Influence of Tree Topology Restrictions on the Complexity of Haplotyping with Missing Data

Michael Elberfeld Ilka Schnoor Till Tantau

Institut für Theoretische Informatik
Universität zu Lübeck
D-23538 Lübeck, Germany
{elberfeld,schnoor,tantau}@tcs.uni-luebeck.de

Abstract

Haplotyping, also known as haplotype phase prediction, is the problem of predicting likely haplotypes based on genotype data. One fast haplotyping method is based on an evolutionary model where a perfect phylogenetic tree is sought that explains the observed data. Unfortunately, when data entries are missing, which is often the case in laboratory data, the resulting formal problem IPPH, which stands for incomplete perfect phylogeny haplotyping, is NP-complete and no theoretical results are known concerning its approximability, fixed-parameter tractability, or exact algorithms for it. Even radically simplified versions, such as the restriction to phylogenetic trees consisting of just two directed paths from a given root, are still NP-complete; but here, at least, a fixed-parameter algorithm is known. We show that such drastic and ad hoc simplifications are not necessary to make IPPH fixed-parameter tractable: we present the first theoretical analysis of an algorithm, which we develop in the course of the paper, that works for arbitrary instances of IPPH. On the negative side we show that restricting the topology of perfect phylogenies does not always reduce the computational complexity: while the incomplete directed perfect phylogeny problem is well-known to be solvable in polynomial time, we show that the same problem restricted to path topologies is NP-complete.

Classification: computational biology, computational complexity, fixed-parameter algorithms

1 Introduction

Haplotype phase prediction is an important preprocessing step in genomic disease and medical condition association studies. In these studies two groups of people are considered, where one group has a certain disease or medical condition while the other has not, and one tries to find correlations between group membership and the genomic data of the individuals in the groups. The genomic data typically consists of information about which bases are present in an individual's DNA at so-called SNP sites (single nucleotide polymorphism sites). While the DNA sequences of different individuals are mostly identical, at SNP sites there may be variations. Low-priced methods for large-scale inference of genomic data can read out, separately for each SNP site, the bases present, of which there can be two since we inherit one chromosome from our father and one from our mother. However, since the bases at different sites are determined independently, we have no information on which chromosome a base belongs to. For *homozygous sites,* where the same base is present on both chromosomes, this is not a problem, but for *heterozygous sites* this information, called the *phase* of an SNP site, is needed for accurate correlations. The idea behind *haplotype phase prediction* or just *haplotyping* is to computationally predict likely phases based on the laboratory data (which misses this information). For an individual, the genomic input data without phase information is called the *genotype* while the two predicted chromosomes are called *haplotypes*.

From a mathematical point of view, haplotypes can be conveniently coded as strings over the alphabet $\{0,1\}$, where for a given site 0 stands for one of the bases that can be observed in practice, while 1 encodes a second base that can also be observed. (The case that three bases are observed happens so seldom that it can be ignored.) A genotype g is, conceptually, a sequence of sets that arises from two haplotypes h_1 and h_2 as follows: The ith set in the sequence g is $\{h_1[i], h_2[i]\}$. However, it is customary to encode the set $\{0\}$ as 0, to encode $\{1\}$ as 1, and $\{0,1\}$ as 2, so that a genotype is actually a string over the alphabet $\{0,1,2\}$. For example, the two haplotypes 0110 and 0101 give rise to (we also say *explain*) the genotype 0122; and so do 0100 and 0111.

Since different haplotype pairs can explain the same genotype and any single haplotype is equally likely *a priori,* haplotyping is not possible if only a single genotype is given. However, if a whole set of genotypes from a larger group of different individuals is given, certain sets of haplotypes that explain these genotypes are more likely than others. For instance, a small set of explaining haplotypes is more likely than a large set since haplotypes mutate only rarely. It is customary to formalize sets of genotypes as matrices (each row is a genotype) and also sets of explaining haplotypes (each row contains a haplotype and rows $2i-1$ and $2i$ of the haplotype matrix explain exactly the genotype in row i of the genotype matrix).

One important method of haplotyping is based on the *perfect phylogeny approach* proposed by Gusfield [13]. The idea is to seek a haplotype matrix that explains the genotype matrix and whose rows (which are the haplotypes) can be arranged in a *perfect phylogenetic tree*. This means the following: A haplotype matrix B *admits a perfect phylogeny* if there exists a tree (an undirected, connected, acyclic graph) T_B such that:

1. Each column of B labels exactly one edge of T_B and each edge is labeled by at least one column.
2. Each row of B labels exactly one node of T_B.
3. For every two rows h_1 and h_2 of B and every column i, we have $h_1[i] \neq h_2[i]$ if, and only if, i lies on the path from h_1 to h_2 in T_B.

The intuition behind these properties is as follows. The nodes of the tree T_B correspond to haplotypes. The edges between the nodes correspond to mutation events: When we move from one node to another node along a single edge, the label(s) of the edge name exactly those columns in which the node labels differ. This means that when we remove an edge labeled by a column c, the two resulting components have the property that all nodes in one component have a 0 in column c and all nodes in the other component have a 1 in that column.

When a haplotype matrix B admits a perfect phylogeny T_B and, at the same time, explains a genotype matrix A, we also say that A *admits a perfect phylogeny*. In this case, it is useful to define a tree T_A as follows: Its topology is the same as T_B's and so are the node labels, but the edges are labeled by the columns of A (which may contain 2-entries) instead of the columns of B (where each 2-entry is replaced by a 0-entry and a 1-entry). We call T_A *a perfect phylogeny for A*. The formal *perfect phylogeny haplotyping* problem (PPH) is the set of all genotype matrices that admit a perfect phylogeny.

Gusfield [13] showed that PPH is solvable in polynomial time. However, in practice, laboratory data is never perfect and some entries may be missing in the input genotype matrices. In this case, the input matrices may contain ?-entries in addition to the 0-, 1-, and 2-entries. The objective is then to replace the missing entries by normal entries such that the resulting matrix is in PPH. This problem is known as IPPH, where the I stands for *incomplete* (in the following, when we prefix a problem with the letter I, we mean that the input matrix may contain ?-entries and the objective is to fill them up so that the resulting matrix is an instance of the problem without the I). Unfortunately, IPPH is NP-complete [20]. A heuristic is known for solving it [19], but no guarantees can be made concerning its runtime.

In order to tackle the problem, one can try to exploit properties of typical input data that may make the problem easier to solve. The first simplification is the notion of *directedness*. In real data, some genotype is typically completely known and is completely homozygous, which means that one haplotype of the sought haplotype matrix is already known. Since the roles of 0-entries and 1-entries can be exchanged individually for each column, we may assume that the known haplotype is the all-0-haplotype. This problem variant is called "directed" because the position of the all-0-haplotype in the phylogenetic tree singles out a node, which is then regarded as the root of the tree and gives an orientation to the tree. The resulting problem is called IDPPH, with D standing for "directed." It is still NP-complete [15].

A second, rather radical simplification (which is nevertheless often backed by the data) was proposed by Gramm, Nierhoff, Sharan and the third author [11]: In addition to being directed, we required that the (undirected) phylogenetic tree must form a simple path. The resulting problem was called incomplete directed perfect *path* phylogeny haplotyping. It is *still* NP-complete, but in [11] we presented a fixed-parameter algorithm for it, where the parameter is the maximum number of ?-entries per column.

