

# Algorithms for Efficient Near-Perfect Phylogenetic Tree Reconstruction in Theory and Practice

Srinath Sridhar, Kedar Dhamdhere, Guy E. Blelloch, Eran Halperin, R. Ravi, and Russell Schwartz

**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. We develop two algorithms for constructing optimal near-perfect phylogenies and provide empirical evidence of their performance. The first simple algorithm is fixed-parameter tractable when the number of additional mutations and the number of characters that share four gametes with some other character are constants. The second, more involved, algorithm for the problem is fixed-parameter tractable when only the number of additional mutations is fixed. We have implemented both algorithms and have shown them to be extremely efficient in practice on biologically significant data sets. This work proves that the BNPP problem is fixed-parameter tractable and provides the first practical phylogenetic tree reconstruction algorithms that find guaranteed optimal solutions while being easily implemented and computationally feasible for data sets of biologically meaningful size and complexity.

**Index Terms**—Computations on discrete structures, trees, biology and genetics.

## 1 INTRODUCTION

THE reconstruction of evolutionary trees is a classical computational biology problem [15], [24]. In the maximum parsimony (MP) model of this problem, one seeks the smallest 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 have made it possible to identify millions of single nucleotide polymorphisms (SNPs) [26], sites at which a single DNA base takes on two common variants.

Minimizing the length of a phylogeny is the problem of finding the most parsimonious tree, a well-known NP-complete problem [12]. Researchers have thus focused on

either sophisticated heuristics or solving optimally for special cases (for example, fixed-parameter variants [1], [8], [20]). Previous attempts at such solutions for the general parsimony problem have only produced theoretical results, yielding algorithms too complicated for practical implementation. A large number of related works have been published, but it is impossible to mention all of them here.

In this work, we focus on the case when the set of character states is binary. In this setting, the input is often represented as an  $n \times m$  matrix  $I$ . The  $n$  rows of the matrix (taxa) can be viewed as points on an  $m$ -cube. Therefore, the problem is equivalent to finding the Steiner minimum tree in a hypercube. In the binary-state case, a phylogeny is called *perfect* if its length equals  $m$ . Gusfield showed that such phylogenies can be reconstructed in linear time [14].

If there exists no perfect phylogeny for input  $I$ , then one option is to slightly modify  $I$  so that a perfect phylogeny can be constructed for the resulting input. Upper bounds and negative results have been established for such problems. For instance, Day and Sankoff [7] showed that finding the maximum subset of characters containing a perfect phylogeny is NP-complete, whereas Damaschke [8] showed fixed-parameter tractability for the same problem. The problem of reconstructing the most parsimonious tree without modifying the input  $I$  seems significantly harder.

Fernandez-Baca and Lagergren recently considered the problem of reconstructing optimal near-perfect phylogenies [11], 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

- S. Sridhar and G.E. Blelloch are with the Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. E-mail: {sridhar, blelloch}@cs.cmu.edu.
- K. Dhamdhere is with Google Inc., Mountain View, CA 94043. E-mail: kedar.dhamdhere@gmail.com.
- E. Halperin is with the International Computer Science Institute, University of California, Berkeley, Berkeley, CA 94704. E-mail: heran@icsi.berkeley.edu.
- R. Ravi is with the Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213. E-mail: ravi@cmu.edu.
- R. Schwartz is with the Department of Biological Sciences, Carnegie Mellon University, 4400 Fifth Ave., Pittsburgh, PA 15213. E-mail: russells@andrew.cmu.edu.

Manuscript received 18 July 2006; accepted 17 Jan. 2007; published online 6 Mar. 2007.

For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBBSI-0145-0706. Digital Object Identifier no. 10.1109/TCBB.2007.1070.

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).

### 1.1 Our Work

*Algorithm 1.* We first present theoretical and practical results on the optimal solution of the BNPP problem. We completely describe and analyze an intuitive algorithm for the BNPP problem that has a runtime of  $O((72\kappa)^q nm + nm^2)$ , where  $\kappa$  is the number of characters that violate the *four-gamete* condition, a test of perfectness of a data set explained below. Since  $\kappa \leq m$ , this result significantly improves the prior runtime. Furthermore, the complexity of the previous work would make its practical implementation daunting; to our knowledge, no implementation of it has ever been attempted. Our results thus describe the first practical phylogenetic tree reconstruction algorithm that finds guaranteed optimal solutions while being computationally feasible for data sets of biologically relevant complexity. A preliminary paper on this algorithm appeared in [23].

*Algorithm 2.* We then present a more involved algorithm that runs in time  $O(21^q + 8^q nm^2)$ . Fernandez-Baca and Lagergren [11] in concluding remarks state that the most important open problem in the area is to develop a parameterized algorithm or prove  $W[t]$  hardness for the near-perfect phylogeny problem. We make progress on this open problem by showing for the first time that BNPP is fixed-parameter tractable (FPT). To achieve this, we use a divide-and-conquer algorithm. Each divide step involves performing a “guess” (or enumeration) with cost exponential in  $q$ . Finding the Steiner minimum tree on a  $q$ -cube dominates the runtime when the algorithm bottoms out. The present work substantially improves on the time bounds derived for a preliminary version of this algorithm, which was first presented by Bleloch et al. in [2].

We further implement variants of both algorithms and demonstrate them on a selection of real mitochondrial, Y-chromosome, and bacterial data sets. The results demonstrate that both algorithms substantially outperform their worst-case time bounds, yielding optimal trees with high efficiency on real data sets typical of those for which such algorithms would be used in practice.

## 2 PRELIMINARIES

In defining formal models for parsimony-based phylogeny construction, we borrow definitions and notations from a couple of previous works [11], [24]. The input to a phylogeny problem is an  $n \times m$  binary matrix  $I$ , where rows  $R(I)$  represent *input taxa* and are binary strings. The column numbers  $C = \{1, \dots, m\}$  are referred to as *characters*. In a *phylogenetic tree* or *phylogeny*, each vertex  $v$  corresponds to a taxon (not necessarily in the input) and has an associated label  $l(v) \in \{0, 1\}^m$ .

The following are equivalent definitions of a phylogeny, both of which have been used in prior literature:

**Definition 1.** A phylogeny  $T$  for matrix  $I$  is:

1. A tree  $T(V, E)$  with the following properties:  $R(I) \subseteq l(V(T))$  and, for all  $(u, v) \in E(T)$ ,  $H(l(u), l(v)) = 1$ , where  $H$  is the Hamming distance.
2. A tree  $T(V, E)$  with the following properties:  $R(I) \subseteq l(V(T))$  and  $l(\{v \in V(T) \mid \text{degree}(v) \leq 2\}) \subseteq R(I)$ . That is, every input taxon appears in  $T$  and every leaf or degree-2 vertex is an input taxon.

The following two definitions provide some useful terminology when discussing either definition of a phylogeny:

**Definition 2.** A vertex  $v$  of phylogeny  $T$  is terminal if  $l(v) \in R(I)$  and Steiner otherwise.

**Definition 3.** For a phylogeny  $T$ ,

$$\text{length}(T) = \sum_{(u,v) \in E(T)} d(l(u), l(v)),$$

where  $d$  is the Hamming distance.

A phylogeny is called an optimum phylogeny if its length is minimized. We will assume that both states, 0 and 1, are present in all characters. Therefore, the length of an optimum phylogeny is at least  $m$ . This leads to the following two definitions:

**Definition 4.** For a phylogeny  $T$  on input  $I$ ,

$$\text{penalty}(T) = \text{length}(T) - m; \text{penalty}(I) = \text{penalty}(T^{\text{opt}}),$$

where  $T^{\text{opt}}$  is any optimum phylogeny on  $I$ .

**Definition 5.** A phylogeny  $T$  is called  $q$ -near-perfect if  $\text{penalty}(T) = q$  and perfect if  $\text{penalty}(T) = 0$ .

Note that, in an optimum phylogeny, no two vertices share the same label. Therefore, we can equivalently define an edge of a phylogeny as  $(t_1, t_2)$ , where  $t_i \in \{0, 1\}^m$ . Since we will always be dealing with optimum phylogenies, we will drop the label function  $l(v)$  and use  $v$  to refer to both a vertex and the taxon it represents in a phylogeny.

With the above definitions, we are now prepared to define our central computational problem.

