FPT Algorithms for Binary Near-Perfect Phylogenetic Trees [1]

Srinath Sridhar[a], Kedar Dhamdhere[b], Guy E. Blelloch[c], Eran Halperin[d], R. Ravi[e] and Russell Schwartz[f]

September 29, 2005
CMU-CS-05-181

[a]Computer Science Department, Carnegie Mellon University. Email: srinath@cs.cmu.edu

[b]Google Inc, Mountain View, CA. Email: kedar.dhamdhere@gmail.com

[c]Computer Science Department, Carnegie Mellon University. Email: blelloch@cs.cmu.edu

[d]ICSI, University of California, Berkeley. Email: heran@icsi.berkeley.edu

[e]Tepper School of Business, Carnegie Mellon University. Email: ravi@cmu.edu

[f]Department of Biological Sciences, Carnegie Mellon University. Email: russells@andrew.cmu.edu

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]Supported in part by NSF grant CCF-0430751 and ITR grant CCR-0122581 (The ALADDIN project).

Keywords: Phylogenetic trees, Parsimony, Near-perfect phylogeny

Abstract

We consider the problem of reconstructing near-perfect phylogenetic trees using binary character states (referred to as BNPP). A perfect phylogeny assumes that every character mutates at most once in the evolutionary tree, yielding an algorithm for binary character states that is computationally efficient but not robust to imperfections in real data. A near-perfect phylogeny relaxes the perfect phylogeny assumption by allowing at most a constant number of additional mutations. In this paper, we present a simple lower bound for the size of an optimal phylogeny, develop two algorithms for constructing optimal phylogenies and show experimental results for one of the variants. The first algorithm is intuitive and reconstructs an optimal near-perfect phylogenetic tree in time $(q + \kappa)^{O(q)}nm + O(nm^2)$ where κ is the number of characters that share four gametes with some other character. A second, more involved algorithm shows the problem to be fixed parameter tractable in q by solving it in time $q^{O(q)}nm + O(nm^2)$ where n is the number of taxa and m is the number of characters. This is a significant improvement over the previous best result of $nm^{O(q)}2^{O(q^2s^2)}$, where s is the number of states per character (2 for binary). We implement the first algorithm and show that it finds the optimal solution quickly for a selection of population datasets including mitochondrial and Y chromosome samples from humans and other primates. Our results describe the first practical phylogenetic tree reconstruction algorithm that finds guaranteed optimal solutions while being easily implemented and computationally feasible for data sets of biologically meaningful size and complexity.

1 Introduction

One of the core areas of computational biology is phylogenetics, the reconstruction of evolutionary trees [13]. This problem is often phrased in terms of a parsimony objective, in which one seeks the simplest possible tree to explain a set of observed organisms. Parsimony is a particularly appropriate metric for trees representing short time scales, which makes it a good choice for inferring evolutionary relationships among individuals within a single species or a few closely related species. The intraspecific phylogeny problem has become especially important in studies of human genetics now that large-scale genotyping and the availability of complete human genome sequences [23, 16] have made it possible to identify millions of single nucleotide polymorphisms (SNPs) [21, 25], sites at which a single DNA base takes on two common variants. If we wish to be able to infer the most plausible evolutionary histories on the rapidly growing human variation datasets, we can expect a growing need for algorithms for the intraspecies phylogeny problem capable of dealing with large data sets, especially large SNP data sets.

Minimizing the length of a phylogeny is the problem of finding the most parsimonious tree, a well known NP-complete problem [10]. Researchers have thus focused on either sophisticated heuristics (for e.g [11], [4]) or solving optimally for special cases (fixed parameter variants, for e.g, [1], [17]). Heuristics such as hill climbing searches are particularly popular among programs that optimize for the parsimony objective, but can result in solutions far from optimal. A popular alternative is to use branch-and-bound algorithms, which guarantee an optimal solution on termination but are computationally intensive and therefore cannot scale to more than a few taxa. Our algorithms can guarantee optimality on termination and can scale to sizes much larger than branch-and-bound. Previous attempts at such solutions for the general parsimony problem have only produced theoretical results, yielding algorithms too complicated for practical implementation.

Fernandez-Baca and Lagergren recently considered the problem of reconstructing optimal near-perfect phylogenies [9], which assume that the size of the optimal phylogeny is at most q larger than that of a perfect phylogeny for the same input size. They developed an algorithm to find the most parsimonious tree in time $nm^{O(q)}2^{O(q^2s^2)}$, where s is the number of states per character, n is the number of taxa and m is the number of characters. This bound may be impractical for sizes of m to be expected from SNP data, even for moderate q. Given the importance of SNP data, it would therefore be valuable to develop methods able to handle large m for the special case of $s = 2$, a problem we call Binary Near Perfect Phylogenetic tree reconstruction (BNPP).

Here we present theoretical and practical results on the optimal solution of the BNPP problem. We first prove a simple lower bound on the size of the optimal phylogenetic tree. We then completely describe and analyze an intuitive algorithm for the BNPP problem that has running time $(q + \kappa)^{O(q)}nm + O(nm^2)$, where κ is the number of characters that are involved in the *four gamete* condition. This algorithm is fixed parameter tractable when q and κ are constants. Since κ can be at most the number of characters in the input, our algorithm can never have a worse running time than the prior algorithm. In practice we find that κ is about the same as q and therefore, the running time of our algorithm is not exponential in the size of the input. Fernandez-Baca and Lagergren [9] stated that the most important open problem in the area is the proof of W[1]-hardness or showing the existence of an FPT algorithm for near-perfect phylogeny. We show, using a somewhat more complicated algorithm that BNPP can be solved in time $q^{O(q)}nm + O(nm^2)$, proving for the first time that BNPP is in FPT.

We demonstrate our initial intuitive algorithm on a selection of real data sets, including examples of mitochondrial and Y-chromosome data. We find that the algorithm generally substantially outperforms its worst-case bounds. The results show the method to be fast for real data sets of interest and in at least one case to yield superior results to a commonly used heuristic method for these problems.

We first develop formal definitions and fundamental theorems in the following section. We provide the complete pseudo-code for the first algorithm in Figures 2 and 3. In Section 3.1 we prove the correctness of the algorithm. In section 4 we provide a highlevel description and analysis of the faster algorithm . Finally, in section 6, we conclude with experimental results and discussion.

2 Related Work and Problem Specification

A phylogenetic tree T is called perfect if for all states s and characters c, all taxa having state s at character c lie in a connected component of the phylogeny. The problem of reconstructing perfect phylogenies was proved to be NP-hard independently by Bodlaender et al. [2] and Steel [20]. Some special cases of the phylogeny model can be solved optimally and efficiently. Gusfield considered an important special case of the perfect phylogeny problem when the number of states is bounded by 2, called the binary perfect phylogeny problem (BPP), showing that such phylogenies can be reconstructed in linear time [12]. Day and Sankoff [6], however, showed that finding the maximum subset of characters containing a binary perfect phylogeny is NP-complete. Bodlaender et al. [3] proved a number of crucial negative results, among them that finding the perfect phylogeny when the number of characters is fixed is $W[t]$-hard for all t. Since $FPT \subseteq W[1]$, this shows in particular that the problem is not fixed parameter tractable (unless the complexity classes collapse).

In defining formal models for parsimony-based phylogeny construction, we borrow definitions and notations from Fernandez-Baca and Lagergren [9]. The input used for the phylogeny reconstruction problem is a matrix I where rows R represent *taxa* and are strings over states. The columns $C = \{1, \cdots, m\}$ are referred to as *characters*. The set of states corresponding to any character c is denoted by A_c. Thus, every taxon $r \in A_1 \times \cdots \times A_m$. In a *phylogenetic tree*, or *phylogeny*, each vertex v corresponds to a taxon and has an associated label $l(v) \in A_1 \times \cdots \times A_m$.

Definition 1 *A* phylogeny *for a set of n taxa R is a tree $T(V, E)$ with the following properties:*

1. *if a taxon $r \in R$ then $r \in l(V(T))$*
2. *for all $(u, v) \in E(T)$, $H(l(u), l(v)) = 1$ where H is the Hamming distance*

Definition 2 *The* length of a phylogeny *T, length(T) $= |E(T)|$*

Definition 3 *The* penalty *of phylogeny T, penalty(T) = length$(T) - \sum_{c \in C}(|A_c| - 1)$*

Definition 4 *A vertex v of phylogeny T is* terminal *if $l(v) \in R$ and* Steiner *otherwise.*

The BNPP problem: Given an integer q and an $n \times m$ input matrix I, where each row(taxon) $r \in \{0, 1\}^m$, find a phylogeny T such that *penalty(T)* is minimized or declare `NIL` if all phylogenies have penalty larger than q. The problem is equivalent to finding the minimum Steiner tree on a hyper-cube if the optimal tree is at most q larger than the number of dimensions or declaring `NIL` otherwise. The problem is fundamental and therefore expected to have diverse applications besides phylogenies.

3 Simple Algorithm

Our algorithm breaks the BNPP problem into several instances of perfect phylogeny problem. The perfect phylogeny solutions are then linked to form the near perfect phylogeny. We start with some basic notations and definitions, and follow it up with a high level description of the algorithm along with complete pseudo-code.

Definition 5 *We define the following* notations*:*

- $l(v) \in \{0,1\}^m$*: the taxa that vertex* v *of the phylogeny represents*
- $r[i] \in \{0,1\}$*: the state in character* i *of taxa* r
- $\mu(e) : E(T) \rightarrow C$*: the character corresponding to edge* $e = (u,v)$ *with the property* $l(u)[\mu(e)] \neq l(v)[\mu(e)]$
- $mul(c', T)$*: for any character* c' *and phylogeny* T *is the number of times* c' *mutates in* T*, i.e* $|\{e \in T \mid \mu(e) = c'\}|$

We say that an edge e *mutates* character c' if $\mu(e) = c'$. We will use the following well known definition and lemma on phylogenies:

Definition 6 *The* set of gametes $G_{i,j}$ *for characters* i,j *is defined as:* $G_{i,j} = \{(k,l) | \exists r \in R, r[i] = k, r[j] = l\}$*. Two characters* $i, j \in C$ *contain (all) four gametes when* $|G_{i,j}| = 4$.

Lemma 3.1 *[12] The most parsimonious phylogeny for input* I *is not perfect if and only if* I *contains the four-gamete property.*

Input Assumptions

If no pair of characters in input I contains the four-gamete property, we can use Gusfield's elegant algorithm [12] to reconstruct a perfect phylogeny. We assume that the all zeros taxa is present in the input. If not, using our freedom of labeling, we convert the data so that it contains the same information with the all zeros taxa (see section 2.2 of Eskin et al [7] for details). We now remove any character that contains only one state. Such characters do not mutate in the whole phylogeny and are therefore useless in any phylogeny reconstruction. We also assume that for every pair of characters c', c'', $|G_{c',c''}| \geq 3$. This does not change the general problem, since if $|G_{c',c''}| = 2$, then the characters c', c'' contain identical information and such pairs are usually referred to as a non-informative pair of characters. We show at the end of Section 3.1 how to pre-process the input such that this condition holds for all characters.

Conflict Graph G

The *conflict graph* G, introduced by Gusfield et al. [14], is used to represent the imperfectness of the input in a graph. Each vertex $v \in V(G)$ of the graph represents a character $c(v) \in C$. An edge (u,v) is added if and only if all the four gametes are present in $c(u)$ and $c(v)$. Let VC be any minimum vertex cover of G. We now show a simple but useful lower bound on the perfectness of the optimal phylogeny:

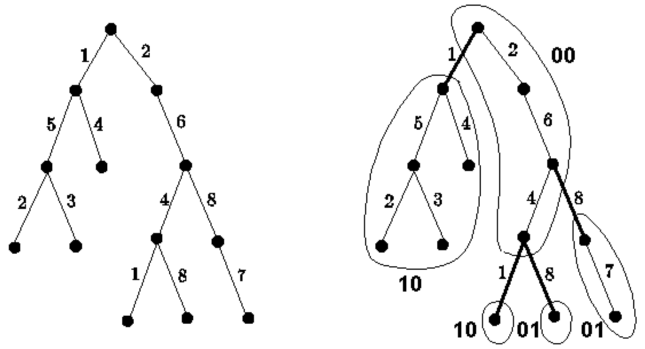


Figure 1: An unrooted phylogeny T and skeleton $s(T, C')$ with $C' = \{1, 8\}$. Edges are labeled with characters that mutate μ and super nodes with tags t. For conciseness, we show the tag only on the characters that mutate in $s(T, C')$; they contain 0 in all other characters. Note that the tags are *not* unique.

Lemma 3.2 *For any phylogeny T, penalty$(T) \geq |VC|$.*

Proof: Follows immediately from the fact that for any pair of characters (i, j) s.t. $|G_{i,j}| = 4$, the phylogeny T must contain two edges e, e' s.t. either $\mu(e) = \mu(e') = i$ or $\mu(e) = \mu(e') = j$. $\square$

We now introduce definitions that will be used to decompose a phylogeny:

Definition 7 *For any phylogeny T and set of characters $C' \subseteq C$:*

- *a* super node *is a maximal connected subtree T' of T s.t. for all edges $e \in T', \mu(e) \notin C'$*
- *the* skeleton *of T, $s(T, C')$, is the tree that results when all super nodes are contracted to a vertex. The vertex set of $s(T, C')$ is the set of super nodes. For all edges $e \in s(T, C')$, $\mu(e) \in C'$.*

Definition 8 *A* tag *$t(u) \in \{0,1\}^m$ of super node u in skeleton $s(T, C')$ has the property that $t(u)[c'] = l(v)[c']$ for all $c' \in C'$ and vertices $v \in u$; $t(u)[i] = 0$ for all $i \notin C'$.*

Throughout this paper, wlog we will deal with phylogenies and skeletons that are rooted at the all zeros taxa and tag respectively. Note that in such skeletons, tag $t(u)[i] = 1$ iff character i mutates an odd number of times in the path from the root to u. Figure 1 shows an example of a skeleton of a phylogeny. We will use the term *sub-phylogeny* to refer to a subtree of a phylogeny.

