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

S. Sridar and G. E. Blelloch are with the Computer Science Deptartment, Carnegie Mellon University, 5000 Forbes Ave., Pitsburgh, PA 15213 U.S.A. Email: {srinath,blelloch}@cs.cmu.edu.

K. Dhamdhere is with Google Inc., Mountain View, CA. Email: kedar.dhamdhere@gmail.com.

E. Halperin is with the International Computer Science Institute at the University of California, Berkeley, Berkely, CA Email: heran@icsi.berkeley.edu.

R. Ravi is with the Tepper School of Business, Carnegie Mellon University. Email: ravi@cmu.edu.

R. Schwartz is with the Department of Biological Sciences, Carnegie Mellon University. Email: russells@andrew.cmu.edu.

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 shown them to be extremely efficient in practice on biologically significant data sets. This work proves the BNPP problem 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

I. INTRODUCTION

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 (e.g. 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 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 while 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 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).

A. 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 running time $O((72\kappa)^q nm + nm^2)$, where κ 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 running time. Furthermore, the complexity of the previous work would make 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 as Sridhar et al. [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 run-time 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 in Blelloch et al. [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.

II. 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 $C = \{1, \cdots, m\}$ 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 $l(\{v \in V(T) | \text{degree}(v) \leq 2\}) \subseteq R(I)$. 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, $\texttt{length}(T) = \sum_{(u,v)\in E(T)} d(l(u), l(v))$, where d is the Hamming distance.

A phylogeny is called an optimum phylogeny if its length is minimized. We will assume that both states $0, 1$ 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, $\texttt{penalty}(T) = \texttt{length}(T) - m$; $\texttt{penalty}(I) = \texttt{penalty}(T^{opt})$, where T^{opt} is any optimum phylogeny on I.

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

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:

The BNPP problem: Given an integer q and an $n \times m$ binary input matrix I, if $\texttt{penalty}(I) \leq q$, 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

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, j is defined as: $G_{i,j} = \{(r[i], r[j]) | r \in R(I)\}$. Two characters i, j *share* t gametes in I i.f.f. $|G_{i,j}| = t$.
In other words, the set of gametes $G_{i,j}$ is a projection on the i, j dimensions.

Lemma 2.1: [14] An optimum phylogeny for input I is not perfect i.f.f. there exists two characters i, 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 i.f.f. $c(u), c(v)$ share all four gametes in I. Such a pair of characters are 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.

Simplifications: We assume that the all zeros taxon is present in the input. If not, using our freedom of labeling, we convert the data into an equivalent input containing the all zeros taxon (see section 2.2 of Eskin et al [9] 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 the phylogenies are rooted at the all zeros taxon.

III. 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 pre-processing 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'', $|G_{c',c''}| \geq 3$.

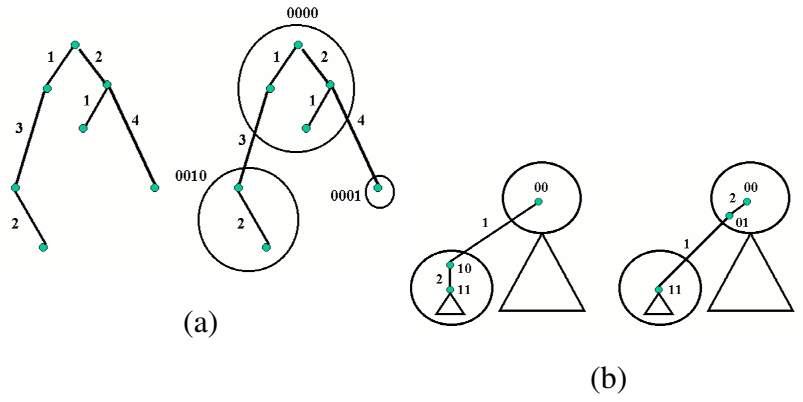

Fig. 1. (a) Phylogeny T and skeleton $s(T, C')$, $C' = \{3, 4\}$. Edges are labeled with characters that mutate μ and super nodes with tags t. (b) Transform to remove a degree 2 Steiner root from a super node. Note: the size of the phylogeny is unchanged.