### 2.1 The BNPP Problem

Given an integer  $q$  and an  $n \times m$  binary input matrix  $I$ , if  $\text{penalty}(I) \leq q$ , then return an optimum phylogeny  $T$ ; else, declare NIL. The problem is equivalent to finding the minimum Steiner tree on an  $m$ -cube if the optimum tree is at most  $q$  larger than the number of dimensions  $m$  or declaring NIL otherwise. The problem is fundamental and therefore expected to have diverse applications besides phylogenies.

**Definition 6.** We define the following notations:

- $r[i] \in \{0, 1\}$ : The state in character  $i$  of taxon  $r$ .
- $\mu(e) : E(T) \rightarrow 2^C$ : The set of all characters corresponding to edge  $e = (u, v)$  with the property for any  $i \in \mu(e)$ ,  $u[i] \neq v[i]$ . Note that, for the first definition of a phylogeny,  $\mu(e) : E(T) \rightarrow C$ .
- For a set of taxa  $M$ , we use  $T_M^*$  to denote an optimum phylogeny on  $M$ .

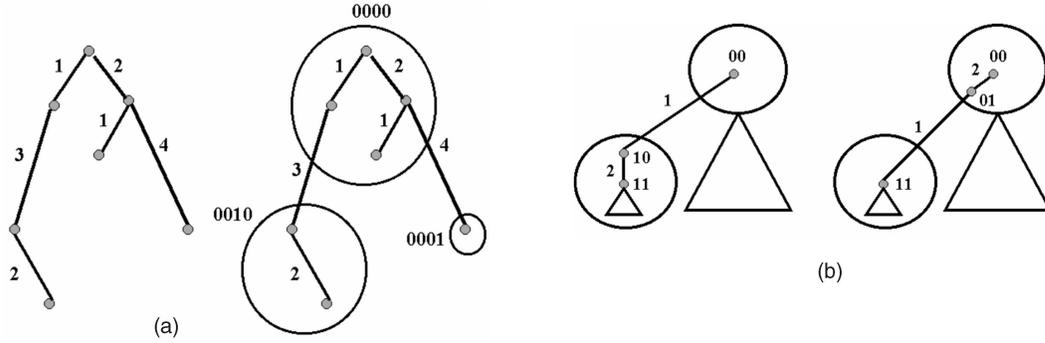


Fig. 1. (a) Phylogeny  $T$  and skeleton  $s(T, C')$ ,  $C' = \{3, 4\}$ . Edges are labeled with characters that mutate  $\mu$  and supernodes with tags  $t$ . (b) Transform to remove a degree-2 Steiner root from a supernode. Note that the size of the phylogeny is unchanged.

We say that an edge  $e$  mutates character  $i$  if  $i \in \mu(e)$ . We will use the following well-known definition and lemma on phylogenies.

**Definition 7.** Given matrix  $I$ , the set of gametes  $G_{i,j}$  for characters  $i$  and  $j$  is defined as  $G_{i,j} = \{(r[i], r[j]) | r \in R(I)\}$ . Two characters  $i$  and  $j$  share  $t$  gametes in  $I$  iff  $|G_{i,j}| = t$ .

In other words, the set of gametes  $G_{i,j}$  is a projection on the  $i$  and  $j$  dimensions.

**Lemma 2.1 [14].** An optimum phylogeny for input  $I$  is not perfect iff there exist two characters,  $i$  and  $j$ , that share (all) four gametes in  $I$ .

**Definition 8 (Conflict Graph [17]).** A conflict graph  $G$  for matrix  $I$  with character set  $C$  is defined as follows: Every vertex  $v$  of  $G$  corresponds to unique character  $c(v) \in C$ . An edge  $(u, v)$  is added to  $G$  iff  $c(u), c(v)$  share all four gametes in  $I$ . Such a pair of characters is defined to be in conflict.

Note that if the conflict graph  $G$  contains no edges, then a perfect phylogeny can be constructed for  $I$ . Gusfield [14] provided an efficient algorithm to reconstruct a perfect phylogeny in such cases.

## 2.2 Simplifications

We assume that the all-zero taxon is present in the input. If not, using our freedom of labeling, we convert the data into an equivalent input containing the all-zero taxon (see the work of Eskin et al. [9, Section 2.2] for details). We also 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. The BNPP problem asks for the reconstruction of an unrooted tree. For the sake of analysis, we will, however, assume that all of the phylogenies are rooted at the all-zero taxon.

## 3 SIMPLE ALGORITHM

This section describes a simple algorithm for the reconstruction of a binary near-perfect phylogenetic tree. Throughout this section, we will use the first definition of a phylogeny (Definition 1).

We begin by performing the following preprocessing step: For every pair of characters  $c', c''$ , if  $|G_{c',c''}| = 2$ , we (arbitrarily) remove character  $c''$ . After repeatedly performing the above step, we have the following lemma:

**Lemma 3.1.** For every pair of characters  $c', c''$ ,  $|G_{c',c''}| \geq 3$ .

We will assume that the above lemma holds on the input matrix for the rest of the paper. Note that such characters  $c', c''$  are identical (after possibly relabeling one character) and are usually referred to as noninformative. It is not hard to show that this preprocessing step does not change the correctness or runtime of our algorithm.

The following additional definitions are required for the description and analysis of the simple algorithm:

**Definition 9.** For any phylogeny  $T$  and set of characters  $C' \subseteq C$ :

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

**Definition 10.** A tag  $t(u) \in \{0, 1\}^m$  of supernode  $u$  in  $s(T, C')$  has the property that  $t(u)[c'] = v[c']$  for all  $c' \in C'$ , vertices  $v \in u$ ;  $t(u)[i] = 0$  for all  $i \notin C'$ .

Throughout this paper, we will assume without loss of generality that we are working with phylogenies and skeletons that are rooted at the all-zero taxon and tag, respectively. Furthermore, the skeletons used in this work themselves form a perfect phylogeny in the sense that no character mutates more than once in the skeleton. Note that, in such skeletons, tag  $t(u)[i] = 1$  iff character  $i$  mutates exactly once in the path from the root to  $u$ . Fig. 1a shows an example of a skeleton of a phylogeny. We will use the term *subphylogeny* to refer to a subtree of a phylogeny.

Throughout the analysis, we fix an optimal phylogeny  $T^*$  and show that our algorithm finds it. We assume that both  $T_{opt}$  and its skeleton are rooted at the all-zero label and tag, respectively. The high-level idea of our algorithm is to first guess the characters that mutate more than once in  $T_{opt}$ . The algorithm then finds a perfect phylogeny on the remaining characters. Finally, it adds back the imperfect components by solving a Steiner tree problem. The algorithm is divided into two functions, `buildNPP` and `linkTrees`, whose pseudocodes are provided in Figs. 2 and 3.

Function `buildNPP` starts by determining the set of characters  $c(V_{nis})$  that corresponds to the nonisolated

```

function buildNPP ( binary matrix  $I$ , integer  $q$  )
1) let  $G(V, E)$  be the conflict graph of  $I$ 
2) let  $V_{nis} \subseteq V$  be the set of non-isolated vertices
3) for all  $M \in 2^{c(V_{nis})}$ ,  $|M| \leq q$ 
   a) construct rooted perfect phylogeny  $PP(V_{PP}, E_{PP})$  on
      characters  $C \setminus M$ 
   b) define  $\lambda : R \mapsto V_{PP}$  s.t.  $\lambda(r) = u$  i.f.f. for all  $i \in C \setminus M$ ,
       $r[i] = t(u)[i]$ 
   c)  $T_f := \text{linkTrees}(PP)$ 
   d) if  $\text{penalty}(T_f) \leq q$  then return  $T_f$ 
4) return NIL

```

Fig. 2. Pseudocode to find the skeleton.

vertices of the conflict graph in Step 2. From set  $c(V_{nis})$ , the algorithm then selects by brute force the set of characters  $M$  that mutate more than once in  $T_{opt}$ . Only characters corresponding to nonisolated vertices can mutate more than once in any optimal phylogeny (a simple proof follows from Buneman graphs [24]). Since all characters of  $C \setminus M$  mutate exactly once, the algorithm constructs a perfect phylogeny on this character set using Gusfield's linear time algorithm [14]. The perfect phylogeny is unique because of Lemma 3.1. Note that  $PP$  is the skeleton  $s(T_{opt}, C \setminus M)$ . Since the tags of the skeleton are unique, the algorithm can now determine the supernode where every taxon resides as defined by function  $\lambda$  in Step 3b. This rooted skeleton  $PP$  is then passed into function `linkTrees` to complete the phylogeny.