Overview of the Algorithm

Throughout the analysis, we fix an optimal phylogeny T_{opt} and show that our algorithm finds it. We assume that both T_{opt} and its skeleton is rooted at the all zeros label and tag respectively. The algorithm starts by determining the set of characters $c(V_{nis})$ that corresponds to the non-isolated vertices of the conflict graph. We will later show that $s(T_{opt}, c(V_{nis}))$ contains a perfect sub-phylogeny in every supernode. The algorithm then guesses the skeleton S so that it is the same as $s(T_{opt}, c(V_{nis}))$. The algorithm can now determine the super node where every taxon resides using the tags of the super nodes. We will later show that the tags of $s(T_{opt}, c(V_{nis}))$ are unique. These steps are performed by function `buildNPP` of the pseudo-code in Figure 2. We have

```
function buildNPP ( matrix $M$, integer $q$ )
    1. Let $G(V, E)$ be the conflict graph of $M$
    2. Let $V_{nis} \subseteq V$ be the set of non-isolated vertices
    3. Guess skeleton tree $S(V_s, E_s)$ s.t. $\mu(e) \in c(V_{nis})$ for all $e \in E_s$ and $penalty(S) \leq q$
    4. If there exists super nodes $u, v \in V_s$ s.t. tags $t(u) = t(v)$, then return NIL
    5. Define $\lambda : R \mapsto V_s$ s.t. $\lambda(r) = u$ iff for all $i \in c(V_{nis})$, $r[i] = t(u)[i]$
    6. $T_f$ := linkTrees ($S(V_s, E_s)$)
```

Figure 2: Pseudo-code to find the skeleton.

therefore reduced the problem of reconstructing an imperfect phylogeny to constructing several perfect phylogenies and linking them together.

For each super node, the algorithm now constructs a unique perfect phylogeny on the set of taxa that resides in it. To determine the root of this sub-phylogeny it finds a small set P_i of candidate vertices based on the states they share with the rest of the phylogeny. The algorithm then links the rooted perfect sub-phylogenies using the skeleton edges in function `linkTrees` of the pseudo-code in Figure 3. We prove the correctness of the pseudo-code in the following section.

3.1 Algorithm correctness

After some basic definitions, we will first prove the correctness of function `buildNPP`. We do this by first showing that the tags of the supernodes in the skeleton are unique in Corollary 3.7. This enables us to define function λ at Step 5 of function `buildNPP`. Throughout the analysis of correctness we will assume that every 'guess' in the pseudo-code is 'correct'. The correctness of a guess is defined as follows (these definitions may assume that previous guesses have all been correct):

1. Step 3 of function `buildNPP`: In T_{opt}, contracting edges $e, \mu(e) \notin c(V_{nis})$ results in tree S.
2. Step 2f of function `linkTrees`: In T_{opt}, w is the root of the minimal sub-phylogeny containing all terminal vertices of S_i.

At the heart of many FPT algorithms is a kernalization proof that shows that the search space of optimal solutions (kernel) is significantly smaller than the space of all solutions. The following definitions and transform are useful in establishing that phylogenies outside a small kernel are non-optimal. These are pertinent only for the proofs of Lemmas 3.4 and 3.6 and are not required for the understanding of the rest of the paper.

Definition 9 *A vertex v in a phylogeny T is* bad *w.r.t (i, j) if the states at characters i, j of $l(v)$ is not present in the input, i.e $(l(v)[i], l(v)[j]) \notin G_{i,j}$. Note that by definition bad vertices are Steiner vertices.*

```
function linkTrees ( skeleton $S(V_s, E_s)$ )
  1. Let $R' := R$
  2. For any leaf super-node $S_i \in V_s$ do
     (a) Let $R_i := \{r \in R' | \lambda(r) = S_i\}$
     (b) Let $T_i :=$ perfect-phylogeny ( $R_i$ )
     (c) Let $P_i :=$ vertex set of $T_i$
     (d) for every character $c$ s.t. there exists $e \in T_i, \mu(e) = c$ do
         i. if for all taxa $r_1, r_2, (\lambda(r_i) \in S \setminus S_i) \implies (r_1[c] = r_2[c])$,
            then remove all vertices $v$ from $P_i$ if $l(v)[c] \neq r[c]$ for any $r, \lambda(r) \in S \setminus S_i$
     (e) Let $c' := \mu(S_i, \text{parent}(S_i))$
     (f) Guess vertex $w$ from $P_i$, and let $w'$ be a vertex with $l(w')[c'] \neq l(w)[c']$ and
         $l(w')[i] = l(w)[i]$ for all $i \neq c'$
     (g) Add $w'$ to $R'$, let $\lambda(w') = \text{parent}(S_i)$ and remove leaf $S_i$ from $S$
  3. repeat step 1 until $S$ is empty
```

Figure 3: Pseudo-code to construct and link perfect phylogenies to form a near-perfect phylogeny.

Definition 10 *The* neighborhood $N_{i,j}(v)$ *of a vertex* $v \in T$ *w.r.t a pair of characters* i, j *is*

$$N_{i,j}(v) = \{u \mid \textit{ path } v \rightarrow u \textit{ does not contain edge } e \textit{ st } \mu(e) = i \textit{ or } j\}$$

A neighborhood is bad or Steiner if every vertex in the neighborhood is bad or Steiner respectively.

Definition 11 *The* boundary *of* $N_{i,j}(v)$ *for a phylogeny* $T(V', E')$ *is the set of edges that connect vertices in* $N_{i,j}(v)$ *to the rest of the tree, i.e* $\{(v_1, v_2) \in E' \mid v_1 \in N_{i,j}(v), v_2 \notin N_{i,j}\}$.

Transform τ: We now introduce transform τ that is used to restructure a phylogenetic tree. Let N denote a Steiner (possibly bad) neighborhood w.r.t (i, j). Transform $\tau_{i,j}(N)$ deletes the mutations of character i from the boundary and adds mutation of i after every mutation of j in the boundary. Since the neighborhood is Steiner, if the tree T was previously a phylogeny, it remains so after an application of transform τ. Since the length of the phylogeny after the transform changes iff the number of mutations of i and j at the boundary are different, we can prove the following simple claim:

Claim 3.3 *In any optimal phylogeny* T_{opt}*, every Steiner neighborhood* $N_{i,j}$ *has an equal number of mutations of* i *and* j *at the boundary.*

The following two lemmas prove that the structure of T_{opt} is restrictive. Intuitively, if a character x mutates twice in T_{opt}, then there has to be a mutation of character x' that lies between the two mutations of x s.t. x and x' share all four gametes in the input. The following lemma is a slightly stronger statement:

Lemma 3.4 *A phylogeny* T *with the following properties cannot be optimal:*

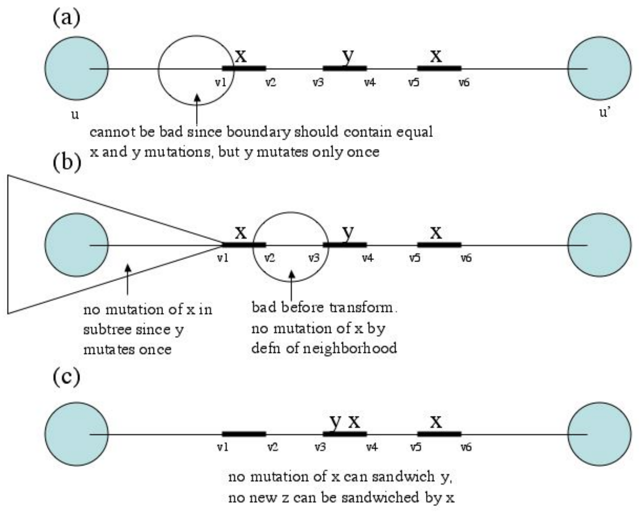

(a)
x
y
x
v1
v2
v3
v4
v5
v6
u
u'
cannot be bad since boundary should contain equal x and y mutations, but y mutates only once
(b)
x
y
x
v1
v2
v3
v4
v5
v6
no mutation of x in subtree since y mutates once
bad before transform. no mutation of x by defn of neighborhood
(c)
y x
x
v1
v2
v3
v4
v5
v6
no mutation of x can sandwich y, no new z can be sandwiched by x

Figure 4: Proof of Lemma 3.6: Figure (a) deals with the case when $N_{x,y}(v_1)$ is bad, Figure (b) is the phylogeny before applying $\tau_{x,y}(v_2)$ and Figure (c) is the phylogeny after transform $\tau_{x,y}(v_2)$.

- *there exists character x and edges $e_1 = (v_1, v_2)$ and $e_2 = (v_5, v_6)$ s.t. $\mu(e_1) = \mu(e_2) = x$*
- *for all characters x' s.t. $|\{e \in \text{path } v_2 \to v_5 \mid \mu(e) = x'\}|$ is odd, at least one of the four induced neighborhoods $N_{x,x'}(v_1)$, $N_{x,x'}(v_2)$, $N_{x,x'}(v_5)$, $N_{x,x'}(v_6)$ is Steiner*

Proof: For the sake of contradiction, assume not. Among the optimal phylogenies that violate the above property, select a phylogeny in which $H(l(v_1), l(v_6))$ is minimized, where H is the Hamming distance. Since optimality prevents $l(v_1)$ and $l(v_6)$ to be identical, we should have some character y that mutates an odd number of times in the path $v_2 \to v_5$, i.e $|\{e \in \text{ path } v_2 \to v_5 \mid \mu(e) = y\}|$ is odd. First consider the case when $N_{x,y}(v_2)$ is Steiner ($N_{x,y}(v_5)$ is symmetric). We can now apply transform $\tau_{y,x}(N_{x,y}(v_2))$, which results in an optimal phylogeny with $H(l(v_1), l(v_6))$ reduced by one, a contradiction. This is because a mutation of y is removed from the path $v_2 \to v_5$. Now consider the case when neighborhood $N_{x,y}(v_1)$ is Steiner ($N_{x,y}(v_6)$ is symmetric). Applying transform $\tau_{y,x}(N_{x,y}(v_1))$ results in adding a mutation of y into the path $v_2 \to v_5$ and once again yields an optimal phylogeny where the Hamming distance is reduced by one, contradiction. □

The following simple corollary to the above lemma states that if a character corresponding to an isolated vertex of the conflict graph mutates more than once, then the phylogeny is non-optimal:

Corollary 3.5 *For all isolated vertices $v \notin V_{nis}$, in any optimal phylogeny T_{opt}, $mul(c(v), T_{opt}) = 1$.*

Lemma 3.6 *The tags of the super nodes in skeleton $s(T_{opt}, c(V_{nis}))$ are unique.*

Proof: We start the proof with a definition that characterizes the structure of the phylogeny:

Definition 12 *The* sandwich strength *of an optimal phylogeny $T_{opt}(V', E')$ is the number of distinct ordered pairs of characters $(\mu(x), \mu(y))$, s.t. x and y are non-isolated and isolated vertices of* *the conflict graph respectively, and there exists edge $e, \mu(e) = y$ in the path connecting two mutations of x: $|\{(x, y) \mid x \in V_{ns}, y \notin V_{ns}, \exists e_1, e_2, e_3 \in E', \mu(e_1) = \mu(e_2) = x, \mu(e_3) = y$ and $e_3 \in$ path $e_1 \rightarrow e_3\}$.*

For the sake of contradiction assume that there exist optimal phylogenies with super nodes $u, u' \in V_s$ s.t. tags $\mathrm{t}(u) = \mathrm{t}(u')$. Among the optimal phylogenies select T_{opt} based on the following criteria (in order):

- the minimum distance between any two super nodes of identical tags is minimized
- the sandwich strength of T_{opt} is minimized

In T_{opt}, let u and u' be the super nodes with $t(u) = t(u')$ separated by minimum distance (see Figure 4). Since T_{opt} is optimal, we know that $l(u'_1) \neq l(u'_2)$. Therefore, in T_{opt} there exists character y that mutates an odd number of times in the path connecting super nodes u and u'. Since $t(u) = t(u')$, the path connecting u and u' should have even mutations of every character corresponding to non-isolated vertices of the conflict graph G. This implies that $y \notin c(V_{nis})$ and therefore $\mathrm{mul}(y) = 1$; let $\mu(v_3, v_4) = y$ be the only mutation of character y. Note that because of our selection of u, u' there exists no super node $u'' \in$ path $u \rightarrow u'$ with $t(u'') = t(u)$. Therefore, there exists character x and vertices v_1, v_2, v_5, v_6 st $\mu(v_1, v_2) = \mu(v_5, v_6) = x$ and $(v_3, v_4) \in$ path $v_2 \rightarrow v_5$. This shows that the sandwich strength of T_{opt} is non-zero. Since $y \notin c(V_{nis})$ we know that $|G_{x,y}| < 4$. Therefore at least one of the four neighborhoods: $N_{x,y}(v_1)$, $N_{x,y}(v_2)$, $N_{x,y}(v_5)$, $N_{x,y}(v_6)$ is bad. The case when $N_{x,y}(v_1)$ is bad is easy to handle using Claim 3.3 and the property that $\mathrm{mul}(y) = 1$. Consider the case when $N_{x,y}(v_2)$ is bad. We now perform transform $\tau_{x,y}(N_{x,y}(v_2))$. As shown in the Figure, (y, x) no longer contributes to the sandwich strength. It is easy to see that there exists no additional character z such that (z, x) contributes to the sandwich strength after the transform. Therefore the sandwich strength reduces by at least one, a contradiction. The case when $N_{x,y}(v_5)$ or $N_{x,y}(v_6)$ is bad is symmetric. □

Corollary 3.7 *The tags found in step 5 of function* `buildNPP` *are unique.*

This completes the proof of correctness of function `buildNPP`. To conclude the analysis, we now show that every super node of T_{opt} contains a perfect phylogeny and then follow it with a proof of correctness of function `linkTrees`. Finally we provide proofs that the two guesses performed by our algorithm can be implemented efficiently.