Another radical simplification is to study matrices in which no heterozygous sites are found. This is the same as getting already phased haplotypes as input and the question is just whether they can be arranged in a perfect phylogeny: for the problem IPP (note the missing H, since no haplotyping needs to be done) we get an incomplete haplotype matrix and must decide whether the missing entries can be filled up with 0-entries and 1-entries so that the completed matrix admits a perfect phylogeny. This problem is still NP-complete [20], but the directed variant IDPP is solvable in polynomial time [2, 17].

In the present paper we further the study of the computational complexity of IPPH and the above variants. We are especially interested in the following question: How do restrictions on the tree topology influence the complexity of the problem? In other words, what is the complexity of $\text{IPPH}_{\text{leafs}\leq l}$, where the explaining phylogenetic tree may have at most l leafs. As stated above, the best known result is that $\text{IDPPH}_{\text{leafs}\leq 2}$ is NP-complete, but lies in FPT.

Let us summarize the notation: For the basic problem PP we get a (complete) haplotype matrix as input and the question is whether it admits a perfect phylogeny. We add the prefix D to indicate that one of the nodes in the phylogeny must be labeled with a given root haplotype and, since the roles of 0- and 1-entries are interchangeable, we insist that it must be the all-0-haplotype. We append H to indicate that the input is a genotype matrix and an explaining haplotype matrix admitting a perfect phylogeny must be found. We add the prefix I to indicate that ?-entries may be present in the input. We add the index "leafs $\leq l$" to indicate that only perfect phylogenies having at most l leafs are allowed (a leaf is a degree 1 node in the undirected graph underlying the perfect phylogeny).

Our Contributions. Our first main result, Theorem 2.1 presented in Section 2, is the following hardness result: $\text{IDPP}_{\text{leafs}\leq l}$ *is* NP-*complete for every* $l \geq 2$. In sharp contrast, IDPP $\in$ P. As detailed in the section on related work, past experience has indicated that restricting the topology of perfect phylogenies makes haplotyping problems easier, not harder. Theorem 2.1 shows that IDPP is a notable exception. Naturally, there are other examples of such exceptions: Finding a spanning tree for a graph is easy, finding a spanning path is hard.

The problem $\text{IDPP}_{\text{leafs}\leq l}$ reduces to many other problems. It is easy to see (but not trivial) that a problem where the input matrix consists of (possibly incomplete) haplotypes reduces to the same problem for (possibly incomplete) genotype matrices via the identity mapping. Thus, the first result implies that $\text{IDPPH}_{\text{leafs}\leq l}$ is also NP-complete, which was previously proved by Gramm et al. [11] for $l = 2$. It is also easy to see that any directed problem reduces to the undirected version by adding an all-0-row. Thus, $\text{IPP}_{\text{leafs}\leq l}$ is also NP-complete. Indeed, all previously known NP-completeness results for variants of IPPH follow from Theorem 2.1, except for the NP-completeness of IPP.

Our second main contribution, presented in Section 3, is an algorithm for solving IPPH that allows a rigorous runtime analysis. In detail, we present an algorithm that on input of a number l and an incomplete $n \times m$ genotype matrix A with at most k many ?-entries per column correctly outputs:

1. Either "$A \notin \text{IPPH}_{\text{leafs}\leq l}$" or
2. a completion of A and a perfect phylogeny for this completion with at most l leafs.

The algorithm's runtime is $f(k,l)n^2m^{O(l)}$, where f is some function. (We only present a decision procedure for IPPH$_{\text{leafs}\leq j}$ in the present paper, but using standard dynamic programming techniques it can easily be modified to also output the desired phylogeny.)

This algorithm allows us to make formal statements about the fixed-parameter tractability of IPPH. First, IPPH lies in the class XP for the parameter pair (k,l). Second and more importantly, for each fixed $l \geq 2$ the problem IPPH$_{\text{leafs}\leq l}$ is fixed-parameter tractable with respect to the number of unknown entries per column, see Theorem 3.1. This settles the central problem that we had to leave open in [11], namely whether the result that IDPPH$_{\text{leafs}\leq 2}$ is fixed-parameter tractable also holds for the undirected case and for larger number of leafs. On both accounts, we answer this question affirmatively.

Due to lack of space, we only sketch proofs in the main text, see the appendix for detailed proofs.

Related Work. Haplotyping methods can be split into two groups: Statistical, see [9] for a literature starting point, and combinatorial. There are two main combinatorial methods: Maximum parsimony haplotyping [4, 12] and the more recent perfect phylogeny approach that was introduced by Gusfield [13] and later explored by numerous authors [1, 3, 5, 8, 16, 18].

The idea of considering restricted tree topologies to speed up haplotyping is due to Gramm et al. [11] and was recently also investigated in the context of finding block partitions [10]. A different approach to deal with the NP-completeness of IPPH is due to Halperin and Karp [14]. They present a polynomial-time algorithm for IPPH that works for special instances satisfying the so-called "rich data hypothesis."

For complete data, numerous results on the complexity of PPH and its variants are known. Gusfield showed that the problem can be solved in polynomial time [13], further papers first presented simpler polynomial-time algorithms [1, 8] and later even linear-time algorithms [3, 5, 16, 18]. In [7] we have shown that PPH is hard for logarithmic space and lies in NC^2.

The influence of restricting the tree topology on the complexity of haplotyping problems has, prior to the present paper, always been benign: In [11] it is shown that IDPPH$_{\text{leafs}\leq 2}$ has a fixed-parameter algorithm, which is not known to be the case for IPPH. In [10] it is shown that partitioning a complete genotype matrix into a minimal number of column sets such that each set admits a perfect *path* phylogeny is equivalent, in complexity theoretic terms, to finding maximal matchings; while the same problem for arbitrary perfect phylogenies is NP-hard and even very hard to approximate. Finally, in [7] it is shown that DPPH$_{\text{leafs}\leq 2}$ lies in AC^0, while DPPH is L-hard.

2 Hardness Result

In the present section we prove our first main result:

Theorem 2.1. IDPP$_{\text{leafs}\leq l}$ *is* NP-*complete for every* $l \geq 2$.

Since IDPP $\in$ P, this is a first example of a perfect *path* phylogeny problem being harder than the corresponding problem for general perfect phylogenies. Our proof is based on a reduction from the NP-complete problem MONOTONE NAE3SAT and is similar to the reduction presented in [11], which starts, however, from NAE3SAT. By starting our reduction from a (conceptually) simpler problem we are able to prove a stronger result than the one presented in [11].

Sketch of proof. Fix an $l \geq 2$. We reduce MONOTONE NAE3SAT to IDPP$_{\text{leafs}\leq l}$. The input for the reduction is a propositional formula ϕ in conjunctive normal form with three positive literals per clause and the question is whether there is a variable assignment such that not all literals of a clause share the same truth value. (The problem MONOTONE NAE3SAT is NP-hard by a reduction from NAE3SAT: replace each negated variable $\overline{x_i}$ by a new variable y_i and append the clause $y_i \vee x_i \vee x_i$.)

