Additive Approximation for Near-Perfect Phylogeny Construction *

Pranjal Awasthi, Avrim Blum, Jamie Morgenstern, and Or Sheffet

Carnegie Mellon University, Pittsburgh,
5000 Forbes Ave., Pittsburgh PA 15213, USA,
{pawasthi,avrim,jamiemmt,osheffet}@cs.cmu.edu

Abstract. We study the problem of constructing phylogenetic trees for a given set of species. The problem is formulated as that of finding a minimum Steiner tree on n points over the Boolean hypercube of dimension d. It is known that an optimal tree can be found in linear time [1] if the given dataset has a perfect phylogeny, i.e. cost of the optimal phylogeny is exactly d. Moreover, if the data has a near-perfect phylogeny, i.e. the cost of the optimal Steiner tree is $d + q$, it is known [2] that an exact solution can be found in running time which is polynomial in the number of species and d, yet exponential in q. In this work, we give a polynomial-time algorithm (in both d and q) that finds a phylogenetic tree of cost $d+O(q^2)$. This provides the best guarantees known—namely, a $(1 + o(1))$-approximation—for the case $\log(d) \ll q \ll \sqrt{d}$, broadening the range of settings for which near-optimal solutions can be efficiently found. We also discuss the motivation and reasoning for studying such additive approximations.

1 Introduction

Phylogenetics, a subfield of computational biology, aims to construct simple and accurate descriptions of evolutionary history. These descriptions are represented as evolutionary trees for a given set of species, each of which is represented by some set of features ([3,4]). A typical choice for these features are single nucleotide polymorphisms (SNPs), binary indicator variables for common mutations found in DNA[5,6]; see, for example, [2,1,7,8,9,10]. This challenging problem has attracted much attention in recent years, with progress in studying various computational formulations of this problem ([3,11,2,1,12,7]). The problem is often posed as that of constructing the most parsimonious tree induced by the set of species.

Formally, a *phylogeny* or a *phylogenetic tree* for a set C of n species, each represented by a string (called taxa) of length d over a finite alphabet Σ, is an unrooted tree $T = (V, E)$ such that $C \subseteq V \subseteq \Sigma^d$. Given a distance metric μ over

* This work was supported in part by the National Science Foundation under grant CCF-1116892, by an NSF Graduate Fellowship, and by the MSR-CMU Center for Computational Thinking.

Σ^d, we define the cost of T as $\sum_{(u,v)\in E} \mu(u,v)$. The tree of *maximum parsimony* for a dataset is the tree which minimizes this cost with respect to the Hamming metric; i.e., it is the optimum Steiner tree for the set C under this metric.

The Steiner tree problem is known to be NP-hard in general [13], and remains NP-hard even in the case of a binary alphabet with the metric induced by the Hamming distance [14]. Extensive recent work, both experimental and theoretical, has focused on the binary character set with the Hamming metric ([3,2,1,12,7,4,15,16]). This version of the phylogeny problem will also be the focus of this paper.

A phylogeny is called *perfect* if each coordinate $i \in [d]$ flips exactly once in the tree (representing a single mutation of i amongst the set of species)[1]. If a dataset admits a perfect phylogeny, an optimal tree can be constructed in polynomial time [17] (even linear time, in the case where the alphabet is binary [3]). In this work, we investigate *near perfect* phylogenies – instances whose optimal phylogenetic tree has cost $d + q$, where $q \ll d$. Near perfect phylogenies have been studied in theoretical ([11,2,12,16]) and experimental settings ([15]). The work of [11,2,12,16] has given a series of randomized algorithms which find the optimal phylogeny in running time polynomial in n and d but exponential in q. Clearly, when $q = \omega(\log d)$, these algorithms are not tractable.

An alternative approach for finding a phylogenetic tree of low cost is to use a generic Steiner tree approximation algorithm. The best current such algorithm yields a tree of cost at most $1.39(d+q)$ [18] (we comment that the exponential size of the explicit hypercube with respect to its small representation size requires one implement such an algorithm using techniques devised especially for the hypercube, e.g. Alon et al. [7].) However, notice that for moderate q (e.g., for $q = polylog(d)$), the *excess* of this tree—meaning the difference between its cost and d—may be extremely large compared to the excess q of the optimal tree. In such cases, one would much prefer an algorithm whose excess could be written as a function of q only.

In this work, we present a randomized $\mathrm{poly}(n, d, q)$-time algorithm that finds a phylogenetic tree of cost $d + O(q^2)$.

Theorem 1. *Given a set $C \subseteq \{0,1\}^d$ of n terminals, such that the optimal phylogeny of C has cost $d+q$, there exists a randomized $\mathrm{poly}(n, d, q)$-time algorithm that finds a phylogenetic tree of cost $d + O(q^2)$ w.p. $\geq 1/2$.*

Note that Theorem 1 provides a substantial improvement over prior work for the case that $\log d \ll q \ll \sqrt{d}$. In this range, the exact algorithms are no longer tractable, and the multiplicative approximations yield significantly worse bounds. Alternatively, viewed as a multiplicative guarantee, in this range our tree is within a $1 + o(1)$ factor of optimal. To the best of our knowledge, this is the first work to give an additive poly-time approximation to either the phylogeny problem or any (non-trivial) setting the Steiner tree problem. One immediate

[1] Without loss of generality, we may assume each coordinate flips at least once, since all coordinates on which all species agree may be discarded up front.

question, which remains open, is whether our results can be improved to $d+o(q^2)$ or perhaps even to $d + O(q)$.