Function `linkTrees` takes a rooted skeleton  $Sk$  (sub-skeleton of  $PP$ ) as argument and returns a tuple  $(r, c)$ . The goal of function `linkTrees` is to convert skeleton  $Sk$  into a phylogeny for the taxa that reside in  $Sk$  by adding edges that mutate  $M$ . Notice that, by using function  $\lambda$ , we know the set of taxa that reside in skeleton  $Sk$ . The phylogeny for  $Sk$  is built bottom-up by first solving the phylogenies on the subskeleton rooted at the children supernodes of  $Sk$ . Tuple  $(r, c)$  returned by the function call to `linkTrees`( $Sk$ ) represents the cost  $c$  of the optimal phylogeny when the label of the root vertex in the root supernode of  $Sk$  is  $r$ . Let  $S = \text{root}(Sk)$  represent the root supernode of skeleton  $Sk$ .  $R_S$  is the set of input taxa that map to supernode  $S$  under function  $\lambda$ . Let its children supernodes be  $S_1, S_2, \dots$ . Assume that recursive calls to `linkTrees`( $S_i$ ) return  $(r_i, c_i)$ . Notice that the parents of the set of roots  $r_i$  all reside in supernode  $S$ . The parents of  $r_i$  are denoted by  $p_i$  and are identical to  $r_i$  except in the character that mutates in the edge connecting  $S_i$  to  $S$ . Set  $\tau$  is the union of  $p_i$  and  $R_S$  and forms the set of vertices inferred to be in  $S$ . Set  $D$  is the set of characters on which the labels of  $\tau$  differ, that is, for all  $i \in D$ ,  $\exists r_1, r_2 \in \tau$ ,  $r_1[i] \neq r_2[i]$ . In Step 9, we guess the root  $r_S$  of supernode  $S$ . This guess is "correct" if it is identical to the label of the root vertex of  $S$  in  $T_{opt}$ . Notice that we are only guessing  $|D|$  bits of  $r_S$ . Corollary 3.3 of Lemma 3.2, along with optimality, requires that the label of the root vertex of  $T_{opt}$  be identical to  $\tau$  in all the characters  $C \setminus D$ .

**Lemma 3.2.** *There exists an optimal phylogeny  $T_{opt}$  that does not contain any degree 2 Steiner roots in any supernode.*

```

function linkTrees ( skeleton  $Sk(V_s, E_s)$  )
1) let  $S := \text{root}(Sk)$ 
2) let  $R_S := \{s \in R \mid \lambda(s) = S\}$ 
3) for all children  $S_i$  of  $S$ 
   a) let  $Sk_i$  be subtree of  $Sk$  rooted at  $S_i$ 
   b)  $(r_i, c_i) := \text{linkTrees}(Sk_i)$ 
4) let  $\text{cost} := \sum_i c_i$ 
5) for all  $i$ , let  $l_i := \mu(S, c_i)$ 
6) for all  $i$ , define  $p_i \in \{0, 1\}^m$  s.t.  $p_i[l_i] \neq r_i[l_i]$  and for all
    $j \neq l_i$ ,  $p_i[j] = r_i[j]$ 
7) let  $\tau := R_S \cup (\cup_i \{p_i\})$ 
8) let  $D \subseteq C$  be the set of characters where taxa in  $\tau$  differ
9) guess root taxon of  $S$ ,  $r_S \in \{0, 1\}^m$  s.t.  $\forall i \in C \setminus D, \forall u \in \tau$ ,
    $r_S[i] = u[i]$ 
10) let  $c_S$  be the size of the optimal Steiner tree of  $\tau \cup \{r_S\}$ 
11) return  $(r_S, \text{cost} + c_S)$ 

```

Fig. 3. Pseudocode to construct and link imperfect phylogenies.

**Proof.** Fig. 1b shows how to transform a phylogeny that violates the property into one that does not. Root 10 is a degree 2 Steiner and is moved into a parent supernode as 01. Since 10 was a Steiner, the transformed tree contains all input.  $\square$

**Corollary 3.3.** *In  $T_{opt}$ , the LCA of the set  $\tau$  is the root of supernode  $S$ .*

In Step 10, the algorithm finds the cost of the optimum Steiner tree for the terminal set of taxa  $\tau \cup \{r_S\}$ . We use the Dreyfus-Wagner recursion [22] to compute this minimum Steiner tree. The function now returns  $r_S$ , along with the cost of the phylogeny rooted in  $S$ , which is obtained by adding the cost of the optimum Steiner tree in  $S$  to the cost of the phylogenies rooted at  $c_i$ . The following lemma bounds the runtime of our algorithm and completes the analysis:

**Lemma 3.4.** *The algorithm described above runs in time  $O((18\kappa)^q nm + nm^2)$  and solves the BNPP problem with a probability of at least  $2^{-2q}$ . The algorithm can be easily derandomized to run in time  $O((72\kappa)^q nm + nm^2)$ .*

**Proof.** The probability of a correct guess at Step 9 of function `linkTrees` is exactly  $2^{-|D|}$ . Notice that the Steiner tree in supernode  $S$  has at least  $|D|$  edges. Since  $\text{penalty}(T_{opt}) \leq q$ , we know that there are at most  $2q$  edges that can be added in all of the recursive calls to `linkTrees`. Therefore, the probability that all guesses at Step 9 are correct is at least  $2^{-2q}$ . The time to construct the optimum Steiner tree in Step 10 is  $O(3^{|\tau|} 2^{|D|})$ . Assuming that all guesses are correct, the total time spent in Step 10 over all recursive calls is  $O(3^{2q} 2^q)$ . Therefore, the overall runtime of the randomized algorithm is  $O((18\kappa)^q nm + nm^2)$ . To implement the randomized algorithm, since we do not know if the guesses are correct, we can simply run the algorithm for the above time and, if we do not have a solution, then we restart. Although presented as a randomized

```

buildNPP(input matrix  $I$ )
1) let  $L := \{I\}, E := \emptyset$ 
2) while  $|\cup_{M_i \in L} N(M_i)| > q$ 
    a) guess vertex  $v$  from  $\cup_{M_i \in L} N(M_i)$ , let
        $v \in N(M_j)$ 
    b) let  $M0 := M_j(c(v), 0)$  and  $M1 := M_j(c(v), 1)$ 
    c) guess taxa  $r$  and  $p$ 
    d) add  $r$  to  $M1$ ,  $p$  to  $M0$  and  $(r, p)$  to  $E$ 
    e) remove  $M_j$  from  $L$ , add  $M0$  and  $M1$  to  $L$ 
3) for each  $M_i \in L$  compute an optimum phylogeny
    $T_i$ 
4) return  $E \cup (\cup_i T_i)$ 

```

Fig. 4. Pseudocode to solve the BNPP problem. For all  $M_i \in L$ ,  $N(M_i)$  is the set of nonisolated vertices in the conflict graph of  $M_i$ . The guess at Step 2a is correct iff there exists  $T_{M_j}^*$ , where  $c(v)$  mutates exactly once. The guess at Step 2c is correct iff there exists  $T_{M_j}^*$ , where  $c(v)$  mutates exactly once and edge  $(r, p) \in T_{M_j}^*$  with  $r[c(v)] = 1$ ,  $p[c(v)] = 0$ . Implementation details for Steps 2a, 2c, and 3 are provided in Section 4.3.

*algorithm for ease of exposition, it is not hard to see that the algorithm can be derandomized by exploring all possible roots at Step 9. The derandomized algorithm has a total runtime of  $O((72\kappa)^q nm + nm^2)$ .*  $\square$

## 4 FPT ALGORITHM

This section deals with the complete description and analysis of our FPT algorithm for the BNPP problem. Throughout this section, we will use the second definition of a phylogeny (Definition 1). For ease of exposition, we first describe a randomized algorithm for the BNPP problem that runs in time  $O(18^q + qnm^2)$  and returns an optimum phylogeny with a probability of at least  $8^{-q}$ . We later show how to derandomize it. In Section 4.1, we first provide the complete pseudocode and describe it. In Section 4.2, we prove the correctness of the algorithm. In Section 4.3, we upper bound the runtime for the randomized and derandomized algorithms and the probability that the randomized algorithm returns an optimum phylogeny. The above work follows that presented in a preliminary paper on the topic [2]. Finally, in Section 4.4, we show how to tighten the above bounds on the derandomized algorithm to achieve our final result of  $O(21^q + 8^q nm^2)$  runtime.