Let ϕ have n variables $v_1,\dots,v_n$ and m clauses $C_1,\dots,C_m$. We construct an incomplete $(n+3m+l-2)\times(3m+l-2)$ haplotype matrix B. The first n rows, which we call *variable rows*, are identified with the variables of ϕ. The next $3m$ rows and the first $3m$ columns are called *literal rows* and *literal columns*, respectively. The remaining $l-2$ columns are marked by $b_1, \dots, b_{l-2}$. First, we describe the non-?-entries of the upper left $(n+3m)\times(3m)$ submatrix: Let C_j be a clause of ϕ with literals $\{l_j^1, l_j^2, l_j^3\}$. For each literal l_j^k and its corresponding variable v_i, we put a 1-entry in row v_i and column l_j^k. Then we put the submatrix $\left[\begin{smallmatrix} 1 & 0 & ? \\ ? & 1 & 0 \\ 0 & ? & 1 \end{smallmatrix}\right]$ in columns l_j^1, l_j^2 and l_j^3 and rows l_j^1, l_j^2 and l_j^3. Finally, we set the lower right $(l-2)\times(l-2)$ submatrix to the identity matrix and all entries in the upper right $(n+3m)\times(l-2)$ submatrix and the lower left $(l-2)\times 3m$ submatrix to 0. An example of this construction for $l=4$ is depicted in Figure 1.

$$\phi = C_1 \wedge C_2 \wedge C_3 \quad \text{with}$$

$$C_1 = (\underbrace{v_1}_{l_1^1} \vee \underbrace{v_2}_{l_1^2} \vee \underbrace{v_3}_{l_1^3}),$$

$$C_2 = (\underbrace{v_2}_{l_2^1} \vee \underbrace{v_3}_{l_2^2} \vee \underbrace{v_4}_{l_2^3}),$$

$$C_3 = (\underbrace{v_2}_{l_3^1} \vee \underbrace{v_4}_{l_3^2} \vee \underbrace{v_5}_{l_3^3})$$

is mapped to

$$\begin{array}{c|ccccccccc|cc|}
 & l_1^1 & l_1^2 & l_1^3 & l_2^1 & l_2^2 & l_2^3 & l_3^1 & l_3^2 & l_3^3 & b_1 & b_2 \\
\hline
v_1 & 1 & & & & & & & & & & \\
v_2 & & 1 & & 1 & & & 1 & & & & \\
v_3 & & & 1 & & 1 & & & & & & \\
v_4 & & & & & & 1 & & 1 & & & \\
v_5 & & & & & & & & & 1 & & \\
l_1^1 & 1 & 0 & ? & & & & & & & & \\
l_1^2 & ? & 1 & 0 & & & & & ? & & \multicolumn{2}{c|}{0} \\
l_1^3 & 0 & ? & 1 & & & & & & & & \\
l_2^1 & & & & 1 & 0 & ? & & & & & \\
l_2^2 & & & & ? & 1 & 0 & & & & & \\
l_2^3 & & & & 0 & ? & 1 & & & & & \\
l_3^1 & & & & & & & 1 & 0 & ? & & \\
l_3^2 & & ? & & & & & ? & 1 & 0 & & \\
l_3^3 & & & & & & & 0 & ? & 1 & & \\
\hline
 & \multicolumn{9}{c|}{0} & 1 & 0 \\
 & & & & & & & & & & 0 & 1 \\
\hline
\end{array}$$

Figure 1: Example of the reduction from MONOTONE NAE3SAT to IDPP$_{\text{leafs}\leq 4}$.

It remains to prove that $\phi \in$ MONOTONE NAE3SAT if, and only if, $B \in$ IDPP$_{\text{leafs}\leq l}$. To obtain a not-all-equal assignment for ϕ from a perfect phylogeny for B, one argues that the literal columns are distributed on exactly two paths and the literals on one of these paths can be set to true, the others to false. For the other direction, a not-all-equal assignment is turned into a phylogenetic tree by placing all literal columns set to true on one path and the other literals on a second path. $\square$

As mentioned in the introduction, IDPP$_{\text{leafs}\leq l}$ can easily be reduced to many other problems, including IPP$_{\text{leafs}\leq l}$, IDPPH$_{\text{leafs}\leq l}$, IPPH$_{\text{leafs}\leq l}$, IDPPH, and IPPH. Thus, as was already known for $l=2$, all of these problems are also NP-complete.

3 Fixed Parameter Tractability Result

In this section we show that, for every fixed $l \geq 2$, the problem IPPH$_{\text{leafs}\leq l}$ is fixed-parameter tractable with respect to the maximal number of missing entries per column:

Theorem 3.1. *For each $l \geq 2$, the problem* IPPH$_{\text{leafs}\leq l}$ *is fixed-parameter tractable with respect to the maximal number of ?-entries per column.*

The theorem generalizes the result from [11] by Gramm, Nierhoff, Sharan and the third author that IDPPH$_{\text{leafs}\leq 2}$ is fixed-parameter tractable. The algorithm from [11] relies strongly on Gusfield's characterization [13]: Given a genotype matrix A, a directed perfect phylogeny T for it, and any genotype g of A, the 1-entries of g label a path from the root to some node v of T and the 2-entries of g label a path containing v. Most algorithms for PPH and its variants from the literature exploit this property as follows: They first reduce the problem to the directed version DPPH and then build the phylogeny by placing columns with many 1-entries and 2-entries near to the root and columns with fewer such entries far from the root. The notions "should be placed near to the root" and "should be placed far from the root" can be quantified more precisely using Gusfield's notion of the leaf count of a column [13].

When the data is incomplete and question marks are present, no reduction from the undirected to the directed case is known. (Indeed, IPP is NP-complete while IDPP $\in$ P.) To solve the undirected problem variant $\text{IPPH}_{\text{leafs} \leq l}$ we need a replacement for the notion of leaf counts and a new characterization of genotype matrices admitting undirected perfect phylogenies. We present such a replacement, which we call the *light component size*, and also a new characterization in terms of the new notion of *mutation trees*. This characterization allows us to construct phylogenies in a stepwise fashion from the "outside" (columns far removed from the root, having a small light component size) to the "inside" (columns near to the root, having a large light component size). In each step, we only need to remember the inner part of the partial phylogeny constructed so far, making a dynamic program feasible.

In the following two sections, we first introduce the new notion and characterization and then show how they can be used in an algorithm.

3.1 A Characterization of Undirected Perfect Phylogeny Haplotyping

A major tool in the development of efficient algorithms for the DPPH problem has been the *leaf count* of a column, which is twice the number of 1-entries plus the number of 2-entries. The name "leaf count" stems from the following observation: In a perfect phylogeny for a genotype matrix A, the number of haplotypes (which are typically attached to leafs) below the edge labeled by a column equals exactly its leaf count. This means that if two columns occur on a path from the root (recall that this is always the all-0-haplotype in a directed perfect phylogeny) to a leaf, the column with a greater leaf count is located nearer to the root.

For undirected perfect phylogenies the leaf count is no longer meaningful since there is no distinguished root node that is known in advance. To tackle this problem, we introduce the new notion of *light component sizes.* For a column c and $x \in \{0,1,2\}$ let $n_x(c)$ denote the number of x-entries in c.