The rest of the paper is organized as follows. After surveying related work in Section 1.1, we detail notation and preliminaries in Section 2. The presentation of our algorithm is partitioned into two parts. In Section 3, we present the algorithm for the case where no pair of coordinates is identical over all terminals (formal definition there). In Section 4, we alter the algorithm for the simple case, in a nontrivial way, so that the modified algorithm finds a low-cost phylogeny for any dataset. We conclude in Section 5 with a discussion, motivating the problem of near-perfect phylogeny tree from a different perspective, and present open problems for future research.

1.1 Related Work

As mentioned in the introduction, the problem of constructing an optimal phylogeny is NP-complete even when restricted to binary alphabets [14]. Schwartz et al. [11] give an algorithm based on an Integer Linear Programming (ILP) formulation to solve the multi-state problem optimally, and show experimentally the algorithm is efficient on small instances. Perfect phylogenies (datasets which admit a tree in which any coordinate changes exactly once) have optimal parsimony trees which can be constructed in linear time in the binary case [1] and in polynomial time for a fixed alphabet [12]. Unfortunately, finding the perfect phylogeny for arbitrary alphabets is NP-hard [19]. Recent work [2] gives an algorithm to construct optimal phylogenetic trees for binary, near-perfect phylogenies (where only a small number of coordinates mutate more than once in the optimal tree). However, the running time of the algorithm presented in their work [2] is exponential in the number of additional mutations.

There has also been a lot of work on computing multiplicative approximations to the Steiner tree problem. A Minimum Spanning Tree (MST) over the set of terminals achieves an approximation ratio of 2 and a long line of work has led to the current best bound of 1.39 [20,21,22,23,24,8,25,9,10,18]. The more recent of these papers use a result due to Borchers and Du [26] showing that an optimal Steiner tree can be approximated to arbitrary precision using k-restricted Steiner trees.

Some of these approximations to the Steiner tree problem are not immediately extendable to the problem of constructing phylogenetic trees. This is because the size of the vertex set for the phylogeny problem is exponential in d (there are 2^d vertices in the hypercube). If an algorithm works on an explicit representation of the graph G defined by the hypercube, then it does not solve the phylogeny problem in polynomial time. However, the line of work started by Robins and Zelikovsky [9,10] used the notion of k-restricted Steiner trees, which *can* be efficiently implemented on the hypercube. In particular, Alon et al. [7] showed that in finding the optimal k-restricted component for a given set of k terminals, it is sufficient to only consider topologies with the given k terminals at the leaves. Using this, they were able to extend that work to achieve a 1.55

approximation ratio for the maximum parsimony problem, and a 16/9 approximation for maximum likelihood. Byrka et al. [18] considered a new LP relaxation to the k-restricted Steiner tree problem and achieved an approximation ratio of 1.39, which can be combined with the topological argument from Alon et al. [7] to achieve the same ratio for phylogenies.

2 Notation and Preliminaries

Our dataset $C \subseteq \{0,1\}^d$ consists of n terminals over d binary coordinates. A Steiner tree (or phylogeny) over C consists of a tree T on the hypercube that spans C (plus possibly additional Steiner nodes), where we label each edge e in T with the index $i \in \{1,\ldots,d\}$ of the coordinate flipped on edge e. The cost of such a Steiner tree is the number of edges in the tree. Given a collection of datasets $\mathcal{P} = \{P_1, P_2, \ldots, P_k\} \subseteq C$ we define the Steiner forest problem as the problem of finding a minimal Steiner tree on every $P \in \mathcal{P}$ separately. We refer to such collection as a partition from now on, even though it may contain a subset of the original terminal set C.

In this work, we consider instances C whose minimum Steiner tree has cost $d+q$, and think of $q = o(\sqrt{d})$ (otherwise, any off-the-shelf constant approximation algorithm for the Steiner problem gives a solution of cost $\leq d + O(q^2)$). We fix T to be some optimal Steiner tree. By optimality, all leaves in T must be terminals, whereas the internal nodes of T may be either terminals or non-terminals (non-terminals are called *Steiner nodes*). We define a coordinate i to be *good* if exactly one edge in T is labeled i, and *bad* if two or more edges in T are labeled with i. We may assume all d coordinates appear in the tree, otherwise, some coordinates in C are fixed and so the dimensionality of the problem is less than d. Therefore, at most q coordinates are bad (each bad coordinate flips at least twice and thus adds a cost of at least 2 to the tree).

Given a coordinate i of a set of terminals P, we define an *i-cut* as the partition $P_0 = \{x \in P : \ x_i = 0\}$ and $P_1 = \{x \in P : \ x_i = 1\}$. We call two coordinates $i \neq j$ *interchangeable* if they define the same cut. We now present the following basic facts which are easy to verify (see [2] for proofs).