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 non-informative. It is not hard to show that this preprocessing step does not change the correctness or running time of our algorithm.

The following additional defintions 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 *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 10: A *tag* $t(u) \in \{0, 1\}^m$ of super node 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 zeros 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 $t(u)[i] = 1$ if and only if character i mutates exactly once in the path from the root to u. Figure 1(a) shows an example of a skeleton of a phylogeny. We will use the term *sub-phylogeny* to refer to a subtree of a phylogeny.

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 $\texttt{PP}(V_{PP}, E_{PP})$ 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) $T_f :=$ **linkTrees** (PP)
   d) if $\texttt{penalty}(T_f) \leq q$ then return T_f
4) return NIL

Fig. 2. Pseudo-code to find the skeleton.

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 is rooted at the all zeros 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 pseudo-code is provided in Figures 2 and 3.

Function `buildNPP` starts by determining the set of characters $c(V_{nis})$ that corresponds to the non-isolated 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 non-isolated 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 super node where every taxon resides as defined by function λ in Step 3b. This rooted skeleton PP is then passed into function `linkTrees` to complete the phylogeny.

function linkTrees (skeleton $Sk(V_s, E_s)$ **)**

1) let $S := \texttt{root}(Sk)$
2) let $R_S := \{s \in R | \lambda(s) = S\}$
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) := **linkTrees**(Sk_i)
4) let $\texttt{cost} := \sum_i c_i$
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 τ 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 $(r_S, \texttt{cost} + c_S)$

Fig. 3. Pseudo-code to construct and link imperfect phylogenies

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 using function λ, 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 sub-skeleton rooted at children super nodes of Sk. Tuple (r, c) returned by function call to $\texttt{linkTrees}(Sk)$ represents the cost c of the optimal phylogeny when the label of the root vertex in the root super node of Sk is r. Let $S = \texttt{root}(Sk)$ represent the root super node of skeleton Sk. R_S is the set of input taxa that map to super node S under function λ. Let its children super nodes be $S_1, S_2, \ldots$. Assume that recursive calls to $\texttt{linkTrees}(S_i)$ return (r_i, c_i). Notice that the parents of the set of roots r_i all reside in super node 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 τ 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 τ differ i.e. 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 super node 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} is identical to τ 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 super node.

Proof: Figure 1(b) shows how to transform a phylogeny that violates the property into one that doesn't. Root `10` is degree 2 Steiner and is moved into parent supernode as `01`. Since `10` was Steiner, the transformed tree contains all input. ■

Corollary 3.3: In T_{opt}, the LCA of the set τ is the root of super node 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 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 running time 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 probability 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 in function `linkTrees` is exactly $2^{-|D|}$. Notice that the Steiner tree in super node S has at least $|D|$ edges. Since $\texttt{penalty}(T_{opt}) \leq q$, 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 running time 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 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 total running time $O((72\kappa)^q nm + nm^2)$. ■

IV. FIXED PARAMETER TRACTABLE ALGORITHM

This section deals with the complete description and analysis of our fixed parameter tractable 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 probability at least 8^{-q}. We later show how to derandomize it. In sub-section IV-A, we first provide the complete pseudo-code and describe it. In sub-section IV-B, we prove the correctness of the algorithm. In sub-section IV-C, we upper-bound the running time 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 sub-section IV-D, we show how to tighten the above bounds on the derandomized algorithm to achieve our final result of $O(21^q + 8^q nm^2)$ run time.