Definition 3.2. For a column c of a genotype matrix A its *light component size* and *heavy component size* are defined as follows:

$$\begin{aligned} \text{lcs}(c) &:= n_2(c) + 2 \cdot \min\{n_0(c), n_1(c)\}, \\ \text{hcs}(c) &:= n_2(c) + 2 \cdot \max\{n_0(c), n_1(c)\}. \end{aligned}$$

The key observation is that when we remove an edge labeled by a column c from a perfect phylogeny T_A, then two components result and the number of node labels in one of these components will be $\text{lcs}(c)$ and we call the component the *light component*, the other will contain $\text{hcs}(c)$ labels and we call it the *heavy component*. (In case $\text{lcs}(c) = \text{hcs}(c)$, the choice is arbitrary.) To see this, recall from the introduction that in one component all node labels have a 0 in column c (and a 1 in the other component). Each of the $n_0(c)$ many 0-entries of c contributes two nodes labels to this component, while each 2-entry contributes one node label, which means that the number of node labels in this component is either $\text{lcs}(c)$ or $\text{hcs}(c)$. The argument is similar for the other component and for 1-entries.

We have just seen that the value in column c of all node labels of the light component is the same. Let us call this value the *light component value* $\text{lcv}(c)$. Clearly, $\text{lcv}(c) = 0$ if $n_0(c) < n_1(c)$ and $\text{lcv}(c) = 1$ if $n_0(c) > n_1(c)$. For $n_0(c) = n_1(c)$ we remarked earlier that the light component can be chosen arbitrarily; at this point we implicitly fix that choice by setting $\text{lcv}(c) = 1$. Symmetrically, the value in column c of the node labels of the heavy component are all the same and equal to $\text{hcv}(c) = 1 - \text{lcv}(c)$.

Our next aim is to define a quasi-ordering $\preceq$ on columns that tells us something about how columns can possibly be arranged in a perfect phylogeny. Suppose that for two columns c and d we know that the light component of d is a superset of the light component of c. Consider a node label l and suppose the value of l at the position of column c happens to be the light component value of c. Then we know that l must lie in the light component of c and, thus, also in the light component of d, which in turn means that at position d in l we must have the light component value of d. Phrased more succinctly: for every $i \in \{1,\dots,n\}$ we have $c[i] = \mathrm{lcv}(c) \implies d[i] = \mathrm{lcv}(d)$ and, by a similar argument, also $d[i] = \mathrm{hcv}(d) \implies c[i] = \mathrm{hcv}(c)$. Let us write $c \preceq d$ whenever these two implications hold for every i. Then $c \preceq d$ is a necessary, but not a sufficient, condition for c's light component being contained in d's light component. We remark that $c \preceq d$ implies $\mathrm{lcs}(c) \leq \mathrm{lcs}(d)$.

The ordering $\preceq$ tells us something about containment of light components. We can similarly say something about columns whose light components are disjoint: then for every $i \in \{1,\dots,n\}$ we have $c[i] = \mathrm{lcv}(c) \implies d[i] = \mathrm{hcv}(d)$ and $d[i] = \mathrm{lcv}(d) \implies c[i] = \mathrm{hcv}(c)$. We write $c \perp d$ whenever these two implications hold for every i.

Our algorithm is only concerned with building a tree whose edges are labeled with columns of the input genotype matrix; the nodes of the tree are not labeled and the rows of the explaining haplotype matrix B are irrelevant to the algorithm. Since edge labels correspond to mutation events, we call the tree that is constructed by the algorithm a *mutation tree*.

Definition 3.3. Let A be a genotype matrix. A *mutation tree T for A* is an undirected tree whose edges are bijectively labeled by A's columns and which has a distinguished root node r such that:

1. *Ordering condition:* For every path originating at the root with edge labels $c_1, c_2, \dots, c_k$ we have $c_1 \succeq c_2 \succeq \cdots \succeq c_k$.
2. *Compatibility condition:* For every two columns c and d that are incident to a common node v and that do not lie on the path from r to v we have $c \perp d$.
3. *Two-path condition:* For every three columns c, d, and e that are incident to the same node, there is no $i \in \{1,\dots,n\}$ such that $c[i] = d[i] = e[i] = 2$.

Note that a mutation tree has a distinguished root, but unlike for directed perfect phylogenies this root is typically not known in advance and does not need to be labeled by the all-0-haplotype (indeed, nodes are not labeled at all in a mutation tree). The proof of the following characterization can be found in the appendix.

Lemma 3.4. *A genotype matrix A admits a perfect phylogeny with l leafs if, and only, if, there exists a mutation tree for A with l leafs.*

3.2 The Fixed-Parameter Algorithm

Our fixed-parameter algorithm for $\mathrm{IPPH}_{\mathrm{leafs} \leq l}$ works in two stages. The first stage is a preprocessing of the input matrix. After the preprocessing the maximal number of columns with the same light component size is bounded by a function in k and l. The basic idea is that if there are many different columns with the same light component size, they must lie on many different paths and, thus, at some point it is no longer possible to arrange them in a perfect phylogeny with only l leafs. The following lemma states the effect of the preprocessing precisely.

Lemma 3.5. *There is an algorithm* PREPROCESSING *that gets an incomplete $n \times m$ genotype matrix A with at most k many ?-entries per column as input and outputs, in time $O(k4^k m^3 n)$, a genotype matrix A' such that:*

1. *$A \in \mathrm{IPPH}_{\mathrm{leafs} \leq l}$ if, and only if, $A' \in \mathrm{IPPH}_{\mathrm{leafs} \leq l}$.*
2. *There are no duplicate columns in A' and no columns that can be completed to a constant column.*
3. *For every i there are at most $(2k+1)l(3l)^k k!$ columns in A' with light component size i.*

Proof. The algorithm and proof are straightforward generalizations of those given by Gramm et al. [11] for $l = 2$. For a detailed presentation of the algorithm and the proof we refer the interested reader to the technical report version of the present paper [6]. □

The second stage is the main part of the algorithm. By Lemma 3.4, in order to decide whether the preprocessed version A has the property $A \in \text{IPPH}_{\text{leafs}\leq l}$, it suffices to test whether A can be completed in such a way that it admits a mutation tree with at most l leafs.

For the presentation of our algorithm we need some additional terminology: Given a set of columns A or a matrix A, let $A|_{\text{lcs}=i}$, $A|_{\text{lcs}\leq i}$, and $A|_{\text{lcs}>i}$ denote the set of all columns c of A with $\text{lcs}(c) = i$, $\text{lcs}(c) \leq i$, and $\text{lcs}(c) > i$, respectively. A *completion* of a set of columns with ?-entries is obtained by replacing all ?-entries by 0-, 1-, or 2-entries. Note that a completion of a column with light component size i can have a light component size between i and $i+2k$, where k is, as always, the number of ?-entries in the column. The *inner part* of a mutation tree T is the set $\text{inner}(T)$ of edges that are incident to the root of T.

The mutation tree construction algorithm works in iterations $i = 1,2,\dots,n$. In iteration i it processes all completions of the set $A|_{\text{lcs}=i}$. The algorithm keeps track of what it has already found out about completions of $A|_{\text{lcs}<i}$ in previous iterations in what we call *tree records* (I,λ,U). Such a record consists of an *inner part* I, a number $\lambda \in \{0,\dots,l\}$ of leafs, and a set of *unprocessed columns* U. The following definition formalizes the properties that tree records should have:

Definition 3.6. Let A be an incomplete $n \times m$ genotype matrix and let $i \in \{1,\dots,n\}$. A tree record (I,λ,U) is *good for A and i* if there exists a completion S_i of $A|_{\text{lcs}\leq i}$ such that (a) $I = \text{inner}(T_i)$ for some mutation tree T_i for $S_i|_{\text{lcs}\leq i}$, (b) λ is the number of leafs of T_i, and (c) $U = S_i|_{\text{lcs}>i}$.