### 4.1 Description

We begin with a high-level description of our randomized algorithm. The algorithm iteratively finds a set of edges  $E$  that decomposes an optimum phylogeny  $T_I^*$  into at most  $q$  components. An optimum phylogeny for each component is then constructed using a simple method and returned along with edges  $E$  as an optimum phylogeny for  $I$ .

We can alternatively think of the algorithm as a recursive divide-and-conquer procedure. Each recursive call to the algorithm attempts to reconstruct an optimum phylogeny for an input matrix  $M$ . The algorithm identifies a character  $c$  such that there exists an optimum phylogeny  $T_M^*$  in which  $c$  mutates exactly once. Therefore, there is exactly one edge

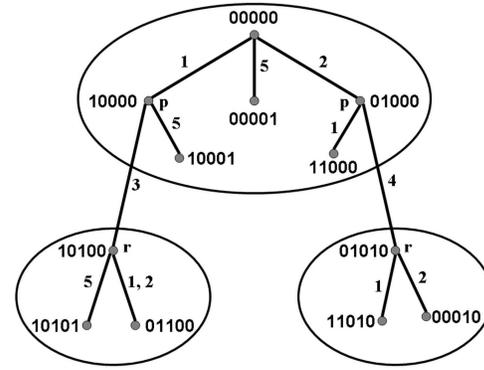


Fig. 5. Example illustrating the reconstruction. The underlying phylogeny is  $T_I^*$ ; taxa  $r$  and  $p$  (both could be Steiner) are guessed to create  $E = \{(10000, 10100), (01000, 01010)\}$ ;  $E$  induces three components in  $T_I^*$ . When all taxa in  $T_I^*$  are considered, character 3 conflicts with 1, 2, and 5 and character 4 conflicts with 1 and 2; two components are perfect (penalty 0) and one has a penalty of 2;  $\text{penalty}(I) =_{\text{def}} \text{penalty}(T_I^*) = 7$ .

$e \in T_M^*$  for which  $c \in \mu(e)$ . The algorithm then guesses the vertices that are adjacent to  $e$  as  $r, p$ . The matrix  $M$  can now be partitioned into matrices  $M0$  and  $M1$  based on the state at character  $c$ . Clearly, all of the taxa in  $M1$  reside on one side of  $e$  and all of the taxa in  $M0$  reside on the other side. The algorithm adds  $r$  to  $M1$  and  $p$  to  $M0$  and recursively computes the optimum phylogeny for  $M0$  and  $M1$ . An optimum phylogeny for  $M$  can be reconstructed as the union of any optimum phylogeny for  $M0$  and  $M1$ , along with the edge  $(r, p)$ . We require at most  $q$  recursive calls. When the recursion bottoms out, we use a simple method to solve for the optimum phylogeny.

We describe and analyze the iterative method that flattens the above recursion to simplify the analysis. For the sake of simplicity, we also define the following notations:

- For the set of taxa  $M$ ,  $M(i, s)$  refers to the subset of taxa that contains state  $s$  at character  $i$ .
- For a phylogeny  $T$  and character  $i$  that mutates exactly once in  $T$ ,  $T(i, s)$  refers to the maximal subtree of  $T$  that contains state  $s$  on character  $i$ .

The pseudocode for the above described algorithm is provided in Fig. 4. The algorithm performs “guesses” at Steps 2a and 2c. If all of the guesses performed by the algorithm are “correct,” then it returns an optimum phylogeny. The guess at Step 2a is correct iff there exists  $T_{M_j}^*$ , where  $c(v)$  mutates exactly once. The guess at Step 2c is correct iff there exists  $T_{M_j}^*$ , where  $c(v)$  mutates exactly once and edge  $(r, p) \in T_{M_j}^*$  with  $r[c(v)] = 1, p[c(v)] = 0$ . Implementation details for Steps 2a, 2c, and 3 are provided in Section 4.3. An example illustrating the reconstruction is provided in Fig. 5.

### 4.2 Correctness

We will now prove the correctness of the pseudocode under the assumption that all of the guesses performed by our algorithm are correct. Specifically, we will show that, if  $\text{penalty}(I) \leq q$ , then function **buildNPP** returns an optimum phylogeny. The following lemma proves the correctness of our algorithm:

**Lemma 4.1.** *At any point in the execution of the algorithm, an optimum phylogeny for  $I$  can be constructed as  $E \cup (\cup_i T_i)$ , where  $T_i$  is any optimum phylogeny for  $M_i \in L$ .*

**Proof.** We prove the lemma using induction. The lemma is clearly true at the beginning of the routine when  $L = \{I\}$ ,  $E = \emptyset$ . As an inductive hypothesis, assume that the above property is true right before the execution of Step 2e. Consider any optimum phylogeny  $T_{M_j}^*$  where  $c(v)$  mutates exactly once and on the edge  $(r, p)$ . Phylogeny  $T_{M_j}^*$  can be decomposed into  $T_{M_j}^*(c(v), 0) \cup T_{M_j}^*(c(v), 1) \cup (r, p)$  with length

$$l = \text{length}(T_{M_j}^*(c(v), 0)) + \text{length}(T_{M_j}^*(c(v), 1)) + d(r, p).$$

Again, since  $c(v)$  mutates exactly once in  $T_{M_j}^*$ , all of the taxa in  $M_0$  and  $M_1$  are also in  $T_{M_j}^*(c(v), 0)$  and  $T_{M_j}^*(c(v), 1)$ , respectively. Let  $T'$  and  $T''$  be arbitrary optimum phylogenies for  $M_0$  and  $M_1$ , respectively. Since  $p \in M_0$  and  $r \in M_1$ , we know that  $T' \cup T'' \cup (r, p)$  is a phylogeny for  $M_j$  with cost  $\text{length}(T') + \text{length}(T'') + d(r, p) \leq l$ . By the inductive hypothesis, we know that an optimum phylogeny for  $I$  can be constructed using any optimum phylogeny for  $M_j$ . We have now shown that, using any optimum phylogeny for  $M_0$  and  $M_1$  and adding edge  $(r, p)$ , we can construct an optimum phylogeny for  $M_j$ . Therefore, the proof follows by induction.  $\square$

### 4.3 Initial Bounds

In this section, we bound the probability of correct guesses, analyze the runtime, and show how to derandomize the algorithm. We perform two guesses at Steps 2a and 2c. Lemmas 4.2 and 4.6 bound the probability that all of the guesses performed at these steps are correct throughout the execution of the algorithm.

**Lemma 4.2.** *The probability that all guesses performed at Step 2a are correct is at least  $4^{-q}$ .*

**Proof. Implementation.** The guess at Step 2a is implemented by selecting  $v$  uniformly at random from  $\cup_i N(M_i)$ .

To prove the lemma, we first show that the number of iterations of the while loop (Step 2) is at most  $q$ . Consider any one iteration of the while loop. Since  $v$  is a nonisolated vertex of the conflict graph,  $c(v)$  shares all four gametes with some other character  $c'$  in some  $M_j$ . Therefore, in every optimum phylogeny  $T_{M_j}^*$  that mutates  $c(v)$  exactly once, there exists a path  $P$  starting with edge  $e_1$  and ending with  $e_3$ , both mutating  $c'$  and containing edge  $e_2$  mutating  $c(v)$ . Furthermore, the path  $P$  contains no other mutations of  $c(v)$  or  $c'$ . At the end of the current iteration,  $M_j$  is replaced by  $M_0$  and  $M_1$ . Both subtrees of  $T_{M_j}^*$  containing  $M_0$  and  $M_1$  contain (at least) one mutation of  $c'$  each. Therefore,  $\text{penalty}(M_0) + \text{penalty}(M_1) < \text{penalty}(M_j)$ . Since  $\text{penalty}(I) \leq q$ , there can be at most  $q$  iterations of the while loop.