A. 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 s.t. there exists an optimum phylogeny T_M^* in which c mutates exactly once. Therefore, there is exactly one edge $e \in T_M^*$ 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 the taxa in $M1$ reside on one side of e and all the taxa in $M0$ reside on the other side. The algorithm adds r to $M1$, 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 which flattens the above recursion to simplify the analysis. For the sake of simplicity, we also define the following notations:

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. Pseudo-code to solve the BNPP problem. For all $M_i \in L$, $N(M_i)$ is the set of non-isolated vertices in the conflict graph of M_i. Guess at Step 2a is correct i.f.f. there exists $T^*_{M_j}$ where $c(v)$ mutates exactly once. Guess at Step 2c is correct i.f.f. 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 IV-C.

- 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 pseudo-code for the above described algorithm is provided in Figure 4. The algorithm performs 'guesses' at Steps 2a and 2c. If all the guesses performed by the algorithm are 'correct' then it returns an optimum phylogeny. The guess at Step 2a is correct if and only if there exists $T^*_{M_j}$ where $c(v)$ mutates exactly once. The guess at Step 2c is correct if and only if 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 IV-C. An example illustrating the reconstruction is provided in Figure 5.

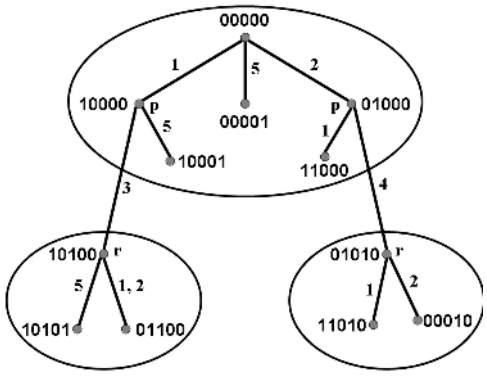

Fig. 5. Example illustrating the reconstruction. Underlying phylogeny is T_I^*; taxa r and p (both could be Steiner) are guessed to create $E = \{(10000, 10100), (01000, 01010)\}$; E induces three components in T_I^*. When all taxa in T_I^* are considered, character 3 conflicts with $1, 2$ and 5 and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

B. Correctness

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

Lemma 4.1: At any point in 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 inductive hypothesis, assume that the above property is true right before an 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 $l = \texttt{length}(T^*_{M_j}(c(v), 0)) + \texttt{length}(T^*_{M_j}(c(v), 1)) + d(r, p)$. Again, since $c(v)$ mutates exactly once in $T^*_{M_j}$, all the taxa in $M0$ and $M1$ are also in $T^*_{M_j}(c(v), 0)$ and $T^*_{M_j}(c(v), 1)$ respectively. Let T', T'' be *arbitrary* optimum phylogenies for $M0$ and $M1$ respectively. Since $p \in M0$ and $r \in M1$ we know that $T' \cup T'' \cup (r, p)$ is a phylogeny for M_j with cost $\texttt{length}(T') + \texttt{length}(T'') + d(r, p) \leq l$. 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 $M0$ and $M1$ and adding edge (r, p) we can

construct an optimum phylogeny for M_j. Therefore the proof follows by induction. ■

C. Initial Bounds

In this sub-section we bound the probability of correct guesses, analyze the running time 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 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 non-isolated 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 with $M0$ and $M1$. Both subtrees of $T^*_{M_j}$ containing $M0$ and $M1$ contain (at least) one mutation of c' each. Therefore, $\texttt{penalty}(M0) + \texttt{penalty}(M1) < \texttt{penalty}(M_j)$. Since $\texttt{penalty}(I) \leq q$, 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 $q+1$ non-isolated 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

$$(\frac{q}{2q}) \times (\frac{q-1}{2q-1}) \times \ldots \times (\frac{1}{q+1}) = \frac{1}{\binom{2q}{q}} \geq 2^{-2q}$$

■

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 if and only if 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, 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 zeros taxon is present in M. Therefore if a pair of characters share 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, j in matrix M'. We extend M' to M'' by adding a taxon t s.t. $t[i] = 0, t[i'] = 1$ 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 Section 5.5, [24] 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, i' share exactly three gametes in M''. Now consider any character j and assume that j, i shared exactly three gametes in M'. For the newly introduced taxon t, $t[i] = 0$. If $t[j] = 1$, then j, i cannot share $(0, 1)$ 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, 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. ■