Fact 1

1. *Let S be a set of interchangeable coordinates. Then all coordinates in S appear together in the optimal tree T, adjacent to one another. That is, in T there are paths s.t. for each path: all of its edges are labeled by some $i \in S$, all coordinates in S have an edge on the path, and all internal nodes on the paths aren't terminals and have degree 2. On these paths, any reordering the S-labeled edges yields an equivalent optimal tree.*
2. *For any two good coordinates, $i \neq j$, one side of the i-cut is contained within one side of the j-cut. Equivalently, there exist values b_j such that all terminals on one side of the i-cut have their jth coordinate set to b_j.*
3. *Fix any good coordinate i and let j be a good coordinate such that all terminals on one side of the i-cut have their j coordinate set to b_j. Then both endpoints of the edge labeled i have their jth coordinate set to b_j.*

4. A good coordinate i and a bad coordinate i' cannot define the same cut.

It immediately follows from Fact 1 that for a given good coordinate i one can efficiently reconstruct the endpoints of the edge on which i mutates, except for at most q coordinates. This leads us to the following definition. Given i, we denote D^i as the set of all coordinates that are fixed to a constant value v_i on at least one side of the i-cut (different coordinates may be fixed on different sides), and we denote $\mathbf{b}^i$ as the vector of the corresponding values, i.e. v_i's, of the coordinates in D^i. The pair $(D^i, \mathbf{b}^i)$ is called the *pattern* of coordinate i. That set of terminals that *match the pattern* of i is the set $P_{\mathbf{b}^i} = \{x \in P : \ \forall j \in D^i, \ x_j = b^i_j\}$.

3 A Simple Case: Each Coordinate Determines a Distinct Cut

To show the main ideas behind our algorithm, we first discuss a special case in which no two coordinates i and j define the same cut on the terminal set C. Algorithms for constructing phylogenetic trees often make this assumption as they preprocess C by contracting any pair of interchangeable coordinates. However, in our case such contractions are problematic, as we discuss in the next section. So in Section 4, when we deal with the general case, we deal with interchangeable coordinates in a non-trivial fashion.

3.1 Basic Building Blocks

We now turn to the description of our algorithm. On a high level it is motivated by the notion of maintaining a proper partition of the terminals.

Definition 1. *Call a partition $\mathcal{P}$* proper *if the forest produced by restricting the optimal tree T to the components $P \in \mathcal{P}$ is composed of edge disjoint trees.*

Equivalently, the path in T between two nodes x and y in the same component P of $\mathcal{P}$ does not pass through any node x' in any different component P' of $\mathcal{P}$. Clearly, our initial partition, $\mathcal{P} = \{C\}$, is proper. Our goal is to maintain a proper partition of the current terminals while decreasing the dimensionality of the problem in each step. This is implemented by the two subroutines we now detail.

Pluck a Leaf and Paste a Leaf. The first subroutine works by building the optimal phylogeny bottom-up, finding a good coordinate i adjacent to a leaf terminal t in the tree, and replacing t with its parent (t with i flipped) in the set of terminals. Observe that if i is a good coordinate, then this removes the only occurrence of i, leaving all terminals in our new dataset with a fixed i coordinate, thus reducing the dimensionality of the problem by 1.

The matching subroutine to `Pluck-a-leaf` is `Paste-a-leaf`: if `Pluck-a-leaf` succeeds and returns some $(x, \mathcal{P}')$, and we have found a Steiner forest for the terminals in $\mathcal{P}'$. Then `Paste-a-leaf` merely connects x with $\bar{x}^i$ by an edge labeled i, then returns the resulting forest. (We omit formal description.)

`Pluck-a-leaf`
input: A partition $\mathcal{P}$ of current terminals.
if there exists $P \in \mathcal{P}$ and $x \in P$ s.t. some coordinate i is non-constant on P, but only the terminal x has $x_i = 0$ (or $x_i = 1$), then:
- Set $P' = P \setminus \{x\} \cup \{\bar{x}^i\}$, where $\bar{x}^i$ is identical to x except for flipping i.
- Return x and $\mathcal{P}' = \mathcal{P} \setminus \{P\} \cup \{P'\}$.

else fail.

Lemma 1. *If $\mathcal{P}$ is a proper partition and* `Pluck-a-leaf` *succeeds, then $\mathcal{P}'$ is a proper partition.*

Proof (Sketch). Let $T[P]$ be the subtree in which x resides. We claim that x is a leaf in $T[P]$, attached by an edge labeled i to the rest of the terminals. If this indeed is the case, then removing i means removing a leaf-adjacent edge from $T[P]$ which clearly leaves all components in the forest edge-disjoint.

Wlog x lies on the $i = 0$ side of the cut. If x isn't a leaf, then at least two disjoint paths connect x to two other terminals. Since $\mathcal{P}$ is proper, both these terminals are in P. This means $T[P]$ crosses the i-cut twice, but then we can replace $T[P]$ with an even less costly tree in which i is flipped once, by projecting the path between the two occurrences of i onto the $i = 1$ side.

□

Observe that lemma 1 holds only when the underlying alphabet of the problem is binary. In particular, for a non-binary alphabet, such x can be a non-leaf.

Split and Merge. When `Pluck-a-leaf` can no longer find leaves to pluck, we switch to the second subroutine, one that works by splitting the set of terminals into two disjoint sets, based on the value of the i-th coordinate. We would like to split our set of terminals according to the i-cut, and recurse on each side separately. But, in order to properly reconnect the two subproblems, we need to introduce the two endpoints of the i-labeled edge to their respective sides of the i-cut. Our `Split` subroutine deals with one particular case in which these endpoints are easily identified.

`Split`(i)
input: A partition $\mathcal{P}$ of current terminals, a coordinate i that is not constant on every component of $\mathcal{P}$.