Lemma 3.8 *The sub-phylogeny contained in every super node of every optimal phylogeny T_{opt} is perfect.*

Proof: For the sake of contradiction, assume not. Let edges e_1, e_2 lie in super node S_i with $\mu(e_1) = \mu(e_2) = c'$. Using Lemma 3.4 we know that the path connecting e_1 and e_2 should contain some character c'' s.t. $|G_{c',c''}| = 4$. However, neither character is a skeleton edge mutation, (i.e.) $c', c'' \notin \mu(E_s)$ and $c', c'' \notin c(V_{nis})$. This shows that c', c'' were isolated in the conflict graph, a contradiction to Corollary 3.5. □

Assume that skeleton S has been guessed and so $s(T_{opt}) = S(V_s, E_s)$. We can now use the skeleton S to decompose T_{opt} into connected components, each of which corresponds to a super node in V_s. We can therefore say that a vertex $v \in T_{opt}$ belongs to super node $u \in V_s$, if it lies in the connected component of T_{opt} that corresponds to u. Similarly, we can say that a skeleton edge $e \in E_s$ is the 'same' as an edge e' in T_{opt} if it joins the corresponding set of connected components.

Lemma 3.9 *In function* `linkTrees` *a vertex that is not present in P_i cannot be the root of the minimal subtree of T_{opt} containing all terminal vertices of S_i.*

Proof: Recall that the perfect phylogeny constructed in S_i is T_i. If a vertex $v \in T_i$ is not in P_i, then there exists character c s.t. $l(v)[c] = 0$ (wlog) and for all terminal vertices of T_{opt}, $v' \in S \setminus S_i$, $v'[c] = 1$. Furthermore, there exists $e, \mu(e) = c$ in T_i and therefore c is not a mutation of any skeleton-edge, $c \notin \mu(E_s)$. For the sake of contradiction assume that v is the root, then there exists $e, \mu(e) = c$ in $S_j \in S \setminus S_i$ of T_{opt}. Using Corollary 3.5 we know that $c \in V_{nis}$. This implies that $c \in \mu(E_s)$, a contradiction. □

We now informally describe the correctness of function `linkTrees`. Let R_i be the taxa assigned to super node S_i. Since the skeleton $s(T_{opt}, c(V_{nis}))$ was guessed and the tags are unique, R_i contains all the labels of terminal vertices of S_i in T_{opt}. Assume inductively that R_i contains the labels of the vertices of T_{opt} that connect S_i to any child. We can now construct a unique perfect phylogeny T_i. It is not hard to show the following two details. Even though a pair of characters might not share three gametes in R_i, the third gamete in T_i should be the same as the third gamete in the input for those characters. There exists an optimal phylogeny T_i s.t. there is no degree 2 Steiner vertex v is adjacent to the edge $(S_i, \text{parent}(S_i))$. We guess the root w correctly and therefore we can add vertex w' into $\text{parent}(S_i)$. This completes the perfect phylogeny for S_i, and w' added into $\text{parent}(S_i)$ ensures that the vertex that connects S_i and $\text{parent}(S_i)$ is present in $\text{parent}(S_i)$. The proof follows inductively.

We now analyze the two guesses performed by our algorithm.

Lemma 3.10 *The probability of a correct guess at Step 2f of function* `linkTrees` *is at least $O(q^{-1})$.*

Proof: We will prove the lemma by showing that $|P_i| \leq q + 1$. Consider any character c s.t. there exists $e \in S_i$ with $\mu(e) = c$. By the construction of set P_i if c has a single state s in $S \setminus S_i$, then $l(v)[c] = s$. This implies that for any two vertices $v_1, v_2 \in P_i$ if $l(v_1)[c] \neq l(v_2)[c]$, then character c mutates at least two times in T_{opt} – once in S_i and at least once in $S \setminus S_i$. Therefore there are at most q characters c in which the vertices of P_i can differ. Since T_i is a perfect phylogeny, each character c can mutate at most once in T_i. It follows that there are at most $q+1$ vertices in P_i. □

Corollary 3.11 *The probability all guesses at Step 2f in function* `linkTrees` *are correct is at least $q^{-O(q)}$.*

Proof: Each time we perform a guess at Step 2f of function `linkTrees`, with $|P_i| > 1$, we know that there exist edges $e_1 \in S_i$ and $e_2 \in S \setminus S_i$ s.t. $\mu(e_1) = \mu(e_2)$. In other words, we have a mutation of a character c in super node S_i that mutates at least once more in $S \setminus S_i$. This proves that there are at most $O(q)$ such super nodes S_i where a guess with $|P_i| > 1$ has to be performed. □

Lemma 3.12 *Step 3 of* `buildNPP` *can be implemented with correct guess probability atleast $(q + \kappa)^{-O(q)}$.*

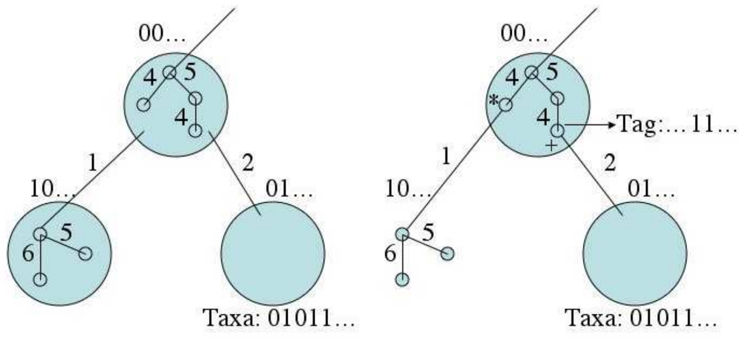


Figure 5: Detailed proof of Lemma 3.12. The initial skeleton super nodes u are filled circles. The skeleton $\rho(u)$ if present, is shown inside the corresponding super node u. Initially S is a perfect phylogeny. Super node of ρ skeleton marked with * is guessed. Super node marked with + is found based on the states of the taxa present in the child.

Proof: We first show that this probability is bounded by $(q+\kappa)^{-O(q+\kappa)}$. We start by guessing $\text{mul}(c', T_{opt})$ for all $c' \in c(V_{nis})$. Since there are $\binom{q+\kappa}{q}$ different ways of assigning $\text{mul}(c', T_{opt})$, the probability of guessing correctly is at least $(q+\kappa)^{-q}$. We can now guess the skeleton as a tree with $2q+\kappa$ labelled edges. Using a naive bound on the number of edge-labelled trees [18], we know that the probability of a correct guess is at least $(2q+\kappa+1)^{-(2q+\kappa-1)}$.

We first provide the intuition for the improved analysis. We can improve the previous implementation by first guessing the set of characters M for which $\text{mul}(c', T_{opt}) > 1$. We can now construct a unique perfect phylogeny P on the characters $c(V_{nis}) \setminus M$. The number of different ways of extending P by the addition of edges mutating M is bounded by $\kappa^{2q} q^{O(q)}$. This essentially proves the above lemma. We now provide the detailed proof.

As mentioned above we begin with guessing the characters M of $c(V_{nis})$ that mutate more than once. There are at most q such characters and so the probability that our subset is correct is at least κ^{-q-1}. We can now guess $\text{mul}(c')$ for all $c' \in M$ with overall probability $1/\binom{2q}{q}$. Using our observation we can construct a perfect phylogeny on the characters of $V_{nis} \setminus M$.

The perfect phylogeny is unique since for all pairs of characters $c', c'' \in V_{nis} \setminus M$, $|G_{c',c''}| = 3$ [12]. Since the all zeros taxa is present, we root skeleton S at the all zeros tag. This will be our initial skeleton S with the super node tags defined by the characters in $V_{nis} \setminus M$. We will now expand S so that it also contains mutations of characters in M. Note that in T_{opt} any mutation of a character in M lies in one of the super nodes of S. Since we already guessed $\text{mul}(c')$ for all c', we can now guess the supernode that contains each mutation of each character. This has a success probability of at least κ^{-2q}. For every super node $u \in S$ using the characters of M assigned to it, we can guess a second skeleton $\rho(u)$ by brute-force with overall probability of success $(2q+1)^{-2q-1}$, since the number of possible edge-labelled trees with k edges is $(k+1)^{k-1}$ [18]. For all $\rho(u)$ we can also guess the root and its tag with overall probability at least $(2q)^{-q}$. We now have a skeleton S and a skeleton $\rho(u)$ associated with some of the super nodes u of S as shown in Figure 5. The skeletons $\rho(u)$ will replace the super nodes u to obtain the final skeleton. For any u, note that the tags of $\rho(u)$ defined by the characters $\mu(e), e \in \rho(u)$ are unique due to Corollary 3.7.

The final step that remains is to replace u by $\rho(u)$ and link using the edges of S. We perform the linking bottom-up starting from the leaves and working up to the root of S. Since we have a perfect phylogeny, the tags of S are unique. Therefore for any super node $u \in S$, we know exactly the set of taxa that lie in the supernode. Every pair of characters $c', c'' \in V_{nis} \setminus M$ mutate just once in T_{opt} and share exactly three gametes. Therefore every super node u contains at least one taxa (not completely composed of Steiner vertices) in T_{opt}.

We now have four cases for any super node u and parent(u) of skeleton S based on whether each super node contains multiple mutations. If both super nodes do not contain any multiple mutations (i.e we do not have $\rho(u)$ or $\rho(\text{parent}(u))$), then there is nothing to do to process u. Similarly, if $\rho(\text{parent}(u))$ does not exist but $\rho(u)$ does, then we simply connect the root of $\rho(u)$ to parent(u). If $\rho(u)$ does not exist but $\rho(\text{parent}(u))$ does, then we find the supernode in $\rho(\text{parent}(u))$ that has the same tag as the taxa assigned to u. If both contain multiple mutations, then we need to guess a super node in $\rho(\text{parent}(u))$ to connect to the root of $\rho(u)$. This node can be guessed with probability $1/q$. Note that during this linking phase, we only perform a guess when both $\rho(u)$ and $\rho(\text{parent}(u))$ exists. Since the number of supernodes of S with multiple mutations is $O(q)$, the probability that all the guesses are correct during the bottom-up linking phase is $q^{-O(q)}$. $\square$

Derandomizing: Although we presented a randomized algorithm for ease of exposition, our algorithm can be naively derandomized. Consider the guesses performed in the implementation of step 3 of function `buildNPP` as described in the proof of Lemma 3.12. It is easy to see that every guess can be performed in time equal to the reciprocal of the probability. Similarly, instead of guessing a root from set P_i of function `linkTrees`, we simply enumerate all possible roots from set P_i.
Pre-processing input I: For every pair of characters c, c' if $|G_{c',c''}| = 2$, we (arbitrarily) remove character c''. Note that, at the end of this step, for all characters c', c'', $|G_{c',c''}| \geq 3$. Let the original input be I_1 and the processed input be I_2. It is not hard to show that since c', c'' are essentially identical, $\text{mul}(c', T_{opt}) = \text{mul}(c'', T_{opt})$ in any optimal phylogeny for I_1. Contracting any edge e s.t. $\mu(e)$ is a character c'' in I_1 but not in I_2, results in a phylogeny T_2 for I_2. We can now add back the characters c'' of I_1 that were absent in I_2 by placing them adjacent to mutation of every character c'. Since this is a phylogeny T_1 for I_1 with $\text{mul}(c', T_1) = \text{mul}(c'', T_1)$, it has to be the optimal phylogeny for I_1. The relative ordering of c', c'' determines the third gamete. Even though T_2 might not be optimal for I_2, it has to be the case that for all vertices $v_1, v_2 \in T_2$, $l(v_1) \neq l(v_2)$ since the labels of T_1 are unique. Furthermore, since T_1 is q-near perfect, contracted tree T_2 has to be q-near perfect. Therefore, our algorithm finds phylogeny T_2 as described above and can identical characters at the end to obtain T_1. Alternatively, we can arbitrarily declare a third gamete for every pair of characters c', c'' at the start of the algorithm without a pre-processing step.

4 Achieving FPT for BNPP

Fernandez-Baca and Lagergren [9] claim that the most important open problem in the area is to determine if the near-perfect phylogeny problem was W[1]-hard or fixed parameter tractable (FPT). We solve the open problem for the case when the number of states s is 2, by demonstrating a fixed parameter tractable algorithm. The algorithm and analysis is rather involved and therefore we provide a high level description in this section. The full details of the algorithm can be found in Section 5. The algorithm has the same flavor as the simpler algorithm but has substantial additional details.