We now bound the probability. Intuitively, if  $|\cup_i N(M_i)|$  is very large, then the probability of a correct guess is large since at most  $q$  out of  $|\cup_i N(M_i)|$  characters can mutate multiple times in  $T_{M_j}^*$ . On the other hand, if  $|\cup_i N(M_i)| = q$ , then we terminate the loop. Formally, at each iteration,  $|\cup_i N(M_i)|$  reduces by at least 1 (guessed vertex  $v$  is no longer in  $\cup_i N(M_i)$ ). Therefore, in the worst

case (to minimize the probability of correct guesses), we can have  $q$  iterations of the loop, with  $q + 1$  nonisolated vertices in the last iteration and  $2q$  in the first iteration. The probability in such a case that all guesses are correct is at least

$$\left(\frac{q}{2q}\right) \times \left(\frac{q-1}{2q-1}\right) \times \dots \times \left(\frac{1}{q+1}\right) = \frac{1}{\binom{2q}{q}} \geq 2^{-2q}.$$

$\square$

#### 4.3.1 Application of Buneman Graphs

We now show that  $r, p$  can be found efficiently. To prove this, we need some tools from the theory of Buneman graphs [24]. Let  $M$  be a set of taxa defined by character set  $C$  of size  $m$ . A Buneman graph  $F$  for  $M$  is a vertex-induced subgraph of the  $m$ -cube. Graph  $F$  contains vertices  $v$  iff, for every pair of characters  $i, j \in C$ ,  $(v[i], v[j]) \in G_{i,j}$ . Recall that  $G_{i,j}$  is the set of gametes (or projection of  $M$  on dimensions  $i$  and  $j$ ). Each edge of the Buneman graph is labeled with the character at which the adjacent vertices differ.

We will use the Buneman graph to show how to incrementally extend a set of taxa  $M$  by adding characters that share exactly two gametes with some existing character. As before, we can assume without loss of generality that the all-zero taxon is present in  $M$ . Therefore, if a pair of characters shares exactly two gametes, then they are identical. Assume that we want to add character  $i$  to  $M$ , and  $i' \in M$  is identical to  $i$ . We extend  $M$  to  $M'$  by first adding the states on character  $i'$  for all taxa. For the rest of the discussion, let  $G_{i,j}$  be the set of gametes shared between characters  $i$  and  $j$  in matrix  $M'$ . We extend  $M'$  to  $M''$  by adding a taxon  $t$  such that  $t[i] = 0$ ,  $t[i'] = 1$ , and, for all other characters  $j$ , if  $(0, 1) \notin G_{j,i}$ , then  $t[j] = 1$ ; else,  $t[j] = 0$ . Since we introduced a new gamete on  $i, i'$ , no pair of characters share exactly two gametes in  $M''$ . Therefore, a Buneman graph  $G''$  for  $M''$  can be constructed as before. A Buneman graph is a median graph [24] and, clearly, a subgraph of the  $m + 1$ -cube, where  $m + 1$  is the number of characters in  $M''$ . Every taxon in  $M'$  is present in  $G''$  by construction. Using the two properties, we have the following lemma:

**Lemma 4.3.** *Every optimum phylogeny for the taxa in  $M'$  defined over the  $m + 1$  characters is contained in  $G''$  (see [24, Section 5.5] for more details).*

We now show the following important property on the extended matrix  $M''$ .

**Lemma 4.4.** *If a pair of characters  $c, c'$  conflict in  $M''$ , then they conflict in  $M'$ .*

**Proof.** For the sake of contradiction, assume not. Clearly,  $i$  and  $i'$  share exactly three gametes in  $M''$ . Now, consider any character  $j$  and assume that  $j$  and  $i$  shared exactly three gametes in  $M'$ . For the newly introduced taxon  $t$ ,  $t[i] = 0$ . If  $t[j] = 1$ , then  $j$  and  $i$  cannot share the  $(0, 1)$  gamete in  $M''$  and, therefore, they do not conflict. If  $t[j] = 0$ , then the newly introduced taxon creates the  $(0, 0)$  gamete, which should be present in all pairs of characters. Now, consider the pair of characters  $(j, i')$ . If  $t[j] = 1$ , then, in

any taxon  $t'$  of  $M'$ , if  $t'[j] = 1$ , then  $t'[i] = 1$  and, therefore,  $t[i] = 1$  (since  $i$  and  $i'$  are identical on all taxa except  $t$ ) and, therefore,  $(1, 1)$  cannot be a newly introduced gamete. If  $t[j] = 0$ , then there exists some taxon  $t'$  for which  $t'[j] = 0$  and  $t'[i] = 1$  and, therefore,  $t'[i'] = 1$  and, again,  $(0, 1)$  cannot be a newly introduced gamete. Finally, consider any pair of characters  $j, j'$ . If taxon  $t$  introduces gamete  $(0, 1)$ , then there exists some taxon  $t'$  with  $t'[j] = 0$  and  $t'[i] = 1$ . If  $t'[j'] = 1$ , then  $(0, 1)$  cannot be a new gamete. If  $t'[j'] = 0$ , then  $t[j'] = 0$  and not 1. The case when  $(1, 0)$  is introduced by  $t$  is symmetric. Finally, if  $t$  introduces  $(1, 1)$ , then consider any taxon  $t'$  with  $t'[i] = 1$ . It has to be the case that  $t'[j] = t'[j'] = 1$  and, therefore,  $(1, 1)$  cannot be a newly introduced gamete.  $\square$

We now have the following lemma:

**Lemma 4.5.** *In every optimum phylogeny  $T_M^*$ , the conflict graph on the set of taxa in  $T_M^*$  (Steiner vertices included) is the same as the conflict graph on  $M$ .*

**Proof.** We say that a subgraph  $F'$  of  $F$  is the same as an edge-labeled tree  $T$  if  $F'$  is a tree and  $T$  can be obtained from  $F'$  by suppressing degree-2 vertices. A phylogeny  $T$  is contained in a graph  $F$  if there exists an edge-labeled subgraph  $F'$  that is the same as the edge-labeled (by function  $\mu$ ) phylogeny  $T$ . We know from Lemma 4.3 that all optimum phylogenies  $T_M^*$  for  $M$  are contained in the (extended) Buneman graph of  $M$ . Lemma 4.4 shows that the conflict graph on  $M''$  (and, therefore, on the extended Buneman graph of  $M''$ ) is the same as the conflict graph of  $M$ .  $\square$

**Lemma 4.6.** *The probability that all guesses performed at Step 2c are correct is at least  $2^{-q}$ .*

**Proof. Implementation.** We first show how to perform the guess efficiently. For every character  $i$ , we perform the following steps in order:

1. If all taxa in  $M0$  contain the same state  $s$  in  $i$ , then fix  $r[i] = s$ .
2. If all taxa in  $M1$  contain the same state  $s$  in  $i$ , then fix  $r[i] = s$ .
3. If  $r[i]$  is unfixed, then guess  $r[i]$  uniformly at random from  $\{0, 1\}$ .

Assuming that the guess at Step 2a (Fig. 4) is correct, we know that there exists an optimum phylogeny  $T_{M_j}^*$  on  $M_j$  where  $c(v)$  mutates exactly once. Let  $e \in T_{M_j}^*$  such that  $c(v) \in \mu(e)$ . Let  $r'$  be an end point of  $e$  such that  $r'[c(v)] = 1$  and let  $p'$  be the other end point. If the first two conditions hold with the same state  $s$ , then character  $i$  does not mutate in  $M_j$ . In such a case, we know that  $r'[i] = s$  since  $T_{M_j}^*$  is optimal and the above method ensures that  $r[i] = s$ . Notice that, if both conditions are satisfied simultaneously with different values of  $s$ , then  $i$  and  $c(v)$  share exactly two gametes in  $M_j$  and, therefore,  $i, c(v) \in \mu(e)$ . Hence,  $r'[i] = r[i]$ . We now consider the remaining cases when exactly one of the above conditions holds. We show that, if  $r[i]$  is fixed to  $s$ , then  $r'[i] = s$ . Note that, in such a case, at least one of  $M0, M1$  contains both of the states on  $i$  and  $i, c(v)$  share at least

three gametes in  $M_j$ . The proof can be split into two symmetric cases based on whether  $r$  is fixed on Condition 1 or 2. One case is presented below:

**Taxon  $r[i]$  is fixed based on Condition 1.** In this case, all of the taxa in  $M0$  contain the same state  $s$  on  $i$ . Therefore, the taxa in  $M1$  should contain both states on  $i$ . Hence,  $i$  mutates in  $T_{M_j}^*(c(v), 1)$ . For the sake of contradiction, assume that  $r'[i] \neq s$ . If  $i \notin \mu(e)$ , then  $p'[i] \neq s$ . However, all of the taxa in  $M0$  contain state  $s$ . This implies that  $i$  mutates in  $T_{M_j}^*(c(v), 0)$  as well. Therefore,  $i$  and  $c(v)$  share all four gametes on  $T_{M_j}^*$ . However,  $i$  and  $c(v)$  share at most three gametes in  $M_j$ —one in  $M0$  and at most two in  $M1$ . This leads to a contradiction to Lemma 4.5. Once  $r$  is guessed correctly,  $p$  can be computed since it is identical to  $r$  in all characters except  $c(v)$  and those that share two gametes with  $c(v)$  in  $M_j$ . We make a note here that we are assuming that  $e$  does not mutate any character that does not share two gametes with  $c(v)$  in  $M_j$ . This creates a small problem that, although the length of the tree constructed is optimal,  $r$  and  $p$  could be degree-2 Steiner vertices. If, after constructing the optimum phylogenies for  $M0$  and  $M1$ , we realize that this is the case, then we simply add the mutation adjacent to  $r$  and  $p$  to the edge  $(r, p)$  and return the resulting phylogeny where both  $r$  and  $p$  are not degree-2 Steiner vertices.

The above implementation therefore requires only guessing states corresponding to the remaining unfixed characters of  $r$ . If a character  $i$  violates the first two conditions, then  $i$  mutates once in  $T_{M_j}^*(i, 0)$  and once in  $T_{M_j}^*(i, 1)$ . If  $r[i]$  has not been fixed, then we can associate a pair of mutations of the same character  $i$  with it. At the end of the current iteration,  $M_j$  is replaced with  $M0$  and  $M1$  and each contains exactly one of the two associated mutations. Therefore, if  $q'$  characters are unfixed, then  $\text{penalty}(M0) + \text{penalty}(M1) \leq \text{penalty}(M_j) - q'$ . Since  $\text{penalty}(I) \leq q$ , throughout the execution of the algorithm, there are  $q$  unfixed states. Therefore, the probability of all the guesses being correct is  $2^{-q}$ .  $\square$

This completes our analysis for upper bounding the probability that the algorithm returns an optimum phylogeny. We now analyze the runtime. We use the following lemma to show that we can efficiently construct optimum phylogenies at Step 3 in the pseudocode:

**Lemma 4.7.** *For a set of taxa  $M$ , if the number of nonisolated vertices of the associated conflict graph is  $t$ , then an optimum phylogeny  $T_M^*$  can be constructed in time  $O(3^s 6^t + nm^2)$ , where  $s = \text{penalty}(M)$ .*

**Proof.** We use the approach described by Gusfield and Bansal (see [16, Section 7]) that relies on the Decomposition Optimality Theorem for recurrent mutations. We first construct the conflict graph and identify the nontrivial connected components of it in time  $O(nm^2)$ . Let  $\kappa_i$  be the set of characters associated with component  $i$ . We compute the Steiner minimum tree  $T_i$  for character set  $\kappa_i$ . The remaining conflict-free characters in  $C \setminus \cup_i \kappa_i$  can be added by contracting each  $T_i$  to vertices and solving the perfect phylogeny problem using Gusfield's linear time algorithm [14].

Since  $\text{penalty}(M) = s$ , there are at most  $s + t + 1$  distinct bit strings defined over character set  $\cup_i \kappa_i$ . The Steiner space is bounded by  $2^t$  since  $|\cup_i \kappa_i| = t$ . Using the Dreyfus-Wagner recursion [22], the total runtime for solving all Steiner tree instances is  $O(3^{s+t}2^t)$ .  $\square$

**Lemma 4.8.** *The algorithm described solves the BNPP problem in time  $O(18^q + qnm^2)$  with a probability of at least  $8^{-q}$ .*

**Proof.** For a set of taxa  $M_i \in L$  (Step 3, Fig. 4), using Lemma 4.7, an optimum phylogeny can be constructed in time  $O(3^{s_i}6^{t_i} + nm^2)$ , where  $s_i = \text{penalty}(M_i)$  and  $t_i$  is the number of nonisolated vertices in the conflict graph of  $M_i$ . We know that  $\sum_i s_i \leq q$  (since  $\text{penalty}(I) \leq q$ ) and  $\sum_i t_i \leq q$  (stopping condition of the while loop). Therefore, the total time to reconstruct the optimum phylogenies for all  $M_i \in L$  is bounded by  $O(18^q + nm^2)$ . The runtime for the while loop is bounded by  $O(qnm^2)$ . Therefore, the total runtime of the algorithm is  $O(18^q + qnm^2)$ . Combining Lemmas 4.2 and 4.6, the total probability that all guesses performed by the algorithm is correct is at least  $8^{-q}$ .  $\square$

**Lemma 4.9.** *The algorithm described above can be derandomized to run in time  $O(72^q + 8^qnm^2)$ .*

**Proof.** It is easy to see that Step 2c can be derandomized by exploring all possible states for the unfixed characters. Since there are at most  $q$  unfixed characters throughout the execution, there are  $2^q$  possibilities for the states.

However, Step 2a cannot be derandomized naively. We use the technique of a bounded search tree [6] to derandomize it efficiently. We select an arbitrary vertex  $v$  from  $\cup_i N(M_i)$ . We explore both the possibilities on whether  $v$  mutates once or multiple times. We can associate a search (binary) tree with the execution of the algorithm, where each node of the tree represents a selection  $v$  from  $\cup_i N(M_i)$ . One child edge represents the execution of the algorithm assuming that  $v$  mutates once and the other assuming that  $v$  mutates multiple times. In the execution where  $v$  mutates multiple times, we select a different vertex from  $\cup_i N(M_i)$  and again explore both paths. The height of this search tree can be bounded by  $2q$  because at most  $q$  characters can mutate multiple times. The path of height  $2q$  in the search tree is an interleaving of  $q$  characters that mutate once and  $q$  characters that mutate multiple times. Therefore, the size of the search tree is bounded by  $4^q$ .

Combining the two results, the algorithm can be derandomized by solving at most  $8^q$  different instances of Step 3 while traversing the while loop  $8^q$  times for a total runtime of  $O(144^q + 8^qnm^2)$ . This is, however, an overestimate. Consider any iteration of the while loop when  $M_j$  is replaced with  $M_0$  and  $M_1$ . If a state in character  $c$  is unfixed and is therefore guessed, we know that there are two associated mutations of character  $c$  in both  $M_0$  and  $M_1$ . Therefore, at iteration  $i$ , if  $q_i$  states are unfixed, then  $\text{penalty}(M_0) + \text{penalty}(M_1) \leq \text{penalty}(M_j) - q_i$ . At the end of the iteration, we can reduce the value of  $q$  used in Step 2 by  $q_i$  since the penalty has been reduced by  $q_i$ . Intuitively, this implies that, if we perform a total of  $q'$  guesses (or enumerations) at Step 2c, then, at Step 3, we only need to solve Steiner trees on  $q - q'$  characters.

The additional cost,  $2^{q'}$ , that we incur results in reducing the runtime of Step 3 to  $O(18^{q-q'} + qnm^2)$ . Therefore, the total runtime is  $O(72^q + 8^qnm^2)$ .  $\square$

#### 4.4 Improving the Runtime Bounds

In Lemma 4.9, we showed that the guesses performed at Step 2c of the pseudocode in Fig. 4 do not affect the overall runtime. We can also establish a trade-off along similar lines for Step 2a that can reduce the theoretical runtime bounds. We now analyze the details of such a trade-off in the following lemma:

**Lemma 4.10.** *The algorithm presented above runs in time  $O(21^q + 8^qnm^2)$ .*

**Proof.** For the sake of this analysis, we can declare each character to be in either a “marked” state or an “unmarked” state. At the beginning of the algorithm, all of the characters are “unmarked.” As the algorithm proceeds, we will mark characters to indicate that the algorithm has identified them as mutating more than once in  $T^*$ .