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-two 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 μ) phylogeny T. We know from Lemma 4.3 that all optimum phylogenies T^*_M for M is 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. ■

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 (Figure 4) is correct, we know that there exists an optimum phylogeny $T^*_{M_j}$ on M_j where $c(v)$ mutates exactly once. Let $e \in T^*_{M_j}$ s.t. $c(v) \in \mu(e)$. Let r' be an end point of e s.t. $r'[c(v)] = 1$ and 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 hold. We show that if $r[i]$ is fixed

to s then $r'[i] = s$. Note that in such a case at least one of $M0, M1$ contain both the states on i and $i, c(v)$ share at least 3 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 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 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 3 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 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-two 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-two 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 $\texttt{penalty}(M0) + \texttt{penalty}(M1) \leq \texttt{penalty}(M_j) - q'$. Since $\texttt{penalty}(I) \leq q$, throughout the execution of the algorithm there are q unfixed states. Therefore the probability of all the guesses being correct is 2^{-q}. ■

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

Lemma 4.7: For a set of taxa M, if the number of non-isolated 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 $s = \texttt{penalty}(M)$.

Proof: We use the approach described by Gusfield and Bansal (see Section 7 of [16]) that relies on the Decomposition Optimality Theorem for recurrent mutations. We first construct the conflict graph and identify the non-trivial connected components of it in time $O(nm^2)$. Let κ_i be the set of characters associated with component i. We compute the Steiner minimum tree T_i for character set κ_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 $\texttt{penalty}(M) = s$, 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 run-time for solving all Steiner tree instances is $O(3^{s+t}2^t)$. ■

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

Proof: For a set of taxa $M_i \in L$ (Step 3, Figure 4), using Lemma 4.7 an optimum phylogeny can be constructed in time $O(3^{s_i}6^{t_i} + nm^2)$ where $s_i = \texttt{penalty}(M_i)$ and t_i is the number of non-isolated vertices in the conflict graph of M_i. We know that $\sum_i s_i \leq q$ (since $\texttt{penalty}(I) \leq q$) and $\sum_i t_i \leq q$ (stopping condition of the `while` loop). Therefore, the total time to reconstruct optimum phylogenies for all $M_i \in L$ is bounded by $O(18^q + nm^2)$. The running time for the while loop is bounded by $O(qnm^2)$. Therefore the total running time 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}. ■

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

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 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 v mutates once and the other assuming 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 running time of $O(144^q + 8^q nm^2)$. This is, however, an over-estimate. Consider any iteration of the while loop when M_j is replaced with $M0$ and $M1$. If a state in character c is unfixed and therefore guessed, we know that there are two associated mutations of character c in both $M0$ and $M1$. Therefore at iteration i, if q'_i states are unfixed, then $\texttt{penalty}(M0) + \texttt{penalty}(M1) \leq \texttt{penalty}(M_j) - q'_i$. At the end of the iteration we can reduce the value of q used in Step 2 by q'_i, since the penalty has 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 running time of Step 3 to $O(18^{q-q'} + qnm^2)$. Therefore the total running time is $O(72^q + 8^q nm^2)$. ■

D. Improving the Run Time Bounds

In Lemma 4.9, we showed that the guesses performed at Step 2c of the pseudo-code in Figure 4 do not affect the overall running time. We can also establish a trade-off along similar lines for Step 2a that can reduce the theoretical run-time 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 $O(21^q + 8^q nm^2)$.

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 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, ρ and γ, which specify the progress made by the de-randomized algorithm in either identifying multiply mutating characters or reducing the problem to sub-problems 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 ρ to be $|S|$. Parameter ρ, intuitively, refers to the number of characters mutating more than once (within trees $T^*_{M_i}$) that have not been identified yet. Parameter γ denotes the sum of the penalties of the remaining matrices M_i, $\gamma = \sum_i \text{penalty}(M_i)$.