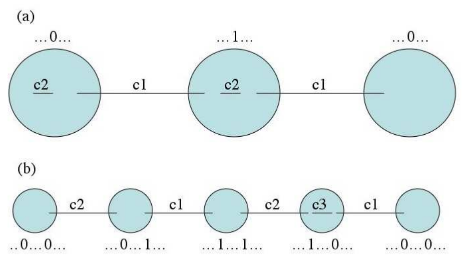


Figure 6: An example of how $s(T_{opt})$ can be found for a 2-near perfect phylogeny. The skeleton contains c_1 at the beginning. Distinguishing character c_2 that partitions tags 0 and 1 is first found. Character c_2 mutates multiple times and the skeleton still contains same tag supernodes. Final distinguishing character c_3 partitions tags $-1-0-$ and $-0-0-$ and mutates once. Tags after adding c_3 are unique

We begin by finding any $O(q)$ size vertex cover VC of the conflict graph G for input I. We now guess the skeleton $S(V_s, E_s)$ s.t. $\mu(e) \in c(VC)$ and penalty of S is at most q. Note that this is similar to Step 3 of function `buildNPP` except that $VC \subseteq V_{nis}$ and the κ term is eliminated from the running time of the step. However, we cannot prove the existence of unique tags and therefore Corollary 3.7 no longer holds. The step to partition the input taxa R (analogous to Step 5 of function `buildNPP`) into super nodes is the hardest step of the algorithm. This amounts to finding 'distinguishing edges':

Definition 13 *An edge $e \in E(T)$ or character $\mu(e)$ is distinguishing super nodes v_1 and v_2 w.r.t a phylogeny T if $|\{e' \in \text{ path } v_1 \rightarrow v_2 | \mu(e') = \mu(e)\}|$ is odd.*

Using the fact that in any T_{opt} there exists no two vertices $v_1, v_2 \in T_{opt}$ s.t. $l(v_1) = l(v_2)$, we can show that there exists a distinguishing edge for every pair of super nodes in T_{opt}. Assume for the sake of simplicity that there exists an oracle $\theta(v_1, v_2)$ that returns a distinguishing character wrt T_{opt} for super nodes v_1, v_2 efficiently. Then we have the following simple algorithm to partition R: Let $v_1, v_2 \in V_s$ with $t(v_1) = t(v_2)$; Let $c \leftarrow \theta(v_1, v_2)$; Add c to VC and guess new skeleton $S(V_s, E_s)$. We repeat the above procedure until the tags of all the super nodes are unique. At most q of the characters returned by θ can mutate multiple times in T_{opt}. Since after each call to θ we either find a multiple mutating character or a character that mutates once and is distinguishing, the algorithm terminates with $O(q)$ calls to θ. It is not hard to see that it spends $O(q^{q^2})$ total time in extending the skeleton one character at a time. Although, good enough for FPT, the technical report shows how to defer adding the characters, such that $O(q)$ characters can be added to S in a single step, thereby reducing the running time to $q^{O(q)}$.

We now have a skeleton S that contains unique tags. We can therefore proceed along the lines of the simpler algorithm: we execute function `linkTrees` as specified in the pseudo-code of Figure 3. The only part left therefore is implementing θ efficiently. To support this, we define equivalence classes V_t of super nodes in any skeleton S based on the equality of tags t. As shown in Figure 6, an edge (or the mutating character) that distinguishes two super nodes v_1, v_2 itself lies in a third super node v_3. If v_1, v_2 have no super node of the same tag in the path $v_1 \rightarrow v_2$ then the distinguishing characters have both states (0 and 1) on two different equivalence classes. We can use this piece

```
1. $G \leftarrow$ conflict graph for input $I$
2. $Q \leftarrow$ 2-approx VC of $G$; if $|Q| > 2q$ then return nil
3. for all trees $S$ with onto function $\mu : E(S) \rightarrow Q$ and $|S| \leq 3q$
   (a) label taxa with elements of $V(S)$
   (b) 'inside' each $v \in V(S)$
       i. build perfect phylogeny $T_v$ for taxa labeled $v$
   (c) link perfect phylogenies $T_v$ into final tree $T_f$
   (d) if $|T_f| < |T_o|$ then $T_o := T_f$
4. if $penalty(T_o) \leq q$ then return $T_o$ else return nil
```

Figure 7: Overview of the algorithm

of information to identify the distinguishing characters. In the following section, we describe the algorithm in detail.

5 FPT for general BNPP

A high level pseudo-code of the algorithm is given in Figure 7. The remainder of this section elaborates on each of the steps of this pseudo-code.

5.0.1 Preprocessing

Before beginning the main algorithm, a preprocessing step is performed to eliminate uninformative characters of the input. We use the usual notation $r[i]$, to denote the state in the i^{th} character of taxa r. The following definitions will be useful in understanding the preprocessing:

Definition 14 *The set of gametes $G_{i,j}$ for characters i, j is defined as: $G_{i,j} = \{(k,l) | \exists r \in R, r[i] = k, r[j] = l\}$. Two characters $i, j \in C$ contain (all) four gametes when $|G_{i,j}| = 4$.*

We begin preprocessing by removing all the characters that that have only a single state (i.e. they do not mutate at all). We then look for the pairs of input characters i, j for which $|G_{i,j}| = 2$. In such a case, both the characters contain the same information. We remove one of the two characters from the input. Only the remaining characters are 'informative' for a phylogenetic tree. As shown in the previous section, the preprocessing steps do not alter the overall running time or the correctness of the algorithm. It is well known that the most parsimonious phylogeny reconstructed is imperfect if and only if the input I contains the four-gamete property [12]. The pre-processing allows us to make the following claim, which simplifies subsequent steps of the algorithm:

Claim 5.1 *At the end of pre-processing, for all characters i, j, $|G_{i,j}| \geq 3$*

Note: We assume that the all zeros taxa is in I. If not, using our freedom of labeling, we convert the data so that it contains the same information with the all zeros taxa (See section 2.2 of [7]).

5.0.2 Finding the Conflict Graph G and Vertex Cover Characters Q

Our next step is to construct a *conflict graph* G corresponding to input I introduced by Gusfield et al. [14]. Each vertex $v \in V(G)$ of the graph represents a character $c(v) \in C$. An edge (u, v) is added if and only if all four gametes are present in $c(u)$ and $c(v)$. To find the vertex cover, we use the classical 2-approximation algorithm [5]. We set Q to be the set of characters corresponding to the vertex cover returned by the algorithm. We note the following simple observation that was proved in the previous section:

Lemma 5.2 *There exists no q-near-perfect phylogeny if* $|Q| > 2q$.

5.0.3 Partitioning Taxa

This is the most complicated step of the algorithm (step 3a in Figure 7) and to describe it we need some preliminary definitions:

Definition 15 *We define the following* notations*:*

- $l(v) \in \{0,1\}^m$*: the taxa that vertex* v *of the phylogeny represents*
- $r[i] \in \{0,1\}$*: the state in character* i *of taxa* r
- $\mu(e) : E(T) \to C$*: the character corresponding to edge* $e = (u, v)$ *with the property* $l(u)[\mu(e)] \neq l(v)[\mu(e)]$
- *mul*(c', T)*: for any character* c' *and phylogeny* T *is the number of times* c' *mutates in* T*, i.e* $|\{e \in T \mid \mu(e) = c'\}|$

Definition 16 *For any phylogeny* T *and set of characters* Q*:*

- *a* **super node** *is a maximal connected subtree* T' *of* T *s.t. for all edges* $e \in T', \mu(e) \notin Q$
- *the* Q-**skeleton** *of* T*,* $s(T, Q)$*, is the tree that results when all super nodes are contracted to a vertex. The vertex set of* $s(T, Q)$ *is the set of super nodes. For all edges* $e \in s(T, Q)$ $\mu(e) \in Q$*.*

Definition 17 *Notation* $\mathcal{P}_{v_1,v_2}(T)$ *denotes the path between (and including) vertices* v_1 *and* v_2 *of a tree* T*. If* v_1 *and* v_2 *are super nodes in* $s(T, Q)$ *and* T *is the phylogeny, then* $\mathcal{P}_{v_1,v_2}(T)$ *denotes the path in* T (not $s(T, Q)$) that connects super nodes v_1 and v_2.

We use the notation $< v_1, v_2, v_3, \cdots > \in T$ which denotes that there exists a path P in T that connects v_1, v_2, $v_3, \cdots$ in that order. Figure 1 shows an example of a phylogenetic tree along with its skeleton. Every super node v in the tree is *tagged* with the states of characters Q and $t(v)$ denotes the tag. By arbitrarily rooting the skeleton at super node v with $t(v) = (0, \cdots, 0)$, the edges of the skeleton define the tags for the other super nodes. **The goal** of this sub-section is to partition the set of input taxa and assign a subset to each super node so that a perfect phylogeny can be constructed on the taxa assigned to each super node. This task would be easy if the tags $t(v)$ of all super nodes of the skeleton are unique. If this is not the case, then we identify a subset of $O(q)$ characters that in addition to Q would enable us to construct an extended skeleton S' that contains unique-tags. Such a skeleton is identified in step 1c in Figure 8 and step 2 in Figure 9. We first describe an important structural property of T_{opt} and then explain the details of the routine.

5.0.4 Structure of T_{opt}

From now unless otherwise stated, Q is the set of characters corresponding to the vertex cover of the conflict graph. The following definitions are helpful:

Definition 18 *A vertex v is bad wrt a pair of characters (i, j) if $i, j \notin Q$ and $(l(v)[i], l(v)[j]) \notin G_{i,j}$. Note: We call a vertex v simply bad if $\exists(i, j)$ st v is bad wrt (i, j); only Steiner vertices can be bad.*

Note that this is slightly different from the definition provided in the previous section. We are now ready to prove an important property about the structure of an optimal phylogeny T_{opt}. The goal of this sub-section is to establish that if there exists an optimal phylogeny with penalty at most q, then there exists one without any bad vertices w.r.t Q.

The main idea of the proof is to take any optimum solution and to transform it into one which has the above property. This is achieved through a series of transformations that reduce the number of bad vertices. We first show that transform τ satisfies the following properties:

Lemma 5.3 *After applying the transform $\tau_{b,c}(N)$ (or $\tau_{c,b}(N)$), for all bad neighborhoods (b, c) of T', all vertices of the tree T' are good w.r.t (b, c).*

Proof: The vertices that were good w.r.t (b, c) stay good even after the transform. All the bad vertices w.r.t (b, c) were replaced by good vertices during transform. So no bad vertices are left. □

Definition 19 *We call a character c* clean *in a phylogeny T, if all the vertices of T are good w.r.t (c, x) for all characters x.*

Fact 5.4 *If the character x is clean in the phylogeny T and y is any other character, then the following properties hold:*

1. *Between any two mutations of character x, there is an even number of mutations of character y.*
2. *Between any two mutations of character y there is an even number of mutations of x.*

Fact 5.5 *If the phylogeny T does not have any bad vertices w.r.t (c, x), then the resulting phylogeny T' after applying transform $\tau_{b,c}$ will not have any bad vertices w.r.t (c, x).*

Lemma 5.6 *If the character x is clean, then the transform $\tau_{b,-}$ will not create any bad vertices w.r.t (b, x).*

Proof: We divide the proof in two cases: an odd number of b's are not sandwiched between two x's and an odd number of x's are not sandwiched between two b's after the transform τ_b. We prove the first case in Claim 5.8 and the second case in Claim 5.9. □

We will be working with bad neighborhoods $V_1, \cdots, V_k$ wrt characters (b, c).

Claim 5.7 *If the character x is clean in the phylogeny T, then mutations of x in T are all outside $\cup_i V_i$ or all inside.*

Proof: For the sake of contradiction, assume that there is a mutation of x inside V_1 and another one outside $\cup_i V_i$. Consider a path between these two mutations. It is easy to see that either b or c mutates odd number of times between the two mutations of x. This contradicts the fact that x is clean. □

Claim 5.8 *Transform $\tau_{b,c}$ does not create odd number of mutations of b between two mutations of a clean character x.*

Proof: Using the result of Claim 5.7, we consider following two cases.

case 1 (*All x's are outside*) Consider a pair of mutations of x. If it does not have any mutation of b or c between them, then their parity will not change. If there were an even number of mutations of b in between, then in transform $\tau_{b,c}$, an even number of mutations (twice the number of bad neighborhoods on the path) will get deleted and hence the parity won't change. If there were an even number of c's, then two mutations of b will be added for every bad neighborhood on the path. Overall, parity does not change.

case 2 (*All x's are inside*) In this case, every c gets replaced by cb. Since there were even number of b's and c's an even number of b's get created and an even number of b's get deleted between any pair of x's. Thus the parities do not change. □

Remark 1 *Claim 5.8 is required only if the character x mutates multiple times in the optimal phylogeny.*

Note that in transform $\tau_{b,c}$ only b's are deleted or added. So a similar claim for c holds automatically.

Claim 5.9 *The phylogeny T' resulting from transform $\tau_{b,c}$ does not leave odd mutations of a clean character x's between two mutations of b's.*

Proof: We show that between every adjacent pair of mutations of b in phylogeny T', there are even number of mutations of x. For the sake of contradiction assume that a pair of b's has odd number of x's in phylogeny T' resulting after the transform $\tau_{b,c}$ was applied. Let b_1 and b_2 denote the two mutations of b that sandwich an odd number of mutations of x. Note that both b_1 and b_2 cannot be present in the phylogeny T. Without loss of generality, assume that b_1 was created in phylogeny T'. Therefore, there is a mutation of character c next to b_1. Call it c_1.

In the phylogeny T, the character x was clean. Since the transform $\tau_{b,c}$ does not change in mutations of c, no mutation of x is sandwiched between two mutations of c in the new phylogeny T'.

If b_2 is a new mutation added to phylogeny T', then let c_2 be a mutation of the character c which is next to b_2 in the phylogeny T'. In this case, the odd number of mutations of x are sandwiched between c_1 and c_2 which is not possible. Therefore b_2 is present in the phylogeny T.