The job of the algorithm is to compute in each iteration i the set R_i of all good tree records for A and i. Clearly, if R_n is nonempty after the last iteration, there exists a completion for A and a mutation tree with at most l leafs; and otherwise no such completion exists. Figure 2 shows the pseudo-code of the algorithm, Figure 3 shows an example of the algorithm in action.

```
Algorithm SOLVE-IPPH$_{\text{leafs}\leq l}$.
Input: An $n \times m$ genotype matrix $A$ with at most $k$ missing entries per column.
1  $A \leftarrow$ PREPROCESS$(A)$
2  $R_0 \leftarrow \{(\emptyset,0,\emptyset)\}$
3  for increasing light component sizes $i \leftarrow 1,2,\dots,n$ do
4      $R_i \leftarrow \emptyset$
5      for each completion $C$ of $A|_{\text{lcs}=i}$ do
6          for each tree record $(I,\lambda,U) \in R_{i-1}$ do
7              for each mutation tree $T$ for $I \cup C|_{\text{lcs}=i} \cup U|_{\text{lcs}=i}$
                       with $\lambda' - \lambda + |I|$ leafs for some $\lambda' \leq l$
                       where all columns from $I$ are incident to leafs of $T$ do
8                  $R_i \leftarrow R_i \cup \{(\text{inner}(T),\lambda',C|_{\text{lcs}>i} \cup U|_{\text{lcs}>i})\}$
9  if $R_n$ is nonempty then output "$A \in$ IPPH$_{\text{leafs}\leq l}$" else output "$A \notin$ IPPH$_{\text{leafs}\leq l}$"
```

Figure 2: Our decision algorithm for $\text{IPPH}_{\text{leafs}\leq l}$.

The following two lemmas imply that the algorithm is correct and that it is a fixed-parameter algorithm for $\text{IPPH}_{\text{leafs}\leq l}$. Together, they prove Theorem 3.1.

Lemma 3.7. *After each iteration i of algorithm* SOLVE-IPPH$_{\text{leafs}\leq l}$*, the set R_i contains exactly the good tree records for A and i.*

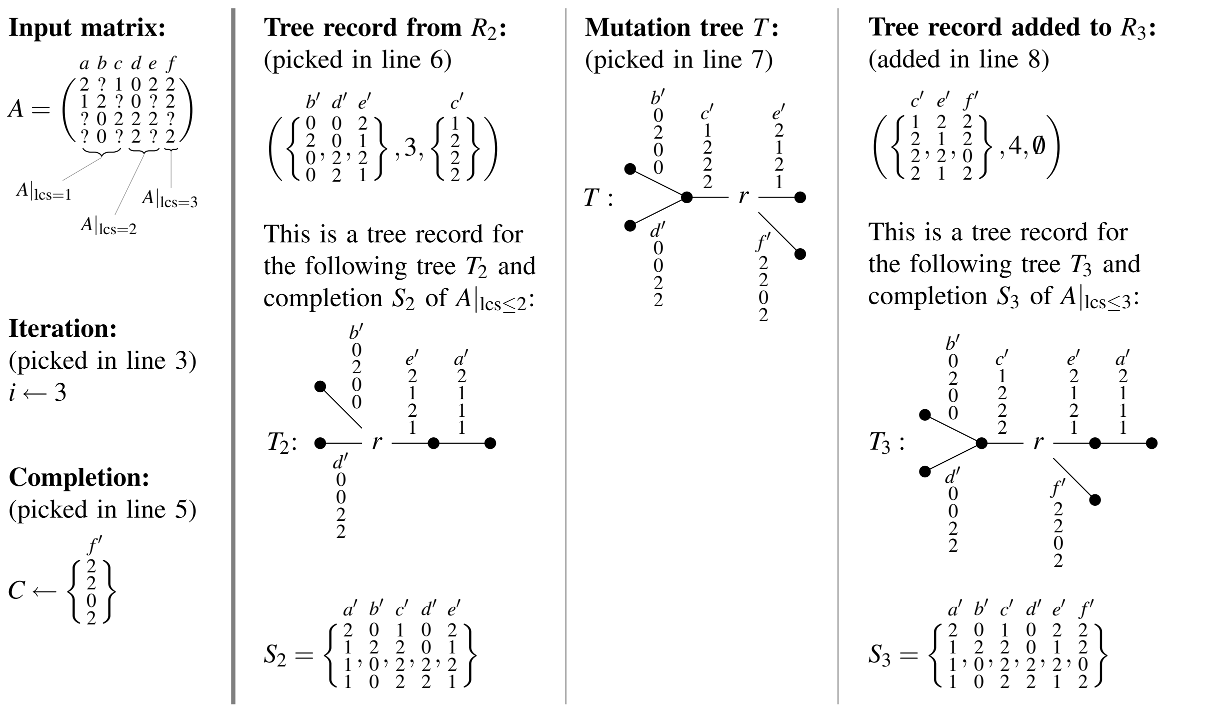

Figure 3: Example of the third iteration of SOLVE-IPPH$_{\text{leafs}\le l}$ for the indicated input matrix A. We depict a set of possible values for the loop variables for which a new tree record is added to R_3.

Sketch of proof. We prove the claim by induction over i. For $i = 0$ the initialization $R_0 = \{(\emptyset, 0, \emptyset)\}$ is correct because the preprocessing ensures that $A|_{\text{lcs}=0} = \emptyset$. For the inductive step from $i-1$ to i, we first argue that the algorithm adds only good tree records to R_i. Suppose the algorithm adds some tree record (I', λ', U') to R_i. Then there exists a completion C of $A|_{\text{lcs}=i}$ and some tree record $(I, \lambda, U) \in R_{i-1}$ such that there is a mutation tree T passing the test from line 7. By the inductive hypothesis, (I, λ, U) is good as witnessed by some T_{i-1} and S_{i-1}. The key observation is that T_{i-1} and T can be combined to a mutation tree T_i for all of $S_i|_{\text{lcs}\le i} = S_{i-1}|_{\text{lcs}\le i-1} \cup C|_{\text{lcs}=i} \cup U|_{\text{lcs}=i}$. This combination is possible, because the algorithm considers only tree combinations where the columns from I label edges incident to leafs.

In order to prove that all good tree records (I', λ', U') are added to R_i, let S_i and T_i witness that the tree record is good. In the tree T_i, columns c with $\text{lcs}(c) = i$ are nearest to the root. By removing them and contracting edges, we get a tree T_{i-1} to which we can apply the induction hypothesis and get a tree record $(I, \lambda, U) \in R_{i-1}$. By considering the inner part of T_i, we get a tree T passing the test of line 7. □

Lemma 3.8. *Algorithm* SOLVE-IPPH$_{\text{leafs}\le l}$ *runs in time* $O(f(k)m^l n^2)$.

