification 9 (1) (1992) 91–116.
URL http://dx.doi.org/10.1007/BF02618470
[26] G. Kimmel, R. Shamir, The incomplete perfect phylogeny haplotype problem, Journal of Bioinformatics and Computational Biology 3 (2) (2005) 359–384.
URL http://dx.doi.org/10.1142/S0219720005001090
[27] J. Gramm, T. Nierhoff, R. Sharan, T. Tantau, Haplotyping with missing data via perfect path phylogenies, Discrete and Applied Mathematics 155 (6–7) (2007) 788–805.
URL http://dx.doi.org/10.1016/j.dam.2005.09.020
[28] M. Elberfeld, I. Schnoor, T. Tantau, Influence of tree topology restrictions on the complexity of haplotyping with missing data, in: Proceedings of the 6th Annual Conference on Theory and Applications of Models of Computation (TAMC 2009), Vol. 5532 of Lecture Notes in Computer Science, Springer, 2009, pp. 201–210.
URL http://dx.doi.org/10.1007/978-3-642-02017-9_23
[29] Y. Song, Y. Wu, D. Gusfield, Algorithms for imperfect phylogeny haplotyping (IPPH) with a single homoplasy or recombination event, in: Proceedings of the 5th International Workshop on Algorithms in Bioinformatics (WABI 2005), no. 3692 in Lecture Notes in Computer Science, Springer, 2005, pp. 152–164.
URL http://dx.doi.org/10.1007/11557067_13
[30] A. Gupta, J. Maňuch, L. Stacho, X. Zhao, Haplotype inferring via Galled-Tree networks is NP-complete, in: Proceedings of the 14th Annual International Conference on Computing and Combinatorics (COCOON 2008), no. 5092 in Lecture Notes in Computer Science, Springer, 2008, pp. 287–298.
URL http://dx.doi.org/10.1007/978-3-540-69733-6_29
[31] A. Gupta, J. Maňuch, L. Stacho, X. Zhao, Haplotype inferring via galled-tree networks using a hypergraph covering problem for special genotype matrices, Discrete Applied Mathematics 157 (10) (2009) 2310–2324.
URL http://dx.doi.org/10.1016/j.dam.2008.06.051