We use Claim 5.7 to separate the following two cases:

case 1 (*All x's are outside*) Let $N_{b,c}$ denote the bad neighborhood where b_1 was created. Let $\hat{b}_1$ be a mutation of b that was deleted from the boundary of $N_{b,c}$. Then the path from $\hat{b}_1$ to c_1 in phylogeny T contains no mutations of x, while the path from c_1 to b_2 contains an odd number of mutations of x. Therefore, in phylogeny T there are an odd number of mutations of x between $\hat{b}_1$ and b_2, which contradicts the fact that x was clean in phylogeny T.

case 2 (*All x's are inside*) Note that b_2 could not have been on the boundary of a bad neighborhood in the phylogeny T, otherwise it would have been removed in the phylogeny T'. Since all the mutations of x were inside bad neighborhoods, at least one of the neighborhoods on the path from b_1 to b_2 had an odd number of mutations. If this bad neighborhood is the one where b_1 was created in the transform, then the path b_1 to b_2 includes the mutation c_1. In T', all the vertices are good w.r.t (b, c). Hence there must be another mutation of c (call it c_2) between b_1 and b_2. Note that the path c_1 and c_2 contains an odd number of mutations of x. This is a contradiction. Therefore, the bad neighborhood containing odd number of mutations of x is not the one where b_1 was created. But in that case, consider the path from b_1 to b_2 in the old phylogeny T. It must have entered and exited the bad neighborhood via mutations of b. Thus the odd number of mutations of x were sandwiched between two mutations of b in the old phylogeny T. This is a contradiction to the fact that x was clean. $\square$

We will now use transformation τ and the preceding lemmas about it to make the following claim: The following claim follows immediately from the definition of transform τ:

Claim 5.10 *The optimal phylogeny contains equal number of mutations of characters b and c.*

Proof: Assume for purposes of contradiction that the claim is untrue. Then applying transformation τ will reduce the number of mutations in the phylogeny. This contradicts the optimality of the phylogeny. $\square$

We are now ready to prove the main result of this sub-section:

Theorem 5.11 *If there exists a q-near-perfect phylogeny that has bad vertices (w.r.t Q) then there exists a q-near-perfect phylogeny that does not contain any bad vertices (w.r.t Q).*

Proof: Consider a lexicographic order on the characters $c_1, c_2, \ldots$. We start with an optimum phylogeny T. We first make character c_1 clean by considering bad vertices w.r.t (c_1, c_i) for each i and applying transform τ_{c_i,c_1}. Note that the transform τ_{c_i,c_1} will not create bad vertices w.r.t $(c_1, c_{i'})$. From Lemma 5.3, it follows that at the end of this step character c_1 will be clean.

Now inductively assume that characters $c_1, \ldots, c_j$ are clean. Next we look at all the bad vertices w.r.t (c_{j+1}, c_i) (for $i > j+1$) for each i and apply transform $\tau_{c_i,j+1}$. Note that the transform $\tau_{c_i,j+1}$ does not create any bad vertices w.r.t $(c_j, c_{i'})$. Moreover, by Lemma 5.6, we can say that the resulting phylogeny has characters $c_1, \ldots, c_j$ clean. Also, it follows from Lemma 5.3 that at the end of this step the character c_{j+1} is clean.

Therefore, in the end all the characters are clean. In other words, there are no bad vertices in the phylogeny. $\square$

This theorem immediately leads to the following corollary:

Corollary 5.12 *In T_{opt}, if there exist two edges $e = (v_1, v_2)$ and $e' = (v_3, v_4)$ such that $\mu(e) = \mu(e') = i$, where $i \notin Q$, $< v_1, v_2, v_3, v_4 > \in T_{opt}$ then for all characters $j \notin Q$, $|\{e \in \mathcal{P}_{v_2,v_3}(T_{opt}) | \mu(e) = j\}|$ is even and therefore $l(v_1)[j] = l(v_4)[j]$.*

Proof: For the sake of contradiction, assume that $|\{e \in \mathcal{P}_{v_2,v_3}(T_{opt}) | \mu(e) = j\}|$ is odd. Consequently, $l(v_1)[j] \neq l(v_4)[j]$. However, note that $l(v_1)[j] = l(v_2)[j]$ and $l(v_3)[j] = l(v_4)[j]$. Now consider the character i. For this character, we can say that $l(v_1)[i] = l(v_4)[i]$ and $l(v_2)[i] = l(v_3)[i]$, but $l(v_1)[i] \neq l(v_2)[i]$. Thus we see that the vertices v_1, v_2, v_3, v_4 give us all four gametes of the pair (i, j). Since both $i, j \notin Q$, we get a contradiction to Theorem 5.11. □

5.0.5 function `partition`

For the rest of the paper, we fix an optimal phylogeny T_{opt} that does not contain any bad vertices. The pseudo-code for function `partition` is given in Figure 8. We will prove briefly describe the function and prove its correctness. The following definitions are important for the proof of correctness:

Definition 20 *Two super nodes v_1 and v_2 of $s(T, Q)$ are tag-adjacent if*

1. $t(v_1) = t(v_2)$ and

2. there exists no super node $v_3 \in \mathcal{P}_{v_1,v_2}(s(T, Q))$ st $v_3 \neq v_1, v_2$ and $t(v_3) = t(v_1)$.

Definition 21 *Two super nodes v_1 and v_2 of $s(T, Q)$ are paired-tag-adjacent if there exists no super node $v_3 \neq v_1, v_2$ in $\mathcal{P}_{v_1,v_2}(s(T, Q))$ st $t(v_3) = t(v_1)$ or $t(v_3) = t(v_2)$.*

Definition 22 *An edge $e \in E(T)$ or character $\mu(e)$ is distinguishing super nodes v_1 and v_2 w.r.t a phylogeny T if $|\{e' \in \mathcal{P}_{v_1,v_2}(T) | \mu(e') = \mu(e)\}|$ is odd.*

It is easy to see the following claim:

Claim 5.13 *For any optimal phylogeny T_{opt}, for all vertices $v_1, v_2 \in T_{opt}$, $l(v_1) \neq l(v_2)$.*

Using Claim 5.13 it is then easy to prove that for the optimal near-perfect phylogeny T_{opt} there exists a distinguishing edge for every pair of tag-adjacent super nodes. We define equivalence classes V_t containing super nodes based on the equality of tags t of super nodes. It is straightforward to partition the taxa R for each equivalence class of super nodes using the tags of each equivalence class. At any point in time, the algorithm maintains a partition $P(V_{t_i})$ for each equivalence class. Each recursive call refines one of the partitions of one of the equivalence classes. To avoid excessive notations we will use $P(V_{t_i}) = \{S_1, \cdots, S_k\}$ for all i. It should be clear from the context which equivalence class S_i belongs to. We say that a character i partitions a set S_j if there exists two taxa $r_1, r_2 \in S_j$ such that $r_1[i] = 0$ and $r_2[i] = 1$. The partition can be written as $S_{j_0} = \{r_1 \in S_j | r_1[i] = 0\}$ and $S_{j_1} = \{r_2 \in S_j | r_2[i] = 1\}$.

The value $mark(v)$ denotes our guess as to whether or not the super node v contains any terminal vertices in T_{opt}. The value is set to ‘terminal’ if we guess that there exists at least one terminal vertex in v and ‘steiner’ otherwise. The skeleton S that is passed into the initial call of function partition is assumed to have the correct $mark$ values guessed. A partition $P(V_{t_i})$ is complete when the number of sets in $P(V_{t_i})$ is equal to the number of super nodes v in V_{t_i} with $mark(v) =$ terminal, that is: $|P(V_{t_i})| = |\{v \in V_{t_i} | mark(v) = \text{terminal}\}|$.

```
function partition ( $P(V_{t_1}), \cdots, P(V_{t_p})$ , int list singleMutations, skeleton $S$ )
1. if $\forall i$, $P(V_{t_i})$ is complete then
    (a) guess the assignments of the partitions $S_i$ to the super nodes of the skeleton
    (b) guess expansion of skeleton $S$ into $S'$ using characters in singleMutations
    (c) return $S'$
2. let $D_2 = \{i | \exists_{>1} j, \exists k$ s.t. $i$ partitions $S_k \in P(V_{t_j})\}$
3. if $|D_2| \leq q + 1$
    (a) guess expansion of skeleton $S$ into $S'$ using characters in singleMutations and $D_2$
    (b) guess $mark(v)$ for all super nodes $v \in S'$
    (c) use-interface ( $S'$ )
4. else
    (a) consider any $D_2' \subseteq D_2$ s.t. $|D_2'| = q + 1$
    (b) $\forall$ $i \in D_2'$ and for any two $j$ s.t. $\exists S_k \in P(V_{t_j})$, $i$ partitions $S_k$
        i. let $P(V_{t_j}) = (P(V_{t_j}) \backslash \{S_k\}) \cup \{S_{k_0}\} \cup \{S_{k_1}\}$
        ii. partition ( $P(V_{t_1}), \cdots, P(V_{t_p})$, singleMutations $\cup$ $\{i\}$, $S$ )
```

Figure 8: Algorithm to partition and assign taxa of the input to super nodes

The partition function initially finds the set D_2 consisting of (distinguishing) characters that refine the partition in at least two different equivalence class. If $|D_2| \geq q + 1$, then the function picks an arbitrary set of $q+1$ characters and performs $2(q+1)$ recursive calls on partition induced by each character on two arbitrary equivalence classes (steps 4 to 4(b)ii). If $|D_2| \leq q$, the recursion bottoms out, and the function guesses all the mutations of the characters in D_2, *singleMutations* and adds them to $s(T, Q)$ (steps 3 to 3c). Note that each added edge splits a single super node into two new super nodes. When we guess the extension, we assume that characters in *singleMutations* mutate exactly once and characters in D_2 mutate once or more. The following two lemmas (lemma 5.14 and lemma 5.15) are used to establish that D_2 becomes empty at the end of step 3a and remains empty and to show the correctness of function partition:

Lemma 5.14 *At any point if D_2 does not contain some character c, then c will not be in D_2 at any later point in the algorithm (specifically after step 3a of function* partition *described in Figure 8 and step 4(d)i of function* use-interface *described in Figure 9).*

Proof: For the sake of contradiction, assume that D_2 did not contain character c and after expanding using character d, c partitions two different equivalence classes V_{t_1} and V_{t_2}. Clearly t_1 and t_2 should have been the same tag before the expansion using character d. Consider terminal vertices $v_1, v_2 \in V_{t_1}$ and $v_3, v_4 \in V_{t_2}$, st $l(v_1)[c] \neq l(v_2)[c]$ and $l(v_3)[c] \neq l(v_4)[c]$. But we know that $l(v_1)[d] = l(v_2)[d] \neq l(v_3)[d] = l(v_4)[d]$. This immediately implies that c and d share four gametes in T_{opt}, a contradiction since neither is in Q. □

```
function use-interface ( skeleton S )
1. let $D_1 = \{i \in C | \exists_1 j,\ i$ partitions $R(V_j)\}$.
2. select some $V_j$ that is incomplete, return $S$ if none exists
3. select a lowest pair $v_1, v_2 \in V_j$ of tag-adjacent super nodes, s.t. $mark(v_1) = mark(v_2) =$ terminal
4. if $\exists h \in D_1$, $v_3 \in V_{t'}$ and $v_4 \in V_{t''}$, for some $t' \neq t''$ and $j \neq t', t''$ such that
   (a) $mark(v_3) = mark(v_4) = terminal$
   (b) $h$ partitions $R(V_{t'}) \cup R(V_{t''})$
   (c) $\mathcal{P}_{v_1,v_2}(S)$ and $\mathcal{P}_{v_3,v_4}(S)$ share a super-node of $S$
   (d) $(v_1, v_3)$ and $(v_2, v_4)$ are paired-tag-adjacent then
       i. guess extension $S'$ of skeleton $S$ by mutating character $h$ (once or multiple times)
       ii. guess $mark(v)$ for super nodes adjacent to mutation(s) of $h$
       iii. use-interface ( $S'$ )
5. else return nil
```

Figure 9: Algorithm to find interface edges and extend the skeleton

For any equivalence class V_{t_i}, consider taxon partition $P_{opt}(V_{t_i})$ for super nodes of skeleton $s(T_{opt}, Q)$.

Definition 23 *A partition $P(V_{t_i}) = \{S_1, \cdots, S_k\}$ is* good *if $\forall S_j^{opt} \in P_{opt}(V_{t_i})$ $\exists S_i \in P(V_{t_i})$, s.t. $S_j^{opt} \subseteq S_i$.*

Informally, a partition is good if it can be refined by further partitions into that of T_{opt} for skeleton $s(T_{opt}, Q)$ defined on Q. A set of partitions $\{P(V_{t_1}), \cdots, P(V_{t_p})\}$ is good if every partition is good.