Sketch of proof. By Lemma 3.5 the preprocessing takes time $O(k4^k m^3 n)$. In each of the n iterations of the algorithm all completions of $A|_{\text{lcs}=i}$ are considered, of which there are at most $3^{k(2k+1)l(3l)^k k!}$ many (recall that Lemma 3.5 limits the size of $A|_{\text{lcs}=i}$). The algorithm then iterates over all tree records, of which there can be at most $3^{k|A|_{i-2k\le\text{lcs}\le i-1}|}$ many. For the runtime of the inner loop, just note that the size of $I \cup C|_{\text{lcs}=i} \cup U|_{\text{lcs}=i}$ depends only on l and k. □

4 Conclusion

We have shown that restrictions on the topologies of perfect phylogenies can greatly influence the complexity of IPPH and its variants. While restrictions can make the complexity jump from P to NP-complete (as for IDPP), we showed that tree topologies provide the first parameter for which a theoretical analysis is possible of an algorithm that works on arbitrary instances of the IPPH problem. Our new notions of mutation trees and light and heavy component sizes have turned out to be useful in the study of undirected perfect phylogenies; we suggest trying to apply them to other problem versions as well.

The first main open problem is to improve the runtime of the fixed-parameter algorithm since the runtime is the range of $3^{O(k!)}$, which is not feasible even for small values like $k = 5$ that are common in practice. One could argue that, in practice, the algorithm will be much quicker because the bound is only a rather pessimistic worst-case bound, but a faster fixed-parameter algorithm would be a much better alternative. The second main open question is whether IPPH is fixed-parameter tractable with respect to the maximal number of ?-entries per column.

References

[1] V. Bafna, D. Gusfield, G. Lancia, and S. Yooseph. Haplotyping as perfect phylogeny: A direct approach. *Journal of Computational Biology*, 10(3–4):323–340, 2003.

[2] C. J. Benham, S. Kannan, M. Paterson, and T. Warnow. Hen's teeth and whale's feet: Generalized characters and their compatibility. *Journal of Computational Biology*, 2(4):515–525, 1995.

[3] P. Bonizzoni. A linear-time algorithm for the perfect phylogeny haplotype problem. *Algorithmica*, 48(3):267–285, 2007.

[4] A. G. Clark. Inference of haplotypes from PCR-amplified samples of diploid populations. *Journal of Molecular Biology and Evolution*, 7(2):111–122, 1990.

[5] Z. Ding, V. Filkov, and D. Gusfield. A linear-time algorithm for the perfect phylogeny haplotyping (PPH) problem. *Journal of Computational Biology*, 13(2):522–553, 2006.

[6] M. Elberfeld, I. Schnoor, and T. Tantau. Influence of tree topology restrictions on the complexity of haplotyping with missing data. Technical Report SIIM-TR-A-08-05, Universität zu Lübeck, 2008.

[7] M. Elberfeld and T. Tantau. Computational complexity of perfect-phylogeny-related haplotyping problems. In *Proceedings of MFCS 2008*, volume 5162 of *LNCS*, pages 299–310. Springer, 2008.

[8] E. Eskin, E. Halperin, and R. M. Karp. Efficient reconstruction of haplotype structure via perfect phylogeny. *Journal of Bioinformatics and Computational Biology*, 1(1):1–20, 2003.

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

[10] J. Gramm, T. Hartman, T. Nierhoff, R. Sharan, and T. Tantau. On the complexity of SNP block partitioning under the perfect phylogeny model. *Discrete Mathematics*, 2008. To appear, doi:10.1016/j.disc.2008.04.002.

[11] J. Gramm, T. Nierhoff, R. Sharan, and T. Tantau. Haplotyping with missing data via perfect path phylogenies. *Discrete and Applied Mathematics*, 155(6–7):788–805, 2007.

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

[13] D. Gusfield. Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. In *Proceedings of RECOMB 2002*, pages 166–175. ACM Press, 2002.

[14] E. Halperin and R. M. Karp. Perfect phylogeny and haplotype assignment. In *Proceedings of RECOMB 2002*, pages 10–19. ACM Press, 2004.

[15] G. Kimmel and R. Shamir. The incomplete perfect phylogeny haplotype problem. *Journal of Bioinformatics and Computational Biology*, 3(2):359–384, 2005.

[16] Y. Liu and C.-Q. Zhang. A linear solution for haplotype perfect phylogeny problem. In *Proceedings Int. Conf. Adv. in Bioinfor. and Appl.*, pages 173–184. World Scientific, 2005.

[17] I. Pe'er, T. Pupko, R. Shamir, and R. Sharan. Incomplete directed perfect phylogeny. *SIAM Journal on Computing*, 33(3):590–607, 2004.

[18] R. Vijaya Satya and A. Mukherjee. An optimal algorithm for perfect phylogeny haplotyping. *Journal of Computational Biology*, 13(4):897–928, 2006.

[19] R. Vijaya Satya and A. Mukherjee. The undirected incomplete perfect phylogeny problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(4):618–629, 2008.

[20] M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9(1):91–116, 1992.

A Technical Appendix

Proof of Theorem 2.1. We claim that $\phi \in$ MONOTONE NAE3SAT if, and only if, $B \in \text{IDPP}_{\text{leafs} \leq l}$, where B is the matrix described in the sketch of proof.

First, let B' be a completion of B that admits a directed perfect phylogeny T with at most l leafs. Our first claim is that the columns $b_1, \ldots, b_{l-2}$ lie on $l-2$ different branches, each of which contains only this one column b_i. To see this, just note that for any column b_i, no other column of B' can lie on the same root-to-leaf path as b_i since these columns contain the submatrix $\left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. The remaining columns, which are the literal columns, must then lie on at most two further branches T_0 and T_1 (recall that T has at most l leafs). Both branches must be nonempty since B contains the submatrix $\left[\begin{smallmatrix} 1 & 0 & ? \\ ? & 1 & 0 \\ 0 & ? & 1 \end{smallmatrix}\right]$ in literal columns and each completion of this matrix contains the submatrix $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$, which forces columns to lie on different branches. The first n rows assure that all literal columns that correspond to the same variable lie on the same path: they contain 1-entries in the same row. The next $3m$ rows enforce that the literal columns of any given clause do not all lie on the same root-to-leaf path. We can now construct the desired truth assignment τ for the variables of ϕ that witnesses $\phi \in$ MONOTONE NAE3SAT: For a variable v_i, we set $\tau(v_i) = 0$, if the corresponding literal columns lie on T_0 and $\tau(v_i) = 1$, if they lie on T_1.

For the other direction let $\tau : \{v_1, \ldots, v_n\} \to \{0,1\}$ be an assignment of the variables of ϕ such that the literals of a clause do not all share the same truth value. We describe, simultaneously, a completion B' for B and a directed perfect phylogeny T with at most l leafs for B'. First, T contains $l-2$ branches each of which contains exactly one column b_i. The leafs at the ends of these branched are labeled with one of the lower $l-2$ rows of B. Next, there are two further paths T_0 and T_1 in T and these paths contain the completions of all of the remaining columns.