- Find a component P on which i isn't constant. Denote the i-cut of P as (P_0, P_1).
- Find $P_{\mathbf{b}^i}$, the set of terminals that match the pattern of i.
- **if** exists some x which is the *unique* terminal that matches the pattern of i in one side of the cut (that is, if for some x we have $P_{\mathbf{b}^i} \cap P_0 = \{x\}$ or $P_{\mathbf{b}^i} \cap P_1 = \{x\}$)
  - Flip the i-th coordinate of x, and let $\bar{x}^i$ be the resulting node.
  - Add x to its side of the i-cut, add $\bar{x}^i$ to the other side of the cut.
  - Return $x, \bar{x}^i$ and $\mathcal{P}' = \mathcal{P} \setminus \{P\} \cup \{P_0, P_1\}$.

  else fail.

The matching subroutine to `Split` is `Merge`: Assume `Split` succeeds and returns some $(x, \bar{x}^i, \mathcal{P}')$, and assume we have found a Steiner forest for the terminals in $\mathcal{P}'$. Then `Merge` merely connects x with $\bar{x}^i$ by an edge labeled i, then returns the resulting forest. (Again, formal description is omitted.)

Lemma 2. *Assume $\mathcal{P}$ is a proper partition. Assume* `Split` *is called on a good coordinate i s.t. the edge labeled i in T has at least one endpoint which is a terminal. Then the returned partition $\mathcal{P}'$ is proper.*

Proof (Sketch). Since $\mathcal{P}$ is proper, then the induced tree $T[P]$ is the only tree in the forest that contains the i-labeled edge. The lemma then follows from showing that x and $\bar{x}^i$ are the two endpoints of i-labeled edge in $T[P]$. This follows from the observation that the endpoints of the i-labeled edge must both match the pattern of i. Let u be an endpoint and wlog u belongs to the $(i = 0)$-side of the cut. On all coordinates that are fixed on the $(i = 0)$-side, u obviously has the right values. All coordinates that are fixed on the $(i = 1)$-side can only flip on the $(i = 0)$-side, but only after traversing u, so u has them set to the value fixed on the $(i = 1)$-side.

□

3.2 The Algorithm

We can now introduce our algorithm.

input: A partition $\mathcal{P}$ of current terminals. Initially, $\mathcal{P}$ is the singleton set $\mathcal{P} = \{C\}$.

1. **if** `Pluck-a-leaf` succeeds and returns $(x, \mathcal{P}')$
   - recurse on $\mathcal{P}$', then `Paste-a-leaf` x back and return the resulting forest.
2. **else-if** the number of non-constant coordinates on $\mathcal{P}$ is at least $40q^2$
   - Pick a non-constant coordinate i u.a.r and invoke `Split`(i) .
   - **if** `Split` succeeds: recurse on $\mathcal{P}'$, then `Merge` x and $\bar{x}^i$, and return the resulting forest; **otherwise** fail.
3. **else**
   - For every $P \in \mathcal{P}$ find its MST, $T(P)$, and return the forest $\{T(P)\}$.

Fig. 1. Algorithm for the simple case

Theorem 2. *With probability $\geq 1/2$, the algorithm in Figure 1 returns a tree whose cost is at most $d + O(q^2)$.*

In order to prove Theorem 2, fix an optimal phylogeny T over our initial set of terminals, and for any partition $\mathcal{P}$ our algorithm creates, denote $T[\mathcal{P}]$ as the forest induced by T on this partition. The proof of the theorem relies on the following lemma.

Lemma 3. *If $\mathcal{P}$ is a proper partition, then with probability $\geq 1 - (8q)^{-1}$,* `Split` *is called on a good coordinate and succeeds. Furthermore,* `Split` *is executed at most $4q$ times.*

Proof (of Theorem 2). The proof follows from lemmas 1 and 3. Since we start with a proper partition, then with probability at least $1 - (4q)(8q)^{-1} \geq 1/2$ we keep recursing on proper partitions, until reaching the base of the recursion. By the time the algorithm reaches the base of the recursion, the dimensionality of the problem was reduced to $d' \leq 40q^2$, so the cost of the optimal Steiner forest is at most $d' + q$. As MSTs give a 2-approximation to the optimal Steiner tree problem, our forest is of cost $\leq 2(d' + q)$. Then, the algorithm reconnects the forest, adding the coordinates (edges) the algorithm as removed in the first two steps of the algorithm. Since the algorithm removed at most $d - d'$ edges, the tree it outputs is of overall cost at most $d - d' + 2(d' + q) = d + 40q^2 + 2q$.

□

Proof (of Lemma 3). Let $\mathcal{P}$ be the partition in the first iteration of the algorithm for which `Split` was invoked, and assume $\mathcal{P}$ is proper. Thus, the forest $T[\mathcal{P}]$ contains disjoint components. We call any vertex in this forest of degree ≥ 3 an *internal split*. Suppose we replace each internal split v with $deg(v)$ many new vertices, each adjacent to one edge. This breaks the forest into a collections of paths we call the *path decomposition* of the tree. In addition, remove from this path decomposition all edges that are labeled with a bad coordinate to obtain the *good path decomposition*. Denote the number of paths in the good path decomposition as t.

First, we claim that any call to `Split` (on $\mathcal{P}$ or any partition succeeding $\mathcal{P}$), on a coordinate i which lies on a path of length ≥ 2 in the abovementioned decomposition, does not fail.