Lemma 5.15 *If the argument set of partitions $P(V_{t_i})$ is good for a call of* `partition`*, then the argument set of partitions $P(V_{t_i})$ for at least one of the recursive calls (step 4(b)ii) is good.*

Proof: Consider any one call to partition function and assume inductively that its argument is good. We recursively perform partition in step 4(b)ii. If $|D_2| \geq q + 1$ then there exists at least one character $i \in D_2$ that mutates only once in T_{opt}. A character i that mutates only once induces a partition on $P(V_{t_j})$ only when two super nodes (of the same equivalence class V_{t_j}) contain e with $\mu(e) = i$ in the path connecting them or if i mutates inside a super node $v \in V_{t_j}$. We know however that i partitions at least two different equivalence classes $V_{t_j}, V_{t_{j'}}$. Since i mutates once, it can mutate inside the super node of at most one of V_{t_j} or $V_{t_{j'}}$ and lie in the path connecting two super nodes of the other equivalence class. We recurse on both the partition induced on $P(V_{t_j})$ and $P(V_{t_{j'}})$. Moreover, since the arguments of partition is good, the partition induced by using i on at least one of $S_j \in P(V_{t_j})$ or $S_j \in P(V_{t_{j'}})$ is good. $\Box$

Correctness: From the above Lemma, it is clear that at Step 1a, if the argument set of partitions is good then $P(V_{t_i}) = P_{opt}(V_{t_i})$ for all t_i and skeleton S' found in Step 1c has $O(q)$ edges and unique tags. This shows one part of the correctness of function `partition`. Assuming that all guesses are correct, we can further claim that the skeleton passed into function `use-interface` is $O(q)$ in size, D_2 is empty for S' and S' is the same as the skeleton of T_{opt} defined over Q, D_2 and *singleMutations.*

5.0.6 function `use-interface`

In the goal of trying to figure out a $O(q)$ sized skeleton that has unique tags, we are left with one last case: discovering distinguishing characters that mutate once or more but partition only one equivalence class of skeleton S that is passed into function `use-interface`. Note that the skeleton S and the tags of S at the beginning of `use-interface` are defined over the characters of D_2 and *singleMutations* in addition to Q. Define $R(V_t) \subseteq R$ for any equivalence class V_t to be the set of taxa that match tag t. Consider the partition of taxa defined by T_{opt} on any one equivalence class V_t of super nodes.

Definition 24 *An edge e in super node v_1 is* internal *if there exists two terminal vertices of T_{opt}, $u, v \in v_1$ st $l(u)[i] \neq l(v)[i]$. We define an edge e that is not internal as* interface.

Since D_2 is empty, $\forall v_1, v_2 \in V_t$ that are tag-adjacent, every distinguishing edge is an interface edge. By definition, a distinguishing character i for super nodes v_1, v_2 with $mark(v_1) = mark(v_2) = terminal$ has the following property: there exists terminal vertices $u \in v_1$, $v \in v_2$ st $l(u)[i] \neq l(v)[i]$.

Definition 25 *Two tag adjacent terminal super nodes v_1 and v_2 are defined to be* lowest pair *in $s(T_{opt}, Q)$ if there exists vertex $x \in T_{opt}$ st for all terminal super nodes $v_3 \in s(T_{opt}, Q)$ with $t(v_3) = t(v_1)$, $x \in \mathcal{P}_{v_1,v_3}(T_{opt})$ and $x \in \mathcal{P}_{v_2,v_3}(T_{opt})$. The vertex x is called the branch point of v_1 and v_2.*

The next lemma shows that for every lowest pair of super nodes in T_{opt}, there exists at least one distinguishing interface edge that has the properties that the pseudo-code uses to identify them in step 4 of function `use-interface`:

Lemma 5.16 *In T_{opt}, for every lowest pair super nodes $v_1, v_2 \in V_t$ with branch point x such that every distinguishing edge of v_1 and v_2 is an interface edge, there exists at least one distinguishing edge e, $\mu(e) = h$ with the following properties(P):*

1. *character h partitions $R(V_t)$*
2. *$\exists v_3 \in V_{t'}, v_4 \in V_{t''}$, $t \neq t', t''$ s.t. v_1, v_3 and v_2, v_4 are paired-tag-adjacent and h partitions $R(V_{t'}) \cup R(V_{t''})$ and*
3. *the paths $\mathcal{P}_{v_1,v_2}(s(T_{opt}, Q))$ and $\mathcal{P}_{v_3,v_4}(s(T_{opt}, Q))$ share a super-node.*

Proof: First note that the character h that partitions $R(V_{t'}) \cup R(V_{t''})$ can only partition it into $\{R(V_{t'}), R(V_{t''})\}$. This is because h induces a partition only in one equivalence class V_t and $t \neq t', t''$. For the same reason $t' \neq t''$. Throughout the proof of the lemma, the distance between two vertices (or edges) on a phylogeny is simply the number of edges in the path connecting the vertices (or edges).

We will make use of a very simple transform, $\rho_{c,c'}$, that operates on any phylogeny T and is defined over two characters c, c' and a Steiner vertex v. To apply transform $\rho_{c,c'}$ on Steiner vertex

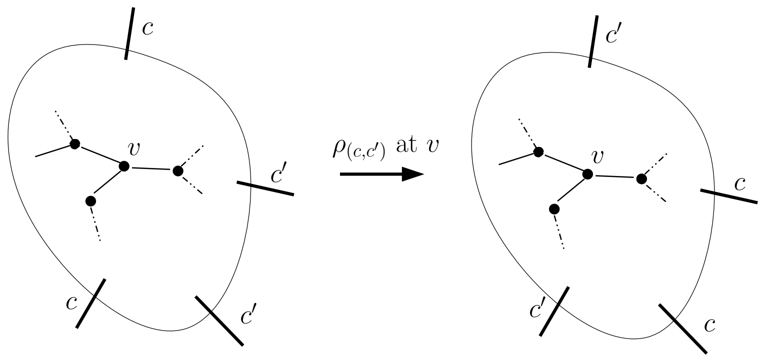


Figure 10: Application of transform ρ. The edge-labels c and c' are interchanged.

v, all vertices v' such that v' is the only terminal vertex in $\mathcal{P}_{v,v'}(T)$ should satisfy the following property. There exists edge $e \in \mathcal{P}_{v,v'}(T)$ s.t. $\mu(e) = c$ or $\mu(e) = c'$. An example of when such a transform can be applied is shown in Figure 10. For all such paths from v to v', the transform replaces the first mutation of c with c' and vice-versa. The result after applying the transform is shown in Figure 10. Clearly, the transform does not change the cost of the phylogeny.

We know that for an optimal phylogeny, there has to exist at least one distinguishing edge h for v_1 and v_2. Let u_1 and u_2 be the vertices in v_1 and v_2 respectively, that lie on either ends of $\mathcal{P}_{v_1,v_2}(T_{opt})$. Therefore h has to mutate odd number of times in exactly one of $\mathcal{P}_{u_1,x}(T_{opt})$ and $\mathcal{P}_{x,u_2}(T_{opt})$. Without loss of generality assume that it mutates an odd number of times in $\mathcal{P}_{u_1,x}$. Consider the edge e closest to u_1 such that $\mu(e) = h$. In T_{opt} if the path from e to u_1 consists entirely of degree 2 steiner vertices, then the mutation of e can be incrementally swapped with the mutation of the neighboring edge, until the mutation of h is moved inside super node v_1. Note that swapping adjacent edges of a degree 2 steiner vertex is just a trivial application of ρ on the vertex. After several applications of ρ, character h is no longer distinguishing between v_1 and v_2 in the new optimal solution.

If the path from e to u_1 contains a vertex w of degree greater than 2, then every branching path from w leads to a terminal vertex (since steiner vertices can not be a leaf). Consider any such terminal vertex r, the super node in which r lies cannot have tag t since it contradicts the assumption that v_1 and v_2 are a lowest pair. Now consider the case when for all such terminal vertices r, the path $\mathcal{P}_{w,r}(T_{opt})$ contains a mutation of h. Using transform ρ, the mutation of h can be swapped with neighboring edges until it is adjacent to w. Once again let e be the edge adjacent to w, st $\mu(e) = h$. Let $e' \in \mathcal{P}_{u_1,x}, e \neq e'$ be another edge adjacent to w. We can now apply transform $\rho_{\mu(e'),h}$ on vertex w. The result of the transform is that the mutations of h and $\mu(e')$ swap positions. We can therefore keep applying transform ρ to move the mutation of h closer to u_1. There are only two cases when we can not apply the transform any further. In both the cases described below, we find super node v_3 that contains a terminal vertex.

Case 1: If the path from e to u_1 contains a terminal vertex r, that belongs to a super node v_3 say, then $t(v_3) \neq t(v_1)$, since we assumed that v_1 and v_2 are tag-adjacent.

Case 2: If the path from e to u_1 contains a vertex w of degree greater than 2, that contains a branch leading to a terminal vertex $r \in v_3$, where $t(v_3) \neq t(v_1)$ such that the path from w to r does not contain a mutation of h. Note that $t(v_3) \neq t(v_1)$ since v_1 and v_2 are a lowest pair.

Now consider the edge $e \in \mathcal{P}_{u_1,x}(T_{opt})$ such that $\mu(e) = h$ and the distance of e from v_1 is the largest. Using a series of transforms ρ, we can move the mutation of h toward x. Once again with a similar argument we can either claim that there exists a terminal vertex r' in a super node v_4 that either lies in the path from e to x or is in the induced phylogeny of w which lies in the path from e to x. If this is the case, then r and r' are two terminal vertices separated by an odd number of mutations of h. Also, $t(v_1) \neq t(v_3), t(v_4)$. This satisfies all the properties of the lemma.

Now consider the case when the mutation of h can be moved so that it is adjacent to x. Now consider any vertex cover character c that mutates an odd number of times between x and v_2. There has to exist at least one such edge since x can not belong to a super node of tag t as v_1 and v_2 are tag-adjacent. Consider the edge e' such that $\mu(e') = c$ and the distance from e' to v_2 is maximum. First assume that all the vertices in $\mathcal{P}_{u_2,x}(T_{opt})$ are of degree 2 and steiner. Clearly now, using the trivial ρ transform, the mutation of c can be moved so that it is adjacent to x. We can now perform transform $\rho_{c,h}$ on x to obtain a new optimal phylogeny where the mutations of h and c adjacent to x have switched places. Note that the condition of $\rho_{c,h}$ requires that every path from x to a terminal vertex contains either a mutation of h or a mutation of c. This can be proved as follows for the optimal phylogeny T_{opt}. Consider any path from x to the first terminal vertex r' that is in a super node v_4. If $t(v_4) = t(v_2)$ then the path from x to v_4 should contain an odd number of mutations of c, since the path from x to v_2 contains an odd number of mutations of c. If $t(v_4) \neq t(v_2)$, then the path from x to v_4 should contain an odd number of mutations of h, otherwise r' and r that lie in v_3 and v_4 respectively satisfy the properties of the Lemma. After applying $\rho_{c,h}$, we can move the mutation of h inside v_2 since all the vertices are degree 2 the result of which is that h is no longer distinguishing between v_1 and v_2.

The last case that is left to analyze is when the path from x to e' contains either terminal vertices or steiner vertices of degree greater than 2. If it contains a terminal vertex r', then r' along with r satisfy the properties of the Lemma. If there exists a vertex w that has degree greater than 2, then the induced subtree rooted at w contains terminal vertices, and all paths to terminal vertices should contain a mutation of h (otherwise we can use the terminal vertex along with r to satisfy the properties of the lemma). Therefore $\nexists e''' \in \mathcal{P}_{x,u_2}$ with $\mu(e''') = h$, since otherwise we can move the mutation of h using a series of ρ transforms so that it is adjacent to x. This is a contradiction to optimality since we already have e adjacent to x with $\mu(e) = h$. Therefore, we can directly apply transform $\rho_{h,c}$ on vertex x. We can then move the mutation of h into u_2 with a series of ρ transforms.

Putting everything together, we have the fact that if for any distinguishing character h there exists no super nodes that satisfy the properties of the lemma, then the edge corresponding to the mutation of h can be moved such that it is no longer distinguishing. Furthermore, no new character becomes distinguishing in this process. We can continue this argument for every distinguishing character. If none satisfy the properties of the lemma, then eventually the first and last vertices of the path $\mathcal{P}_{v_1,v_2}(T_{opt})$ are identical, a contradiction to optimality. □

Note that the converse — only distinguishing interface edges satisfy these properties — is not true. We do prove, though, that only internal edges that mutate multiple times can satisfy them. Intuitively, this shows that not too many internal characters are are used in Step 4(d)i of `use-interface`.

Lemma 5.17 *If internal edge $e, \mu(e) = h$ satisfies P then h mutates more than once in T_{opt}.*

Proof: Let e be an internal edge satisfying all the properties. Note that at step 4 of function

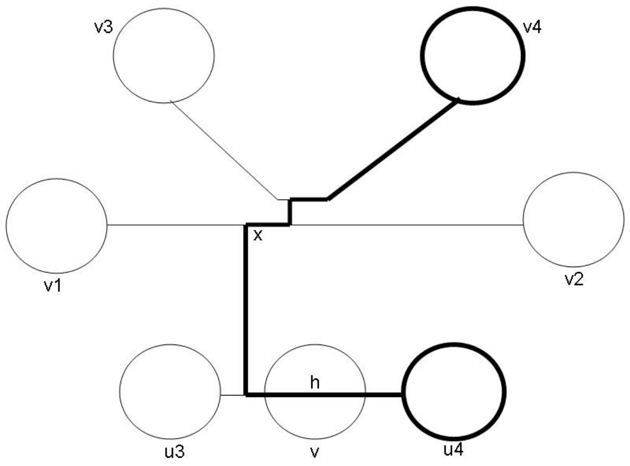


Figure 11: Proof that character h has to mutate multiple times to be an internal character that also satisfies the properties of a distinguishing interface character.

use-interface, character h partitions V_t and also partitions $R(V_{t'}) \cup R(V_{t''})$. Since h is an internal character, there exists internal edge e'' in some super node $v \in V_t$ s.t. $\mu(e'') = h$. See Figure 11. Super node $v \notin \mathcal{P}_{v_3,v_4}$ because we assumed that (v_1, v_3) and (v_2, v_4) are paired-tag adjacent. Assume that h lies in $\mathcal{P}_{u_3,u_4}$ in T_{opt} where $u_3 \in V_{t'}$ and $u_4 \in V_{t''}$. Note that for h to partition $R(V_{t'}) \cup R(V_{t''})$ it has to be the case that both u_3 and u_4 contain at least one terminal vertex. Note that one of u_3 or u_4 could be v_3 or v_4 respectively.