The path T_0 contains all literal columns whose corresponding variables are set to 0 by τ. The ordering of the columns on the path in root-to-leaf order is as follows: First, all literal columns for a clause C_i come earlier than all literal columns for clause C_{i+1}. For the literals inside a single clause C_i, we only need to explain what happens when there are exactly two literals l_i^k and l_i^p inside C_i with $\tau(l_i^k) = \tau(l_i^p) = 0$. In this case, we may assume that $p \equiv k+1 \pmod 3$ holds (otherwise exchange the meanings of k and p). Then l_1^k comes before l_1^p on the path T_0. Each edge on the path T_0 is now labeled with some column l_i^k. We label the node following this column by the row l_i^k; note that this positioning implicitly yields a completion for this row. The branch T_1 is constructed in the same way, only we now consider only literals with $\tau(l_i^k) = 1$. The last node on the path T_0 is labeled by all variable rows v_i with $\tau(v_i) = 0$, similarly for T_1 and variables with $\tau(v_i) = 1$. Again, the positioning implicitly assigns completions to these rows. This completes the construction of the completion B' and of the sought directed perfect phylogeny with at most l leafs. □

Proof of Lemma 3.4. For the first direction let A be a genotype matrix, T_A a perfect phylogeny for A, and B an explaining haplotype matrix. We may assume that each edge of T_A is labeled exactly once, otherwise replace edges with multiple labels by paths of appropriate lengths in which each edge has a unique label. We argue that T_A with an appropriate root and without node labels (and, in some cases, some minor additional changes) is a mutation tree for A.

Let $\{d_1, \ldots, d_s\}$ be the set of all columns with maximal light component size. Then this set forms a connected component in T_A and $\text{lcs}(d_1) = \cdots = \text{lcs}(d_s)$. We distinguish two cases. First, if $\text{lcs}(d_1) < \text{hcs}(d_1)$, there are no two columns d_i and d_j such that d_i belongs to the light component of d_j and vice versa. This assures the existence of a column $c \in \{d_1, \ldots, d_s\}$ that lies in the heavy component of every other column d_i. Second, if $\text{lcs}(d_1) = \text{hcs}(d_1)$, assume there are d_i, d_j, and d_k with maximal light component size such that all three of them are incident to a common node v. Then the component of d_i that contains v also contains the components of d_j and d_k that do not contain v. So $\text{lcs}(d_i) \geq \text{lcs}(d_j) + \text{lcs}(d_k)$, which contradicts $\text{lcs}(d_i) = \text{lcs}(d_j) = \text{lcs}(d_k)$. Thus, the d_i's form a path and no haplotypes label its inner nodes. Hence, the columns on this path can be rearranged so that one of the columns, call it c, lies in the heavy component of every other column. In both cases let r be the node incident to c in its heavy component. Then r lies in the heavy component of each column d_i. For columns d with $\text{lcs}(d) < \text{lcs}(c)$ the light component of d contains neither the light component nor the heavy component of c. Thus, regardless of whether d lies in the light component or heavy component of c, the node r lies in the heavy component of d.

We show that T_A (with the edges d_i possibly rearranged as described earlier) with root r is a mutation tree when we disregard the labeling of its nodes. To show that the ordering condition is satisfied, consider two columns c_1 and c_2 where c_1 lies on the path from r to c_2. Since r lies in the heavy component of each column, the heavy component of c_1 is a subtree of the heavy component of c_2 and the light component of c_2 is a subtree of the light component of c_1, which implies $c_1 \succeq c_2$. For the compatibility condition consider two edges c and d that are incident to a node v and that do not lie on a path from r to v. Then the light component sides of c and d are disjoint and, thus, $c \perp d$. Finally, for the two-path condition recall that by Gusfield's characterization [13] the path between the explaining haplotypes for a genotype g contains exactly the columns where the genotype has 2-entries. Thus, these columns cannot be distributed among more than two branches.

To prove the other direction, let T be a mutation tree with root r. We show that the following node labeling makes T a perfect phylogeny for A: For each column c and node v, we set the label of v in column c to $\text{lcv}(c)$ whenever c lies on the path between r and v and to $\text{hcv}(c)$, otherwise. It suffices to show that for every genotype $g \in A$ there are two labels explaining it. Let g be a genotype from A and let $A_g^{\text{lcv}} := \{c \mid c \text{ is column of } A,\ g[c] = \text{lcv}(c)\}$ and $A_g^2 := \{c \mid c \text{ is column of } A,\ g[c] = 2\}$. In the following we show that (a) there is a node v that is connected to r via a path labeled exactly by the columns in A_g^{lcv}, (b) there are two nodes w and w' connected by a path that goes through v and is labeled exactly by the columns in A_g^2, and (c) the labels of w and w' explain g.

Since T satisfies the ordering condition, the columns from A_g^{lcv} form a connected component in T that contains r. The compatibility condition ensures that this component is a single path from r to some node v. Thus, (a) is true. Let T_v be the subtree of T rooted at v. Assume there is a $c \in A_g^2$ that labels an edge not belonging to T_v. Let v' be the least common ancestor of c and v. Let d_1 and d_2 be the columns connected to v' such that d_1 starts the path from v' to v and d_2 starts the path from v' to c. Then, due to the compatibility condition, it holds $d_1 \perp d_2$ and therefore $g[d_2] = \text{hcv}(d_2)$. The ordering condition gives $d_2 \succeq c$, which implies $g[c] = \text{hcv}(c)$, a contradiction to the choice of c. So, all columns in A_g^2 are in T_v. With the property that all columns from T_v have a heavy component value or a 2-entry in g, the columns in A_g^2 form a connected component in T_v that contains v. The two-path condition implies that this component is a path, so (b) holds. Finally, to prove (c) let w and w' be the two vertices that are connected by this A_g^2-path. The path from r to v contains exactly the columns with light component value in g, so the labels both have a light component value in these columns. The columns from $c \in A_g^2$ are distributed among the paths from v to w and v to w'. Therefore one label has $\text{lcv}(c)$ and the other $\text{hcv}(c)$ in column c. All columns not belonging to A_g^{lcv} or A_g^2 do not appear in either path, thus the labels contain heavy component values in these columns. Hence, the labels of w and w' explain g. □

Proof of Lemma 3.7. We prove the claim by induction over i. For $i = 0$ the initialization $R_0 = \{(\emptyset, 0, \emptyset)\}$ is correct because the preprocessing ensures that $A|_{\text{lcs}=0} = \emptyset$, see the second property of Lemma 3.5.

For the inductive step from $i-1$ to i, we first argue that the algorithm adds only good tree records to R_i. Suppose the algorithm adds some tree record (I', λ', U') to R_i. Then there exists a completion C of $A|_{\text{lcs}=i}$ and some tree record $(I, \lambda, U) \in R_{i-1}$, such that there is a mutation tree T passing the test from line 7. By the inductive hypothesis, (I, λ, U) is good as witnessed by some T_{i-1} and S_{i-1}. Our objective is to combine T_{i-1} and T into a a mutation tree T_i for all of $S_i|_{\text{lcs}\le i} = S_{i-1}|_{\text{lcs}\le i-1} \cup C|_{\text{lcs}=i} \cup U|_{\text{lcs}=i}$. First, we split T_{i-1} at the root, which leads to $|\text{inner}(T_{i-1})|$ many subtrees. We then identify the top edges of these subtrees with the edges that are labeled by columns from $\text{inner}(T_{i-1})$ in T. Remember that these columns label edges that are incident to leafs. Figure 4 shows the construction of T_i from T and T_{i-1}.

The tree T_i is a mutation tree since the local properties at incident edges and edges around nodes are satisfied by construction. Since T has $\lambda' - \lambda + |I| \leq l - \lambda + |I|$ leafs and T_{i-1} has λ leafs, T_i has $\lambda' \leq l$ leafs.