We will then examine two parameters,  $\rho$  and  $\gamma$ , which specify the progress made by the derandomized algorithm in either identifying multiple-mutating characters or reducing the problem to subproblems of lower total penalty. Consider the set of characters  $S$  such that, for all  $c \in S$ , character  $c$  is unmarked and there exists matrix  $M_i$  such that  $c$  mutates more than once in  $T_{M_i}^*$ . We define parameter  $\rho$  to be  $|S|$ . Parameter  $\rho$  intuitively refers to the number of characters mutating more than once (within trees  $T_{M_i}^*$ ) that have not yet been identified. Parameter  $\gamma$  denotes the sum of the penalties of the remaining matrices  $M_i$ ,  $\gamma = \sum_i \text{penalty}(M_i)$ .

Consider Step 2a in Fig. 4, when the algorithm selects character  $c(v)$ . After selecting  $c(v)$ , the algorithm proceeds to explore both cases when  $c(v)$  either mutates once or multiple times in  $T_{M_j}^*$ . In the first case,  $\text{penalty}(T_{M_j}^*)$  decreases by at least 1. Therefore,  $\gamma$  decreases by at least 1. In the second case, the algorithm has successfully identified a multiple mutant. We now proceed to mark character  $c(v)$ , which reduces  $\rho$  by 1 and leaves  $\gamma$  unchanged.

If the main loop at Step 3 terminates, then the algorithm finds optimal Steiner trees using the Dreyfus-Wagner recursion and the runtime is bounded by  $18^\gamma$  using Lemma 4.7, as before. Therefore, the runtime of this portion of our algorithm can be expressed as

$$T(\gamma, \rho) \leq \max\{18^\gamma, T(\gamma - 1, \rho) + T(\gamma, \rho - 1) + 1\}.$$

Function  $T(\gamma, \rho)$  can be upper bounded by  $18^{\gamma+1}(19/17)^{\rho+1}$ . We can verify this by induction. The right side of the above equation is

$$\begin{aligned} & \max\{18^\gamma, 18^\gamma(19/17)^{\rho+1} + 18^{\gamma+1}(19/17)^\rho + 1\} \\ & = \max\{18^\gamma, 18^\gamma(19/17)^\rho(19/17 + 18) + 1\} \\ & \leq \max\{18^\gamma, 18^\gamma(19/17)^\rho(19/17 + 19)\} \\ & = 18^{\gamma+1}(19/17)^{\rho+1}. \end{aligned}$$

Since we know that  $\gamma \leq q$  and  $\rho \leq q$ , we can bound  $T(q, q) = O(20.12^q)$ . Therefore, we can improve the

runtime bound for the complete algorithm to  $O(20.12^q + 8^q nm^2)$ .  $\square$

We note that further improvements may be achievable in practice for moderate  $q$  by preprocessing possible Steiner tree instances. If all Steiner tree-problem instances on the  $q$ -cube are solved in a preprocessing step, then our runtime would depend only on the number of iterations of the while loop, which is  $O(8^q nm^2)$ . Such preprocessing would be impossible to perform with previous methods. Alternate algorithms for solving Steiner trees may be faster in practice as well.

## 5 EXPERIMENTS

We tested both algorithms using a selection of nonrecombining DNA sequences. These include mitochondrial DNA (mtDNA) samples from two human populations [28] and a chimpanzee population [27], Y-chromosome samples from human [19] and chimpanzee populations [27], and a bacterial DNA sample [21]. Such nonrecombining data sources provide a good test for the algorithms' ability to perform inferences in situations where recurrent mutation is the probable source of any deviation from the perfect phylogeny assumption.

We implemented variants of both algorithms. The simple algorithm was derandomized and used along with a standard implementation of the Dreyfus-Wagner routine. For the FPT algorithm, we implemented the randomized variant described above using an optimized Dreyfus-Wagner routine. The randomized algorithm takes two parameters,  $q$  and  $p$ , where  $q$  is the imperfectness and  $p$  is the maximum probability that the algorithm has failed to find an optimal solution of imperfectness  $q$ . On each random trial, the algorithm tallies the probability of failure of each random guess, allowing it to calculate an upper bound on the probability that that trial failed to find an optimal solution. It repeats random trials until the accumulated failure probability across all trials is below the threshold  $p$ . An error threshold of 1 percent was used for the present study.

The results are summarized in Table 1. Successive columns of the table list the source of the data, the input size, the optimal penalty  $q$ , the parsimony score of the resulting tree, the runtimes of both of our algorithms in seconds, and the number of trials the randomized FPT algorithm needed to reach a 1 percent error bound. All runtimes reported are based on execution on a 2.4 GHz Intel Pentium 4 computer with 1 Gbyte of RAM. One data point, the human mtDNA sample from the Buddhist population, was omitted from the results of the simple algorithm because it failed to terminate after 20 minutes of execution. All other instances were solved optimally by the simple algorithm and all were solved by the randomized FPT algorithm. The randomized variant of the FPT algorithm in all but one case significantly outperformed the derandomized simple algorithm in runtime. This result may reflect the superior asymptotic performance of the FPT algorithm in general, the performance advantage of the randomized versus the deterministic variants, and the advantage of a more highly optimized Dreyfus-Wagner subroutine. The

TABLE 1  
Empirical Results on a Collection of  
Real SNP Variation Data Sets

Description	Rows × Cols	$q$	Pars. Score	Run time – Simple (secs)	Run time – FPT (secs)	trials
mtDNA, genus Pan [27]	24×1041	2	63	0.59	0.14	25
chr Y, genus Pan [27]	15×98	1	99	0.33	0.02	12
Bacterial DNA sequence [21]	17×1510	7	96	0.47	4.61	262
HapMap chr Y, 4 ethnic groups [19]	150×49	1	16	0.3	0.02	16
mtDNA, Humans (Muslims) [28]	13×48	3	30	0.61	0.28	117
mtDNA, Humans (Buddhists) [28]	26×48	7	43	—	18.44	1026

randomized algorithm also generally needed far fewer trials to reach a high probability of success than would be expected from the theoretical error bounds, suggesting that those bounds are quite pessimistic for realistic data sets. Both implementations, however, appear efficient for biologically realistic data sets with moderate imperfectness.

We can compare the quality of our solutions to those produced on the same data sets by other methods. Our methods produced trees of identical parsimony score to those derived by the `pars` program from the PHYLIP package [10]. However, whereas we can guarantee the optimality of the returned results, `pars` does not provide any guarantee on the quality of the tree. (Note that our preliminary paper [23] incorrectly stated that `pars` produced an inferior tree on the chimpanzee mtDNA data set.) Our methods also yielded an identical output for the chimpanzee Y-chromosome data to a branch-and-bound method used in the paper in which that data was published [27].

## 6 CONCLUSIONS

We have presented two new algorithms for inferring optimal near-perfect binary phylogenies. The algorithms substantially improve on the runtimes of any previous methods for the BNPP problem. This problem is of considerable practical interest for phylogeny reconstruction from SNP data. Furthermore, our algorithms are easily implemented, unlike previous theoretical algorithms for this problem. The algorithms can also provide guaranteed optimal solutions in their derandomized variants, unlike popular fast heuristics for phylogeny construction. Experiments on several nonrecombining variation data sets have further shown the methods to be generally extremely fast on real-world data sets typical of those for which one would apply the BNPP problem in practice. Our algorithms perform in practice substantially better than would be

expected from their worst-case runtime bounds, with both proving practical for at least some problems with  $q$  as high as seven. The FPT algorithm in its randomized variant shows generally superior practical performance to the simple algorithm. In addition, the randomized algorithm appears to find optimal solutions for these data sets in far fewer trials than would be predicted from the worst-case theoretical bounds. Even the deterministic variant of the simple algorithm finds optimal solutions in less than one second for all but one example. The algorithms presented here thus represent the first practical methods for provably optimal near-perfect phylogeny inference from biallelic variation data.

## ACKNOWLEDGMENTS

This work was supported in part by US National Science Foundation Grants CCR-0105548, IIS-0612099, and CCR-0122581 (the ALADDIN project). The authors thank the anonymous reviewers of earlier work on this project for their many helpful suggestions. Preliminary work on the algorithms presented here was published in the *Proceedings of the International Workshop on Bioinformatics Research and Applications* [23] and the *Proceedings of the International Colloquium on Automata, Languages, and Programming* [2].