Consider the two paths $\mathcal{P}_{v_3,u_3}$ and $\mathcal{P}_{v_4,u_4}$ (shown in Figure 11). If h mutates just once, then it should be the case that the mutation of h, edge e'', should lie in one of the two paths. This is however a contradiction since h now partitions two equivalence classes V_t and one of $V_{t'}$ or $V_{t''}$ (since v_3, u_3, v_4 and u_4 each contain at least one terminal vertex in T_{opt}). □

Correctness: Inductively assume that the skeleton S passed into any call of `use-interface` is the same as $s(T_{opt}, Q)$ where both skeletons are defined over some subset Q' of the characters. It is easy to see that for any V_j, there has to exist two super nodes v_1, $v_2 \in V_j$ that are a lowest pair. Using Lemma 5.16 we know that there exists a character h that satisfies properties P. Steps 4(d)i and 4(d)ii simply guesses the locations of h and the mark as present in T_{opt}. Assuming that the guesses are correct, the skeleton S passed in the call to `use-interface` in Step 4(d)iii has to be the same as $s(T_{opt}, Q)$ defined over the characters $Q' \cup \{h\}$. If the argument S is the same as $s(T_{opt}, Q)$ for some set of characters Q' and all V_j are complete, then skeleton S that is returned has unique tags and the correctness follows inductively. Furthermore, it is easy to see that the returned skeleton S has $O(q)$ edges as follows. Using Lemma 5.17, we know that in the returned skeleton S, the sum of the number of internal edges and the number of multiple mutant interface edges is at most q. When adding any single mutant interface edge h to the skeleton, the number of terminal super nodes remains the same. This is because, by the definition of interface edges, one of the two newly created super nodes is marked steiner. However, the number of equivalence classes

```
function linktrees ( skeleton S )
1. let $S_i \leftarrow$ any leaf super-node of $S$
2. build perfect phylogeny $T_i$ on the taxa assigned to $S_i$
3. let $P_i \leftarrow$ the set of vertices of $T_i$
4. for each character $c$ st $\exists e \in S_i$ with $\mu(e) = c$, do
   if $\exists k \in \{0,1\}$, $\forall v_1 \in S \setminus S_i$, $l(v_1)[c] = k$, then remove all vertices $v$ from $P_i$ st $l(v)[c] \neq k$
5. guess a vertex $w$ from $P_i$, and let $l(w')[i] = l(w)[i]$ for all $i \neq \mu(e)$ and $l(w')[\mu(e)] \neq l(w)[\mu(e)]$,
   where edge $e$ is $(S_i, p(S_i))$
6. add $w'$ to $p(S_i)$
7. if $(S \setminus S_i = \Phi)$ then return $T_i$ else return (linktrees($S \setminus S_i$) $\cup T_i \cup (S_i, p(S_i))$ )
```

Figure 12: Algorithm to build the tree using the skeleton and partition

increases by at least one. This is because, v_1 and v_2 belonged to the same equivalence class before adding interface character h. After adding h, however, the tags of v_1 and v_2 differ in character h. Therefore, after adding q single mutant distinguishing interface edges, all V_j are complete. This completes the proof of correctness for function `use-interface`.

5.0.7 Building and Linking Perfect Phylogenies

We now show the final step to complete the near-perfect phylogeny (Step 3c of Figure 7. From the proofs of correctness, after `partition` and `use-interface`, the resulting skeleton S' (say) has $O(q)$ edges and unique tags. By simple book-keeping we can project partitions defined by S' back to the skeleton S defined just over Q. Recall that S was found in Step 3 of the pseudocode in Figure 7 and passed into the first call to function `partition`. From now we will simply work with skeleton S defined over Q. This keeps the analysis simple and allows us to perform Step 3a of Figure 7. We can prove the following intuitive property on the optimal phylogeny T_{opt} similar to the statement in the previous section. The proof is different since we do not assume that the skeleton contains all characters corresponding to non-isolated vertices of the conflict graph.

Lemma 5.18 *Every super node of the skeleton $s(T_{opt}, Q)$ contains a perfect phylogeny.*

Proof: For the sake of contradiction, assume that there exist edges e, e' in super node u with $\mu(e) = \mu(e') = c$. Let $\mathcal{P}$ denote the connecting edges e and e'. There exists no character $c' \notin Q$ that mutates an odd number of times in $\mathcal{P}$ according to Corollary 5.12. By the definition of the super-nodes and skeleton we know that there exists no edge $e \in u$ with $\mu(e) = c$ if $c \in Q$. For optimality, we can not have two vertices v and v' with $l(v) = l(v')$. Therefore there can not exists edges e and e' as stated. □

We discover the edges of T_{opt} bottom up in function `linktrees` explained in Figure 12. We can reconstruct a unique perfect phylogeny that has no bad vertices (Step 2) within each S_i as follows. Restricting to the taxa assigned to S_i, we know that for any pair of characters $(i, j), |G_{i,j}| \leq 3$.

If $|G_{i,j}| = 2$, then we arbitrarily remove i. If $|G_{i,j}| = 1$ then we remove both i and j. After this preprocessing, every pair of characters contain exactly three gametes. Therefore a unique perfect phylogeny can be built [12]. For every pair (i, j) st $|G_{i,j}| = 2$, we add back character i adjacent to mutation of j. The relative ordering of i and j is determined based on the third gamete that is present in the input. Let the parent of a vertex v in a rooted tree be denoted by $p(v)$. The following Lemma is trivial and assumes that phylogeny T_{opt} is rooted at the all zeros taxa:

Lemma 5.19 *In rooted phylogeny T_{opt}, consider any induced subtree T'. If $T_{opt} \setminus T'$ contains vertices v and v' st $l(v)[c] = 0$ and $l(v')[c] = 1$ then there exists an edge e st $\mu(e) = c$ in $T_{opt} \setminus T'$.*

Proof: Assuming not, the mutation of character c must occur in T'. But T' and $T_{opt} \backslash T'$ are connected by a single edge. Thus, $T_{opt} \backslash T'$ contains only one of either 0 or 1 in character c, a contradiction. $\square$

Corollary 5.20 *If both T' and $T_{opt} \backslash T'$ each contain both states of some character c, then there are at least 2 edges $e, e' \in T_{opt}$ s.t. $\mu(e) = \mu(e') = c$.*

Proof: The proof follows directly from lemma 5.19 by treating T' and $T_{opt} \backslash T'$ of the corollary as the induced subtree T' of the lemma. $\square$

Note that in the rooted T_{opt}, the perfect phylogeny inside each super node u, T_u, can be decomposed into a minimal connected component of a perfect phylogeny connecting all terminal vertices T_u^r, which we call a *core tree.* The edges in $T_u \setminus T_u^r$ are called *peripheral* edges. An interface edge (defined in Section 5.0.6) has to be a peripheral edge. The core tree is the unique perfect phylogeny on the taxa that have been assigned to the super node. We now consider the optimal phylogeny T_{opt} with the following additional property: no super node u (except the root) contains a degree two vertex that is adjacent to both a peripheral edge and the vertex cover edge $(u, p(u))$. The next lemma establishes that such an optimal tree exists:

Lemma 5.21 *If there exists a q-near-perfect phylogeny, then there exists an optimal phylogeny T_{opt} that contains no vertex v in super node u st degree(v) = 2 and v is adjacent to both a peripheral edge and the vertex cover edge $(u, p(u))$.*

Proof: Assume for the purposes of contradiction that no such phylogeny exists. Then consider the optimal phylogeny T_{opt} that contains the minimum number of vertices v with the above property. One edge adjacent to v is the vertex cover edge say e_v, with say $\mu(e_v) = k$. The second edge connects to a vertex v' inside super node u and let $\mu(v, v') = l$. Now we can flip the ordering of mutations of e_v and (v, v') such that $\mu(e_v) = l$ and $\mu(v, v') = k$ to obtain a new phylogeny T'_{opt}. Note that this is just a trivial operation of transform $\rho_{k,l}$ on steiner vertex v. It is easy to see that the only pair of characters in v of T'_{opt} that could contain a new gamete not present in the original phylogeny T_{opt} contains one character of the vertex cover. Therefore in T'_{opt}, all the vertices are good (assuming they were in the original phylogeny). The number of vertices in the super node $p(u)$ increased by one and the number of super nodes in u decreased by one since v was previously part of u and now is part of $p(u)$. Therefore the number of vertices with the property given in the claim decreased by one, a contradiction to the assumption. $\square$

Correctness: In $s(T_{opt}, Q)$, consider a super node S_0, that has children super nodes $S_1, \cdots, S_k$. Since S is assumed to be identical to $s(T_{opt}, Q)$, we use S_i to refer to the super node of both the skeleton S and $s(T_{opt}, Q)$. Let T_i be sub-phylogeny present inside the super node S_i in T_{opt}. Let T'_i be the sub-phylogeny of T_{opt}, obtained by removing the vertex cover edge (S_i, S_0), rooted at a vertex in S_i. Assume inductively that the subtrees $T_1, \ldots, T_k$ corresponding to the supernodes $S_1, \ldots, S_k$ have been built as in T_{opt}. Now we proceed to construct the tree inside super node S_0. This amounts to building the core tree and finding the peripheral edges (used in connecting the core tree to S_i) of T_0. Using Corollary 5.20 and Lemma 5.18 we know that $|P_i| \leq q$. We now make the following claim:

Claim 5.22 *Only the selected vertices in P_i can be vertices of T_{opt} that connects S_i to S_0.*

Proof: There exists exactly one edge $e = (v_0, v_1)$, s.t. $\mu(e) = c$ in S_i. The edge e partitions S_i into S_i^0 and S_i^1 based on the value on the character c and assume $v_0 \in S_i^0$ and $v_1 \in S_i^1$. For the sake of contradiction assume that in T_{opt}, a vertex v from S_i^1 connects to S_0 via the vertex cover edge. Clearly every path to a terminal vertex in $T_{opt} \backslash S'_i$ from v contains an edge $e' = (v'_1, v'_0)$ with $\mu(e') = c$.

Consider the edges e' that is the first mutation of character c in every path from e to a terminal vertex in $T_{opt} \backslash T'_i$. Let M be the set of characters that mutate in the path connecting v_0 and v. Since there are no bad vertices, all mutations of characters except vertex cover between e and e' occur even numbers of times. Therefore, specifically v_0 and v'_0 are identical in all the characters in M. Therefore, connecting v_0 instead of v to u via vertex cover and deleting all mutations of M that occurs in the path between e and e' results in a phylogeny of smaller cost, a contradiction to optimality. □

We continue by finding sets P_i for each of S_i. Now, using exhaustive search, we select a vertex from each P_i. One would be the correct set of vertices as used in T_{opt}. For each v selected from P_i, we mutate the character $\mu(S_i, S_0)$ and add to S_0. Since we guessed correctly, each vertex added to S_0 should be present in T_0 of T_{opt}. To identify all the peripheral edges, we simply grow the unique perfect phylogeny in S_0, which we know exists from Lemma 5.18. It follows from Lemma 5.21 that we have now completely identified the tree in S_0. The correctness follows inductively. This therefore completes the description and correctness of the BNPP algorithm.

5.1 Evaluating Running Time

The running time for pre-preprocessing and constructing the conflict graph is $O(nm^2)$. The running time for function partition can be bounded as follows. The recursion tree has height q, and any function call to partition makes at most $2(q+1)$ recursive calls. Therefore the number of leaves of the recursion tree is bounded by $2^q(q+1)^q$. Each leaf of the recursion tree contains at most q characters in its list *singleMutations*. Expanding the skeleton, Step 1b can be naively implemented by once again enumerating all skeletons S' using the characters of Q and *singleMutations* together. We can then discard any trees S' from which S cannot be obtained by deleting edges of *singleMutations* from S' in time $q^{O(q)}$. Step 3a can be implemented similarly. Since the number of trees with $2q$ vertices is bounded by $2^{O(q)}q^{O(q)}$, the total time spent by partition function is bounded by $2^{O(q)}q^{O(q)}nm$.

We can similarly bound the time spent in function use-interface. As already analyzed in the correctness, the depth of the recursion tree and therefore the size of the skeletons examined is $O(q)$.

Addition of an edge splits a super node into two new super nodes. The number of possible resulting skeletons for an addition of a single edge on a specific super node of degree g in the skeleton, is at most 2^g. Therefore, the total number of skeletons examined in Step 3a (and hence the degree of a node in the recursion tree) is at most $2^{O(q)}q^{O(q)}$. A naive analysis gives a bound of $2^{O(q^2)}q^{O(q^2)}$ number of calls to the function `use-interface`. This however is a weak bound. It is easy to see that if the skeleton S currently has penalty q' then character h of function `use-interface` can mutate at most $q - q' + 1$ times. Therefore, the time spent in executing function `use-interface` on a skeleton with penalty $q - p$ and current depth d on the recursion tree can be bounded as: $T(p, d) \leq O(nm + \sum_{i=0}^{p} \binom{cq}{i+1} 2^{2i} T(p - i, d - 1)), T(x, 0) = 1, T(0, y) = cq$, where cq is an upper bound on the size of any skeleton explored, $0 \leq x, 1 \leq y$. The term $\binom{cq}{i+1}$ comes from placing $i + 1$ mutations of character h in the super nodes of the current skeleton. Since super nodes contain perfect phylogenies, a super node can not have two mutations of character h. Term 2^{2i} is for guessing *mark* values of the newly created super nodes. After adding $i + 1$ mutations, the penalty of the skeleton increases by i and therefore the skeleton passed into the recursive call has penalty $q - p + i$, and the recursive call increases the depth by 1. The recursion can be bounded by $T(q, q) \leq (cq)^{O(q)}nm$ since the maximum depth and maximum penalty is q.