Assume `Split` was called on i and denote its adjacent coordinate on the path as j (choose one arbitrarily if i has two adjacent coordinates on its path), and both are non-constant on $P \in \mathcal{P}$. Observe that our decomposition leaves only good coordinates, so both i and j are good. Therefore, j is fixed on one side of the i-cut and i is fixed on one side of the j-cut. It follows that there exist binary values b_i, b_j s.t. for every $x \in P$, if $x_i = b_i$ then $x_j = b_j$; and if $x_j = 1 - b_j$ then $x_i = 1 - b_i$. In fact, the only node on the entire tree for which $x_i = 1 - b_i$ and $x_j = b_j$ is the node connecting the i-edge and the j-edge. Recall that we assume for the special case i and j do not define the same cut. It follows that the node between i and j has to be a terminal, so now we can use Lemma 2 and deduce `Split` succeeds.

So, `Split` can either fail or return a non-proper partition only if it was invoked either on a bad coordinate or on a good coordinate that lies on a path of length 1 in our path decomposition. There are at most q bad coordinates and at most t paths of length 1, so each call to `Split` fails w.p. $\leq \frac{q+t}{40q^2}$. Furthermore, calling `Split` on a good edge i lying on a path of length at least 2 results in both i's endpoints as new leaves in their respective sides of the i-cut. As a result, `Pluck-a-leaf` then completely unravels the path on which i lies. Therefore, in a successful run of the algorithm, `Split` is called no more than t times. All that remains is to bound t.

t is the number of paths on the path decomposition of $\mathcal{P}$, a partition for which `Pluck-a-leaf` failed to execute. Observe that if the forest $T[\mathcal{P}]$ had even a single

leaf connected to the rest of its tree by a good coordinate, then `Pluck-a-leaf` would continue – such a leaf, by definition, is the only terminal on which the good coordinate takes a certain value. It follows that l, the number of leaves in $T[\mathcal{P}]$ is bounded by $2q$, the number of bad edges in T. Removing the internal splits then leaves us with at most $2l$ paths; removing the bad coordinates' edges adds at most $2q - l$ new paths (for every bad coordinate k adjacent to a leaf, removing k does not create a new path). All in all, $t \leq 2l + 2q - l \leq 4q$. Therefore, each call to `Split` has success probability $\geq 1 - \frac{4q+q}{40q^2} = 1 - \frac{1}{8q}$, and `Split` is called at most $4q$ times.

□

4 The General Case: Interchangeable Coordinates May Exist

Before describing the general case, let us briefly discuss why the conventional way of initially contracting all interchangeable coordinates and applying the algorithm from the Section 3 might result in a tree of cost $d+\omega(q^2)$. The analysis of the first two steps of the algorithm still holds. The problem lies in the base of the recursion, where the algorithm runs the MST-based 2-approximation. Indeed, the MST algorithm is invoked on $< 40q^2$ contracted coordinates, but they correspond to $\tilde{d}$ original coordinates, and it is possible that $\tilde{d} \gg q^2$. So by using any constant approximation on this entire forest, we may end with a tree of cost $d + 2\tilde{d}$ which isn't $d + O(q^2)$.

Our revised algorithm does not contract edges initially. Instead, let us define a *simple* coordinate as one for which `Split`(i) succeeds. So, the first alteration we make to the algorithm is to call `Split` as long as the set of simple coordinates is sufficiently big. However, most alterations lie in the base of the recursion. Below we detail the algorithm and analyze its correctness. In the algorithm's description, for any coordinate i we denote the set of coordinates interchangeable with i by W_i, and their number as $w(i) = |W(i)|$.

Theorem 3. *With probability $\geq 1/2$, the algorithm in Figure 2 returns a tree of cost $d + O(q^2)$.*

The proof of Theorem 3 follows the same outline as the proof of Theorem 2. Observe that Lemmas 1 and 3 still hold[2]. Therefore, with probability $\geq 1/2$, the algorithm enters the base of the recursion with a proper partition. Thus, by the following lemma, the algorithm outputs a tree of cost $d + O(q^2)$.

Lemma 4. *Assume that the base of the recursion (i.e., Step 3) is called on a proper partition $\mathcal{P}$ of the terminals over d' non-constant coordinates. Then the algorithm returns a forest of cost $d' + O(q^2)$.*

The full proof of Lemma 4 is deferred to the appendix. However, let us sketch the main outline of the proof. Recall the good path decomposition we used in the proof of Lemma 3. We partition its paths in the following way.

[2] Clearly, `Split` cannot abort now, but it might be the case that the algorithm picks i which is a bad coordinate. This can happen with probability $\leq q/8q^2 = 1/8q$.

input: A partition $\mathcal{P}$ of current terminals. Initially, $\mathcal{P}$ is the singleton set $\mathcal{P} = \{C\}$.