Consider Step 2a of Figure 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, $\texttt{penalty}(T^*_{M_j})$ decreases by at least 1. Therefore, γ 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 ρ by 1 and leaves γ unchanged.

If the main loop at Step 3 terminates, then the algorithm finds optimal Steiner trees using the Dreyfus-Wagner recursion and the run-time is bounded by 18^γ using Lemma 4.7 as before. Therefore, the running time 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 run-time bound for the complete algorithm to $O(20.12^q + 8^q nm^2)$. ■

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

V. EXPERIMENTS

We tested both algorithms using a selection of non-recombining DNA sequences. These include mitochondrial DNA 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 non-recombining data sources provide a good test of 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% was used for the present study.

The results are summarized in Table I. 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 run times of both of our algorithms in seconds, and the number of trials the randomized FPT algorithm needed to reach a 1% error bound. All run times reported are based on execution on a 2.4 GHz Intel P4 computer with 1 Gb 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 run time. This result that 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 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 imperfection.

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 while we can guarantee 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 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].

TABLE I

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

VI. CONCLUSIONS

We have presented two new algorithms for inferring optimal near-perfect binary phylogenies. The algorithms substantially improve on the running times 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 non-recombining 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 run time 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, though, finds optimal solutions in under 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.

ACKNOWLEDGMENT

This work is supported in part by U.S. National Science Foundation grants CCR-0105548, IIS-0612099, and CCR-0122581 (The ALADDIN project). We thank anonymous reviewers of earlier work on this project for 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 Journal on 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. Intl. Colloq. Automata, Languages and Programming*, 2006.

[3] H. Bodlaender, M. Fellows and T. Warnow. "Two Strikes Against Perfect Phylogeny." *Proc. Intl. Colloq. 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, no. 1–2, pp. 167–188, 2000.

[5] M. Bonet, M. Steel, T. Warnow and S. Yooseph. "Better Methods for Solving Parsimony and Compatibility." *Journal of Computational Biology*, vol. 5, no. 3, pp. 409–422, 1992.

[6] R.G. Downey and M. R. Fellows. *Parameterized Complexity (Monographs in Computer Science)*, 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. Intl. Workshop on 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." *Journal of 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. Department of Genome Sciences, University of Washington, Seattle, 2005.

[11] D. Fernandez-Baca and J. Lagergren. "A Polynomial-Time Algorithm for Near-Perfect Phylogeny." *SIAM Journal on 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 Mathematics*, vol. 3, pp. 43–49, 1982.

[13] G. Ganapathy, V. Ramachandran and T. Warnow. "Better Hill-Climbing Searches for Parsimony." *Proc. Workshop on Algorithms in Bioinformatics*, 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 University Press, 1999.

[16] D. Gusfield and V. Bansal. "A Fundamental Decomposition Theory for Phylogenetic Networks and Incompatible Characters." *Proc. Research in Computational Molecular Biology (RECOMB)*, pp. 217–232, 2005.

[17] D. Gusfield, S. Eddhu and C. Langley. "Efficient Reconstruction of Phylogenetic Networks with Constrained Recombination." *Proc. Computational Systems Bioinformatics Conference*, pp. 363–374, 2003.

[18] D. A. Hinds, L. L. Stuve, G. B. Nilsen, E. Halperin, E. Eskin, D. G. Ballinger, K. A. Frazer, 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 Journal on 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." *Eighth Symposium on 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. Intl. Workshop on Bioinformatics Research and Applications*, 2006.

[24] C. Semple and M. Steel. *Phylogenetics.* Oxford University Press, 2003.

[25] M. A. Steel. “The Complexity of Reconstructing Trees from Qualitative Characters and Subtrees.” *Journal of 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.” *Proceedings of the National 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.” *Proceedings of the National Academy of Sciences USA*, vol. 101, no. 14, pp. 4746–4751, 2004.