The running time for `linktrees` can be bounded as follows. Recall that we used S_0 to denote a super node that had children $\{S_0, \cdots, S_k\}$ in $s(T_{opt}, Q)$. The function guesses one vertex from each of S_i and adds it to S_0. This implies that the time taken to exhaustively search for all possible vertices to add into S_0 is $O(\prod_i |P_i|)$. From the proof of correctness, $|P_i| \leq q$ and the number of super nodes in S at any point is at most cq. Therefore, the time taken to process the entire subtree of S_0, $T(S_0) \leq O(nm + q^k \prod_i T(S_i)), T(L) = 1$, where L is any leaf super node. This recursion can be solved as: $T(R_S) \leq O(nmq^{cq})$, where R_S is the root supernode. Putting together the preceding analyses leads to our primary result:

Theorem 5.23 *The algorithm solves the BNPP problem in* $q^{O(q)}nm + O(nm^2)$ *time.*

6 Experiments

We implemented a version of the simpler $(q + \kappa)^{O(q)}nm + O(nm^2)$ algorithm and applied to it datasets of non-recombining DNA sequences. In such sequences, the most likely explanation for a pair of characters exhibiting all four gametes is recurrent mutations. We find as a common theme that κ is nearly the same size of q. Since this was the case, we simply enumerate all skeletons by brute-force at step 3 of function `buildNPP`, a modification which would not affect correctness of the method.

Data Sources

In humans, autosomal chromosomes are not typically used for phylogenetic analysis because of the hardness of accounting for recombination events in phylogenetic trees [24]. The two most frequently used sources of genetic variation are therefore mitochondrial DNA (mtDNA) and the Y chromosome. The hyper-variable segments of the mitochondrial DNA (HVS I and II) are of particular use when analyzing population groups that diverged more recently, due to their high mutation rates. We use examples of both mtDNA and Y chromosome DNA for empirical validation.

We first analyzed data from the International HapMap Consortium [15]. The HapMap contains SNPs typed in four population groups: CEPH (Utah Residents with Northern and Western European ancestry); Han Chinese in Beijing, China; Japanese in Tokyo, Japan; and Yoruba in Ibadan, Nigeria. The SNPs typed varied from one population to the other. We used all the males typed by the HapMap project on the SNP loci that were common to all the four populations. This resulted in 150 sequences typed at 49 SNP sites. Missing SNP allele values were replaced with the major alleles for the corresponding loci.

Our second source of data is from a study conducted by Stone et al. [22] on the relationship between humans and our closest living relatives, the chimpanzee (*Pan troglodytes*) and the bonobo (*Pan paniscus*). The main contribution of their work was analyzing sequence diversity of the Y chromosome. The Y chromosome data was composed of humans (1), gorilla (1), *P. troglodytes troglodytes* (5), *P. paniscus* (3), *P. troglodytes verus* (2), *P. troglodytes* of unknown sub-species (2) and *P. troglodytes schweifurthii* (1), for a total of 15 taxa. The sequences had 98 polymorphic sites of which 10 were indels and the remaining were SNPs. As in Stone et al., we treated indel mutations as character changes, leaving exclusively binary markers.

We also use *P. troglodytes* mtDNA, which was also gathered by Stone et al. and was published as supplementary data to the same paper. The mtDNA consisted of 25 taxa: *P. troglodytes troglodytes* (16), *P. troglodytes schweifurthii* (5) and *P. troglodytes verus* (4). The sequences were typed at 1041 loci, presenting us with a much larger input than in the other examples.

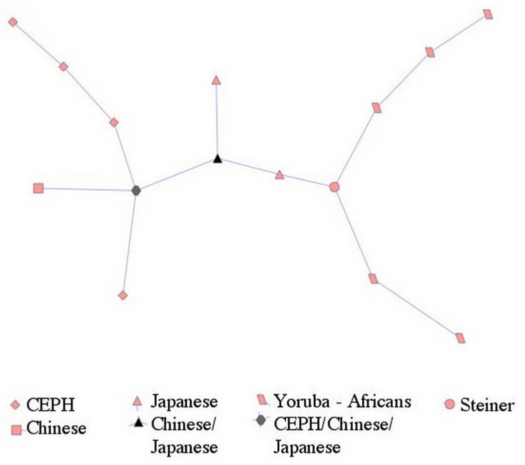
Results

We derived optimal phylogenetic trees for all the input data. In all three cases, the algorithm completed within seconds. For the HapMap data set, the algorithm finds a 1-near perfect phylogeny (Figure 13 (a)). The phylogeny partitions all haplotypes by population sample, with haplotypes found in multiple population groups occurring at the boundaries between the population-specific haplotypes. The result therefore supports the validity of the parsimony metric for these data.

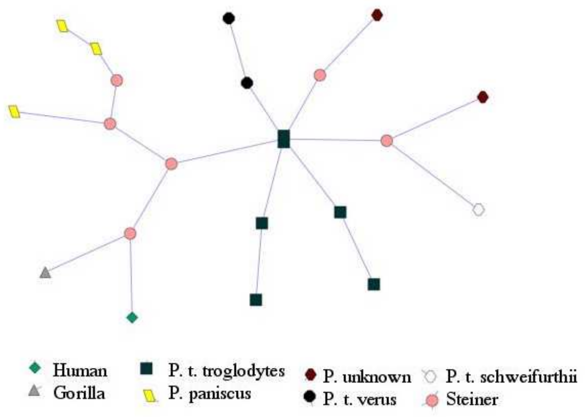
Figure 13(b) shows a 1-near-perfect phylogeny inferred from Stone et al. Y chromosome data [22]. Distinct population groups are again perfectly partitioned into distinct subtrees. Our optimal solution is identical to that inferred by Stone et al. They, however, used branch-and-bound to infer the optimal phylogeny, whereas our algorithm finds the solution without a worst-case exponential time search.

Figure 13(c) shows an optimal 2-near-perfect phylogeny inferred from the Stone et al. chimpanzee mtDNA data. The total number of base changes in our phylogeny (called the *parsimony score*) is 63. The data did not partition as neatly by subspecies as did the Y-chromosome data, perhaps suggesting lesser validity to the parsimony assumption for this data set.

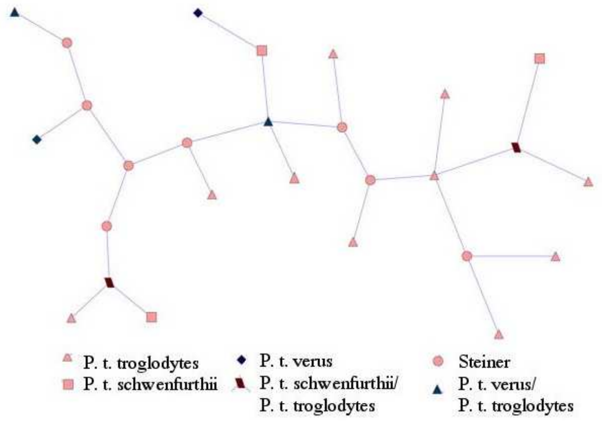
For comparison, we applied the *pars* program for phylogenetic tree inference from the `PHYLIP` software package [8] to all three data sets. The program with default parameters finds the optimal solution as does our algorithm for the two 1-near-perfect Y-chromosome samples. For the mtDNA data, though, *pars* produces tree with parsimony score 254, substantially worse than our provably optimal result of 63. These results suggest that while heuristic methods in common use may often produce optimal or near-optimal trees, they can do much worse in practice on at least some real datasets.



(a)



(b)



P. t. troglodytes
P. t. schwenfurthii
P. t. verus
P. t. schwenfurthii/ P. t. troglodytes
Steiner
P. t. verus/ P. t. troglodytes

(c)

Figure 13: Near-perfect phylogenies inferred for real testing data. Individuals with identical haplotypes are collapsed into single nodes. Single edge could represent multiple character mutations. Nodes are labeled according to the population groups (HapMap data) or the species (Stone et al. data) (a): 1-near-perfect phylogeny constructed from the HapMap Y chromosome data [15]. (b): 1-near-perfect phylogeny inferred from primate Y-chromosome DNA [22]. (c): 2-near-perfect phylogeny inferred from the chimpanzee mtDNA [22]

7 Conclusion

We have presented two algorithms for inferring optimal near-perfect binary phylogenies. This problem is of considerable practical interest for evolutionary tree reconstruction from SNP data. One algorithm provides a simple, practical method for optimally finding near-perfect phylogenies. The other proves the problem fixed-parameter tractable by showing it to be solvable in time $q^{O(q)}nm + O(nm^2)$. Empirical results on the first method show it can quickly reconstruct optimal trees for near-perfect real data sets, at least occasionally substantially outperforming a commonly used heuristic method. Our methods may have practical value not just for mitochondrial and Y-chromosome data, but for such applications as tracing evolution of non-eukaryotic organisms or explaining recombination-free regions of autosomal eukaryotic chromosomes. It remains to be seen what values of q can be made computationally feasible in practice and how often values permitting provable optimization are found in real data sets of interest.

References

[1] R. Agarwala and D. Fernandez-Baca. A Polynomial-Time Algorithm for the Perfect Phylogeny Problem when the Number of Character States is Fixed. In: *SIAM Journal on Computing*, 23 (1994). pp 1216-1224.

[2] H. Bodlaender, M. Fellows and T. Warnow. Two Strikes Against Perfect Phylogeny. In *proc. 19th International Colloquium on Automata, Languages and Programming, LNCS*, (1992). pp 273-283.

[3] H. Bodlaender, M. Fellows, M. Hallett, H. Wareham and T. Warnow. The Hardness of Perfect Phylogeny, Feasible Register Assignment and Other Problems on Thin Colored Graphs. In *Theoretical Computer Science*, (2000). pp 167-188.

[4] M. Bonet, M. Steel, T. Warnow and S. Yooseph. Better Methods for Solving Parsimony and Compatibility. In: *Journal of Computational Biology*, 5(3), (1992). pp 409-422.

[5] T. Cormen, C. Leiserson, R. Rivest and C. Stein. Introduction to Algorithms. *The MIT press* 2001.

[6] W. H. Day and D. Sankoff. Computational Complexity of Inferring Phylogenies by Compatibility. *Systematic Zoology*, (1986). pp 224-229.

[7] E. Eskin, E. Halperin and R. M. Karp. Efficient Reconstruction of Haplotype Structure via Perfect Phylogeny. In *JBCB* 2003. pp 1-20.

[8] J. Felsenstein. PHYLIP (Phylogeny Inference Package) version 3.6. Distributed by the author. Department of Genome Sciences, University of Washington, Seattle. (2005).

[9] D. Fernandez-Baca and J. Lagergren. A Polynomial-Time Algorithm for Near-Perfect Phylogeny. In: *SIAM Journal on Computing*, 32 (2003). pp 1115-1127.

[10] L. R. Foulds and R. L. Graham. The Steiner problem in Phylogeny is NP-complete. In: *Advances in Applied Mathematics* (3), (1982). pp 4

[11] G. Ganapathy, V. Ramachandran and T. Warnow. Better Hill-Climbing Searches for Parsimony. In: *WABI* (2003).

[12] D. Gusfield. Efficient Algorithms for Inferring Evolutionary Trees. In: *Networks*, 21 (1991). pp 19-28.

[13] D. Gusfield. Algorithms on Strings, Trees and Sequences. *Cambridge University Press*, 1999.

[14] D. Gusfield, S. Eddhu and C. Langley. Efficient Reconstruction of Phylogenetic Networks with Constrained Recombination. In Proc *IEEE Computer Soc. Bioinformatics Conf* (2003). pp. 363-374.

[15] The International HapMap Consortium. The International HapMap Project. *Nature* 426 (2003), pp. 789-796.

[16] The International Human Genome Sequencing Consortium. Initial Sequencing and Analysis of the Human Genome. In *Nature*, 409 (2001). pp 860-921.

[17] S. Kannan and T. Warnow. A Fast Algorithm for the Computation and Enumeration of Perfect Phylogenies. In *SIAM Journal on Computing*, 26 (1997). pp 1749-1763.

[18] Oleg Pikhurko. Generating Edge Labelled Trees. To appear *American Math Monthly* 3pp.

[19] Srinath Sridhar, Kedar Dhamdhere, Guy E. Blelloch, Eran Halperin, R. Ravi and Russell Schwartz. FPT Algorithms for Binary Near Perfect Phylogenetic Trees. Technical Report *TR CMU-CS-05-181*. http://reports-archive.adm.cs.cmu.edu/cs2005.html, also available from the authors at: www.cs.cmu.edu/~srinath/recomb06.pdf

[20] M. A. Steel. The Complexity of Reconstructing Trees from Qualitative Characters and Subtrees. In *J. Classification*, 9 (1992). pp 91-116.

[21] S. T. Sherry, M. H. Ward, M. Kholodov, J. Baker, L. Pham, E. Smigielski, and K. Sirotkin. dbSNP: The NCBI Database of Genetic Variation. In *Nucleic Acids Research*, 29 (2001). pp 308-311.

[22] A. C. Stone, R. C. Griffiths, S. L. Zegura, and M. F. Hammer. High levels of Y-chromosome nucleotide diversity in the genus Pan. In *Proceedings of the National Academy of Sciences* (2002). pp 43-48.

[23] J. C. Venter, M. D. Adams, E. W. Myers, et al. The sequence of the human genome. In *Science*, 291 (2001), pp 1304-1351.

[24] L. Wang, K. Zhang, and L. Zhang. Perfect Phylogentic Networks with Recombination. In *Journal of Computational Biology*, (2001). pp 69-78.

[25] D. L. Wheeler, T. Barret, D. A. Benson, et al. Database resources for the National Center for Biotechnology Information. In *Nucleic Acids Research*, 33 (2005). pp D39-D45.