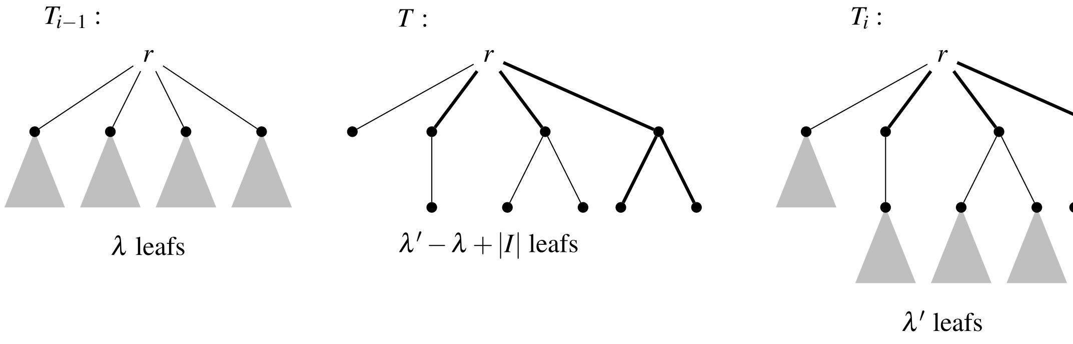

Figure 4: Example of the construction of T_i from T and T_{i-1}. In line 7, the algorithm combines edges that are labeled with columns inner(T_{i-1}) (thin lines) and edges that are labeled with columns in $C|_{\text{lcs}\leq i} \cup U|_{\text{lcs}\leq i}$ (thick lines) to construct a mutation tree T for them, such that the columns from inner(T_{i-1}) label edges that are incident to leafs. Since the ordering, compatibility and two-path conditions are satisfied locally in trees T_{i-1} and T, there exists the depicted combined mutation tree T_i for $S_{i-1}|_{\text{lcs}\leq i-1} \cup C|_{\text{lcs}=i} \cup U|_{\text{lcs}=i}$. We also know that T_i has $\lambda' \leq l$ leafs since T_{i-1} has λ leafs and T has $\lambda' - \lambda + |I| \leq l - \lambda + |I|$ leafs.

It remains to argue that all good tree records (I', λ', U') are added to R_i. Let S_i and T_i witness that the tree record is good for A and i. Partition S_i into two sets S_{i-1} and C, such that S_{i-1} is a completion of $A|_{\text{lcs}\leq i-1}$ and C is a completion of $A|_{\text{lcs}=i}$. Let T_{i-1} be obtained from T_i by contracting all edges labeled by columns with light component size i.

We claim that T_{i-1} is a mutation tree for the columns from $S_{i-1}|_{\text{lcs}\leq i-1}$. To prove this, we show that for every mutation tree T' and every edge $\{v,w\}$ labeled d, where v lies on the path from the root to w, if we contract the edge $\{v,w\}$, the resulting tree T'' still satisfies the ordering, compatibility, and two-path conditions:

1. T'' clearly still satisfies the ordering condition.
2. For the compatibility condition, let u be a node of T''. If u is neither v nor w, the compatibility condition is still true at u. Otherwise, $u = v = w$. Suppose for sake of contradiction, that u is incident to two columns c and c' that are not on the path to the root and where $c \perp c'$ does not hold. Without loss of generality, we may assume $c[g] = \text{lcv}(c)$ and $c'[g] \neq \text{hcv}(c')$ for a genotype g. Since the compatibility condition holds for T', we know that neither v nor w is incident to both c and c' in T'. First, consider the case that v is incident to c and w is incident to c'. Since $c[g] = \text{lcv}(c)$, we have $d[g] = \text{hcv}(d)$. Together with the ordering condition it follows that c' must have a heavy component value in genotype g, a contradiction. Second, assume that v is incident to c' and w is incident to c. From the ordering condition we can deduce that d has a light component value in genotype g. Thus, c' and d at node v in T' contradict the compatibility condition.
3. For the two-path condition, let u be a node of T''. Again, if u is neither v nor w, the two-path condition is still satisfied at u, so assume $u = v = w$. For the sake of contradiction, assume that u is incident to three columns c, c' and c'' with $c[g] = c'[g] = c''[g] = 2$ for a genotype g. Since the two-path condition holds in T', the nodes v and also w are incident to at least one of these columns. First, we assume that v is incident to c and w is incident to c' and c''. The ordering condition implies that d has either a 2-entry or a light component value in g. If d has a 2-entry in g, the two-path condition is not satisfied at node w. If d has a light component value in g, we

distinguish the cases that c labels the path between v and the root in T' and that c does not label this path. In the first case, the ordering condition is not satisfied for c and d, and in the second case the compatibility condition is not satisfied at node v. If we assume that two columns c and c' are incident to v and one column c'' is incident to w, we can analogously deduce a contradiction. Thus, the two-path condition holds for every node of T''.

We have now proved that T_{i-1} is a mutation tree for $S_{i-1}|_{\mathrm{lcs}\leq i-1}$. By the inductive assumption, there must be good tree record $(\mathrm{inner}(T_{i-1}), \lambda, S_{i-1}|_{\mathrm{lcs}>i-1}) \in R_{i-1}$ for T_{i-1} and S_{i-1}. Since light component sizes only increase as we near the root, in T_i there can be no column c with $\mathrm{lcs}(c) \leq i-1$ that labels an edge between the root and a column c' with $\mathrm{lcs}(c') = i$. With this property in mind, we combine $\mathrm{inner}(T_{i-1})$ and the columns with light component size exactly i from S_i to a tree T with $\mathrm{inner}(T) = \mathrm{inner}(T_i)$ that has $\lambda' - \lambda + |\mathrm{inner}(T_{i-1})| \leq l - \lambda + |\mathrm{inner}(T_{i-1})|$ leafs. This construction is the converse of the construction in Figure 4. Then T will pass the test of line 7 and $(\mathrm{inner}(T_i), \lambda', U')$ will, indeed, be inserted into R_i. □

Proof of Lemma 3.8. By Lemma 3.5 the preprocessing takes time $O(k4^k m^3 n)$. The number of completions of $A|_{\mathrm{lcs}=i}$ considered in round i is bounded by $3^{k(2k+1)l(3l)^k k!}$ since Lemma 3.5 limits the size of $A|_{\mathrm{lcs}=i}$ by $(2k+1)l(3l)^k k!$. We now argue that the number of good tree records over which the algorithm iterates can be at most $3^{k|A|_{i-2k\leq\mathrm{lcs}\leq i-1}|}$. A tree record consists of a set I of at most l complete columns for which there are at most $m^l 3^{kl}$ possibilities, a value $\lambda \leq l$, and a set U of unprocessed completions. The set U contains complete columns from the preceding iterations that have light component size at least i. When we complete a column, its light component size can only increase by at most $2k$ and, thus, a complete column corresponds to an incomplete column from the last $2k$ rounds. Thus there are at most $3^{k|A|_{i-2k\leq\mathrm{lcs}\leq i-1}|}$ possibilities, which is also bounded by a function in k and l. Finally, for the runtime of the inner loop, just note that the size of $I \cup C|_{\mathrm{lcs}=i} \cup U|_{\mathrm{lcs}=i}$ depends only on l and k. □