1. **if** `Pluck-a-leaf` succeeds and returns $(x, \mathcal{P}')$
   - recurse on $\mathcal{P}$', then `Paste-a-leaf` x and return the resulting forest.
2. **else-if** the number of simple coordinates on $\mathcal{P}$ is at least $8q^2$
   - Pick a simple coordinate i u.a.r and invoke `Split`(i) .
   - **if** `Split` succeeds: recurse on $\mathcal{P}'$, then `Merge` x and $\bar{x}^i$, and return the resulting forest; **otherwise** fail.
3. **else**
   - Contract all W_i into $\bar{i}$
   - For every $\bar{i}$ with $w(i) > q$ and the (unique) component P in which the i-cut resides,
     - Apply pattern matching to (P, i). Let $(D^i, \mathbf{b}^i)$ be the pattern of i.
     - **if** i is simple, split P into $P_0 \cup \{x^i\}$ and $P_1 \cup \{\overline{x}^i\}$.
     - **else**
       - Define the node $y(i)$ as the node where $y_i = 0$, every coordinate $j \in D^i$ is set to b^i_j, and every coordinate $j \notin D^i$ is set to 0.
       - Define $\overline{y(i)}$ to be $y(i)$ with coordinate i flipped.
       - $\mathcal{P} = \mathcal{P} \setminus \{P\} \cup \{P_0 \cup \{y(i)\}\} \cup \{P_1 \cup \{\overline{y(i)}\}\}$.
   - For every $P \in \mathcal{P}$ find its MST $T(P)$, and retrieve the forest $\{T(P)\}$.
   - For every i with $w(i) > q$:
     if i was simple, add an edge labeled i between x^i and $\overline{x^i}$
     else add an edge labeled i between $y(i)$ and $\overline{y(i)}$.
   - Expand all contracted coordinates to their original set of coordinates by replacing i with a path of length $w(i)$. Return the resulting forest.

Fig. 2. Algorithm for the general case

- *Paths with at least one terminal on them.* On such paths, because all interchangeable coordinates may appear in T in any order, then all coordinates on such paths are simple. So, when we enter the base of the recursion, there are at most $8q^2$ edges on such paths.
- *Paths with no terminal on them, with length* $> q$. Such paths are composed of interchangeable coordinates, and since there are more than q of those, we deduce all of them are good. Therefore, the endpoints of such paths are fixed up to at most q (bad) coordinates. We therefore contract these edges, split on them, and introduce into each side of the cut an arbitrary endpoint, by replacing non-fixed coordinates with zeros. So on each side of the cut the cost of the subtree increases by at most q, and since there are at most $4q$ such paths, our overall cost for introducing these artificial endpoints is $O(q^2)$.
- *Paths with no terminal on them, with length* $\leq q$. Such paths are composed of interchangeable coordinates, but we do not contract them. Since there are at most $4q \cdot q$ edges on such paths, we run the MST approximation, and incur a cost of $O(q^2)$ for edges on such paths.

Runtime analysis: `Pluck-a-leaf` can be implemented in time linear in the size of the dataset, i.e. $O(nd)$. Counting the number of simple coordinates takes time $O(nd^2)$, and `Split` takes time $O(nd)$. A naive implementation of the base case of the recursion takes time $O(nd^2)$ for contracting coordinates, and the rest

can be implemented in time $O(nd)$. Hence the time to process each node in the recursion tree is at most $O(nd^2)$. Since there are at most $O(q)$ nodes in the recursion tree, the total runtime is $O(qnd^2)$.

5 Discussion and Open Problems

This paper presents a randomized approximation algorithm for constructing near-perfect phylogenies. In order to achieve this, we obtain a Steiner tree of low additive error. However, from the biological perspective, the goal is to find a good evolutionary tree, one that will give correct answers to questions like "what is the common ancestor of the following species?" or "which of the two gene-mutations happened earlier?". Such questions, we hope, can be answered by finding the most-parsimonious phylogenetic tree over the given taxa. Hence, it is also desirable that any low-cost tree which we output also captures a lot of the structure of the optimal tree.

We would like to point out that our algorithm in fact has this valuable property. Notice that until the base case of the recursion, both `Pluck-a-leaf` and `Split` subroutines construct the optimal tree, and correctly identify the endpoints of the edges they remove. Even when the algorithm reaches the base case of the recursion – we can declare every edge of weight $> q$ to be good, and we know its endpoints up to at most q coordinates. In total, our algorithm gives the structure of the optimal tree up to $O(q^2)$ edges, and those edges can be marked as "unsure".

Several open problems remain for this work. The most straight-forward one is whether one can devise an algorithm outputting a phylogenetic tree of cost $d + O(q)$? Alternatively, one may try to design exact algorithms that are efficient even for $q = \omega(\log d)$. We suspect that even the case of $q = O((\log d)^2)$ poses quite a challenge. Finally, extending our results to non-binary alphabets is intriguing. Note however that even the case of perfect phylogenies is NP-hard, and tractable only for moderately sized alphabets. Furthermore, our bottom-up approach completely breaks down for non-binary alphabets (see comment past Lemma 1), so devising an additive-approximation algorithm for the phylogeny problem with non-binary alphabets requires a different approach altogether.

References

1. Ding, Z., Filkov, V., Gusfield, D.: A linear-time algorithm for the perfect phylogeny haplotyping (pph) problem. In: RECOMB, Springer (2005) 585–600
2. Blelloch, G.E., Dhamdhere, K., Halperin, E., Ravi, R., Sridhar, S.: Fixed parameter tractability of binary near-perfect phylogenetic tree reconstruction. In: ICALP, Springer (2006) 667–678
3. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press (1997)
4. Semple, C., Steel, M.: Phylogenetics. Oxford lecture series in mathematics and its applications. Oxford University Press (2003)