## 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," *SIAM J. Computing*, vol. 23, pp. 1216-1224, 1994.
- [2] G.E. Blelloch, K. Dhamdhere, E. Halperin, R. Ravi, R. Schwartz, and S. Sridhar, "Fixed Parameter Tractability of Binary Near-Perfect Phylogenetic Tree Reconstruction," *Proc. 33rd Int'l Colloquium Automata, Languages, and Programming*, 2006.
- [3] H. Bodlaender, M. Fellows, and T. Warnow, "Two Strikes against Perfect Phylogeny," *Proc. 19th Int'l Colloquium on Automata, Languages, and Programming*, pp. 273-283, 1992.
- [4] 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," *Theoretical Computer Science*, vol. 244, nos. 1-2, pp. 167-188, 2000.
- [5] M. Bonet, M. Steel, T. Warnow, and S. Yoosheph, "Better Methods for Solving Parsimony and Compatibility," *J. Computational Biology*, vol. 5, no. 3, pp. 409-422, 1992.
- [6] R.G. Downey and M.R. Fellows, *Parameterized Complexity*. Springer, 1999.
- [7] W.H. Day and D. Sankoff, "Computational Complexity of Inferring Phylogenies by Compatibility," *Systematic Zoology*, vol. 35, pp. 224-229, 1986.
- [8] P. Damaschke, "Parameterized Enumeration, Transversals, and Imperfect Phylogeny Reconstruction," *Proc. Int'l Workshop Parameterized and Exact Computation*, pp. 1-12, 2004.
- [9] E. Eskin, E. Halperin, and R.M. Karp, "Efficient Reconstruction of Haplotype Structure via Perfect Phylogeny," *J. Bioinformatics and Computational Biology*, vol. 1, no. 1, pp. 1-20, 2003.
- [10] J. Felsenstein, PHYLIP (Phylogeny Inference Package) version 3.6, distributed by the author, Dept. of Genome Sciences, Univ. of Washington, Seattle, 2005.
- [11] D. Fernandez-Baca and J. Lagergren, "A Polynomial-Time Algorithm for Near-Perfect Phylogeny," *SIAM J. Computing*, vol. 32, pp. 1115-1127, 2003.
- [12] L.R. Foulds and R.L. Graham, "The Steiner Problem in Phylogeny Is NP-Complete," *Advances in Applied Math.*, vol. 3, pp. 43-49, 1982.
- [13] G. Ganapathy, V. Ramachandran, and T. Warnow, "Better Hill-Climbing Searches for Parsimony," *Proc. Third Int'l Workshop Algorithms in Bioinformatics (WABI '03)*, pp. 254-258, 2003.
- [14] D. Gusfield, "Efficient Algorithms for Inferring Evolutionary Trees," *Networks*, vol. 21, pp. 19-28, 1991.
- [15] D. Gusfield, *Algorithms on Strings, Trees and Sequences*. Cambridge Univ. Press, 1999.
- [16] D. Gusfield and V. Bansal, "A Fundamental Decomposition Theory for Phylogenetic Networks and Incompatible Characters," *Proc. Ninth Ann. Int'l Conf. Research in Computational Molecular Biology*, pp. 217-232, 2005.
- [17] D. Gusfield, S. Eddhu, and C. Langley, "Efficient Reconstruction of Phylogenetic Networks with Constrained Recombination," *Proc. Second IEEE CS Computational Systems Bioinformatics Conf.*, pp. 363-374, 2003.
- [18] D.A. Hinds, L.L. Stuve, G.B. Nilsen, E. Halperin, E. Eskin, D.G. Ballinger, K.A. Frazer, and D.R. Cox, "Whole Genome Patterns of Common DNA Variation in Three Human Populations," *Science*, vol. 307, no. 5712, pp. 1072-1079, 2005.
- [19] The International HapMap Consortium, "The International HapMap Project," *Nature*, vol. 426, pp. 789-796, 2003.
- [20] S. Kannan and T. Warnow, "A Fast Algorithm for the Computation and Enumeration of Perfect Phylogenies," *SIAM J. Computing*, vol. 26, pp. 1749-1763, 1997.
- [21] M. Merimaa, M. Liivak, E. Heinaru, J. Truu, and A. Heinaru, "Functional Co-Adaption of Phenol Hydroxylase and Catechol 2,3-Dioxygenase Genes in Bacteria Possessing Different Phenol and P-Cresol Degradation Pathways," *Proc. Eighth Symp. Bacterial Genetics and Ecology*, 2005.
- [22] H.J. Promel and A. Steger, *The Steiner Tree Problem: A Tour through Graphs Algorithms and Complexity*. Vieweg Verlag, 2002.
- [23] S. Sridhar, K. Dhamdhere, G.E. Blelloch, E. Halperin, R. Ravi, and R. Schwartz, "Simple Reconstruction of Binary Near-Perfect Phylogenetic Trees," *Proc. Int'l Workshop Bioinformatics Research and Applications*, 2006.
- [24] C. Semple and M. Steel, *Phylogenetics*. Oxford Univ. Press, 2003.
- [25] M.A. Steel, "The Complexity of Reconstructing Trees from Qualitative Characters and Subtrees," *J. Classification*, vol. 9, pp. 91-116, 1992.
- [26] S.T. Sherry, M.H. Ward, M. Kholodov, J. Baker, L. Pham, E. Smigielski, and K. Sirotkin, "dbSNP: The NCBI Database of Genetic Variation," *Nucleic Acids Research*, vol. 29, pp. 308-311, 2001.
- [27] A.C. Stone, R.C. Griffiths, S.L. Zegura, and M.F. Hammer, "High Levels of Y-Chromosome Nucleotide Diversity in the Genus Pan," *Proc. Nat'l Academy of Sciences USA*, vol. 99, no. 1, pp. 43-48, 2002.
- [28] T. Wirth, X. Wang, B. Linz, R.P. Novick, J.K. Lum, M. Blaser, G. Morelli, D. Falush, and M. Achtman, "Distinguishing Human Ethnic Groups by Means of Sequences from *Helicobacter pylori*: Lessons from Ladakh," *Proc. Nat'l Academy of Sciences USA*, vol. 101, no. 14, pp. 4746-4751, 2004.



**Srinath Sridhar** received the BS degree in computer science from the University of Texas at Austin in 2003. Since then, he has been a PhD student in the Department of Computer Science at Carnegie Mellon University. His primary area of research is computational biology.



**Kedar Dhamdhere** received the PhD degree from the Computer Science Department, Carnegie Mellon University, in 2005. Since then, he has joined Google Inc. where he works on improving Web search.



**Guy E. Blelloch** received the BA degree in physics and the BS degree in engineering from Swarthmore College in 1983 and the MS and PhD degrees in computer science from the Massachusetts Institute of Technology in 1986 and 1988, respectively. He is currently a professor of computer science at Carnegie Mellon University and codirector of the ALADDIN center for the study of algorithms. His research interests are in programming languages and applied algorithms.



**R. Ravi** received the BTech degree in computer science and engineering from the Indian Institute of Technology, Madras, in 1989 and the PhD degree in computer science from Brown University in 1993. After postdoctoral fellowships at the University of California, Davis, and the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) at Princeton University, he joined the Operations Research Faculty in the Tepper School of Business at Carnegie Mellon University in 1995, where he is currently a Carnegie Bosch professor of operations research and computer science.



**Eran Halperin** received the MSc and PhD degrees from the Computer Science Department, Tel Aviv University, Israel. Since then, he has held research positions at the University of California, Berkeley, as a postdoctoral researcher and at Princeton University as a research associate and has been part of the Algorithm Development Group of Compugen Ltd., a bioinformatics company. Since 2005, he has been a senior research scientist at the International Computer Science Institute (ICSI), Berkeley, California.

His research is focused on statistical and computational approaches in biology, particularly genetics and applications to disease association studies.



**Russell Schwartz** received the BS, MEng, and PhD degrees from the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology, the last of which was in 2000. He later worked in the Informatics Research Group at Celera Genomics. Since 2002, he has been an assistant professor in the Department of Biological Sciences at Carnegie Mellon University.

▷ **For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/publications/dlib](http://www.computer.org/publications/dlib).**