5. Hinds, D.A., Stuve, L.L., Nilsen, G.B., Halperin, E., Eskin, E., Ballinger, D.G., Frazer, K.A., Cox, D.R.: Whole-genome patterns of common dna variation in three human populations. Science **307**(5712) (2005) 1072–1079
6. : The international hapmap project. Nature **426**(6968) (2003) 789–96
7. Alon, N., Chor, B., Pardi, F., Rapoport, A.: Approximate maximum parsimony and ancestral maximum likelihood. IEEE/ACM Trans. Comput. Biol. Bioinformatics **7** (Jan 2010) 183–187
8. Robins, G., Zelikovsky, A.: Improved steiner tree approximation in graphs. In: SODA, Society for Industrial and Applied Mathematics (2000) 770–779
9. Robins, G., Zelikovsky, A.: Improved steiner tree approximation in graphs (2000)
10. Robins, G., Zelikovsky, A.: Tighter bounds for graph steiner tree approximation. SIAM Journal on Discrete Mathematics **19** (2005) 122–134
11. Misra, N., Blelloch, G.E., Ravi, R., Schwartz, R.: Generalized buneman pruning for inferring the most parsimonious multi-state phylogeny. In Berger, B., ed.: RECOMB. Volume 6044 of Lecture Notes in Computer Science., Springer (Apr. 2010) 369–383
12. Fernández-Baca, D., Lagergren, J.: A polynomial-time algorithm for near-perfect phylogeny. SIAM J. Comput. **32** (May 2003) 1115–1127
13. Karp, R.M.: Reducibility among combinatorial problems. In Miller, R.E., Thatcher, J.W., eds.: Complexity of Computer Computations. Plenum, New York (1972) 85–103
14. Foulds, L.R., Graham, R.L.: The Steiner problem in phylogeny is NP-complete. Adv. Appl. Math. 3 (1982)
15. Sridhar, S., Dhamdhere, K., Blelloch, G., Halperin, E., Ravi, R., Schwartz, R.: Algorithms for efficient near-perfect phylogenetic tree reconstruction in theory and practice. IEEE/ACM Trans. Comput. Biol. Bioinformatics **4** (Oct 2007) 561–571
16. Damaschke, P.: Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. Theor. Comput. Sci. **351** (Feb 2006) 337–350
17. Agarwala, R., Fernandez-Baca, D.: A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. In: SFCS. (Nov 1993) 140–147
18. Byrka, J., Grandoni, F., Rothvoß, T., Sanità, L.: An improved lp-based approximation for steiner tree. In: STOC, ACM (2010)
19. Bodlaender, H.L., Fellows, M.R., Warnow, T.: Two strikes against perfect phylogeny. In: ICALP'92. (1992) 273–283
20. Takahashi, H., Matsuyama, A.: An approximate solution for the steiner problem in graphs. Mathematica Japonica **24** (1980) 573–577
21. Berman, P., Ramaiyer, V.: Improved approximations for the steiner tree problem. In: SODA. (1992) 325–334
22. Pramel, H., Steger, A.: Rnc-approximation algorithms for the steiner problem. In Reischuk, R., Morvan, M., eds.: STACS 97. Volume 1200 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (1997) 559–570 10.1007/BFb0023489.
23. Karpinski, M., Zelikovsky, A.: New approximation algorithms for the steiner tree problems. Journal of Combinatorial Optimization **1** (1995) 47–65
24. Zelikovsky, A.: Better approximation bounds for the network and euclidean steiner tree problems. Technical report (1996)
25. Hougardy, S., Promel, H.J.: A 1.598 approximation algorithm for the steiner problem in graphs. In: SODA. (1999) 448–453
26. Borchers, A., Du, D.Z.: The k-steiner ratio in graphs. In: STOC, ACM (1995) 641–649

A Appendix

Here, we give short proofs of the basic properties presented in Fact 1. Many of these results are also found in other work [2].

1. Let S be a set of coordinates that define the same cut over the terminals (up to renaming of P_0 and P_1). Then whenever any edge in T is labeled with some $i \in S$, then the edge lies on a path on which each edge is labeled with a unique coordinate from S, and no node of the $|S|$-long path is a terminal.
   Proof. Assume $i, j \in S$ define the same cut over C. Given some edge e_i where i flips, consider the shortest path from a terminal t_1 through e_i to another terminal t_2. If every node on this path is of degree 2, there is no terminal before t_2, and j flips before t_2. Now, suppose some node on the path between t_1 and t_2 has degree more than 2. Either j flips on each outgoing path before any terminals, or j does not define the same cut (since each outgoing path has a terminal on it). But if j flips on all outgoing paths before any terminal occurs on those paths, relabel the Steiner nodes on each path so that j is constant along those outgoing paths. Then, add one Steiner node and an edge from the endpoint of e_i which flips j. This tree has cost strictly less than the tree which flipped j on each outgoing path, since there were at least two paths, each of which flipped j.
   □
2. For any two good coordinates, $i \neq j$, one side of the i-cut is contained within one side of the j-cut. Alternatively, there exist values b_j such that all terminals on one side of the i-cut have their jth coordinate set to b_j.
   Proof. Suppose i is good. Then, consider the i-cut in T. Since j is good, j may flip only once in the tree. If j flips in C_0, then j is constant in C_1. If j flips in C_1, then j is constant in C_0.
   □
3. Fix any good coordinate i and let j be a good coordinate such that all terminals on one side of the i-cut have their j coordinate set to b_j. Then both endpoints of the edge labeled i have their j^{th} coordinate set to b_j.
   Proof. Suppose that some endpoint of the edge on which i flips has j (which is constant on C_0) set to $\bar{b}_j$. Then, since the edge allows only one coordinate to flip across it, both endpoints are labelled with $j = \bar{b}_j$. Then, the side where j is constant (say C_0) has to pay for j.
   If j is constant and set to b_j on C_1, j has to flip twice, a contradiction since j is good. If j is constant and set to $\bar{b}_j$ on C_0, then the labels on i's edge are labelled by some constant setting of j. If j is non-constant Assuming j is non-constant for C_1, j flips somewhere in C_1. But then, j flips twice, contradicting the fact that j is good.
   □
4. A good coordinate i and a bad coordinate i' cannot define the same cut.
   Proof. This follows directly from Fact 2.1, since i and i' will occur the same number of times in an optimal tree and a good coordinate occurs exactly once, while a bad coordinate occurs at least twice.
   □

B Proof of Lemma 4

Here we give the full proof of Lemma 4. For convenience, we reiterate the lemma.

Lemma 5. *Assume that when step 3 is entered, $\mathcal{P}$ is a proper partition of the terminals over d' non-constant coordinates. Then step 3 of the algorithm returns a forest of cost $d' + O(q^2)$.*

Proof. Recall that for any coordinate i, we define the weight of i as the total number of coordinates interchangeable with i. Let $\mathcal{P}^{\text{post}}$ denote the instance which results after splitting on all coordinates with weight greater than q. The proof of the lemma reduces to showing that there exists a forest over $\mathcal{P}^{\text{post}}$ of cost $O(q^2)$. If such a forest exists, the MST-based 2-approximation for each component returns a forest of cost $O(q^2)$, and reconstructing the tree with plucked and split edges has weight at most d'. So, the forest over $\mathcal{P}$ we get has cost $\leq d' + O(q^2)$.

The set of all terminals in all components of $\mathcal{P}^{\text{post}}$ is composed of terminals that belong to the original $\mathcal{P}$, and the new terminals, added by coordinates of weight more than q which are *not* simple. We construct a forest of low cost over $\mathcal{P}^{\text{post}}$ by (i) taking an optimal Steiner forest $\mathcal{T}$ over $\mathcal{P}$, (ii) for every simple coordinate which is split upon, remove the path of corresponding coordinates from $\mathcal{T}$, and (iii) for every non-simple coordinate split upon, remove the path of corresponding coordinates *and* introduce the two vertices $y(i)$ and $\overline{y(i)}$, connecting each vertex to the end-point of the path that resides in the same side of the cut. We denote the resulting forest by $\mathcal{T}^{\text{post}}$, and show that $\mathcal{T}^{\text{post}}$ contains at most $O(q^2)$ edges.

First, observe that since there are at most q bad coordinates, by splitting only on coordinates with weight more than q, we are guaranteed to split only on good coordinates (we know from Fact 1 that the good and bad coordinates cannot define the same cut). Therefore, since $\mathcal{P}$ is proper, the coordinate i resides in a single component. Second, observe that the base of the recursion is executed only when `Pluck-a-leaf` fails. Therefore, as in the proof of Lemma 3, $\mathcal{T}$ is a forest with at most $2q$ leaves, so its path decomposition contains at most $4q$ paths. Our next observation is the following proposition.

Proposition 1. *Fix a path in the path decomposition of the forest. If there exists some non-endpoint terminal on this path, all coordinates on this path are simple.*

Proof. Let i be a coordinate associated with an edge on such a path. Let x^i be a terminal on the path which is the closest to i. Recall the observation from before, that any two coordinates that yield the same terminal cut must appear in the optimal tree adjacent to one another, and furthermore, their order does not matter (see Fact 1). Therefore, we may assume i is adjacent to x^i. Furthermore, as x^i is not an endpoint, there exists a good coordinate $j \neq i$ adjacent to x^i on this path. It follows that i and j determine two different cuts over the set of terminals. Then, just as in the proof of Lemma 3, i is simple, where x^i is the unique terminal on one side of its cut matching i's pattern.

□

Proposition 1 means that any non-simple coordinate corresponds to an entire path in the path decomposition (because all edges on a path with no terminals on it determine the same cut). We deduce that (a) the number of non-simple contracted coordinates of weight more than q is at most $8q$. Furthermore, the number of edges on any path of length no more than q with no terminal on them is at most $8q^2$. Finally, since the base of the recursion is executed only when `Split` fails to execute, the number of edges on paths that do have a terminal on them is at most $16q^2$. It follows that the path decomposition of $\mathcal{T}$ contains at most $16q^2$ edges, in addition to the edges on paths of length more than q with no terminal on them (and each such path causes the algorithm to split exactly once).

We can now upper bound the number of edges in $\mathcal{T}^{\text{post}}$. The forest $\mathcal{T}^{\text{post}}$ contains at most $2q \leq 2q^2$ bad edges; at most $16q^2$ edges from $\mathcal{T}$; and all the paths between the vertices we introduce by adding the $y(i)$-vertices and connecting them to $\mathcal{T}$. Let i be a non-simple contracted coordinate which was split upon, and let $u \to v$ be the corresponding path on $\mathcal{T}$ which was removed. Assume $y(i)$ resides on the same side of the cut as u. Clearly, u and $y(i)$ identify on all coordinates on the $u \to v$ path, but they also identify on all other good coordinates: a good coordinate j appears most once in $\mathcal{T}$, so u, v and all terminals on at least one side of the $u - v$ cut has the same value on their jth coordinate. It follows that the path between $y(i)$ and u is of length at most q, and the same applies to the path between $\overline{y(i)}$ and v. Therefore, the number of edges on paths we have yet to upper bound, sum to no more than $8q \cdot 2 \cdot q = 16q^2$. Thus $\mathcal{T}^{\text{post}}$ contains no more than $34q^2$ edges. This completes the proof.

□