Additive Approximation for Near-perfect Phylogeny Construction

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

* **Carnegie Mellon University**
Pittsburgh, PA
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 [9] if the cost of the optimal phylogeny is exactly d (a perfect phylogeny). Moreover, if the data is a near-perfect phylogeny, i.e., the cost of the optimal tree is $d + q$, it is known that an exact solution can be found in running time which is polynomial in the number of species and d, but exponential in q [5]. In this work, we study the case of near-perfect phylogenies, where the optimal phylogenetic tree has cost $d+q$, and give a polynomial-time algorithm (polynomial in both d and q) that finds a phylogenetic tree of cost $d + O(q^2)$. We also discuss the motivation and reasoning for studying such additive approximations.

1998 ACM Subject Classification F.2.0

Keywords and phrases Phylogeny, Steiner Tree, Additive Approximation

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 ([12, 22]). A typical choice for these features are single nucleotide polymorphisms (SNPs), binary indicator variables for common mutations found in DNA[13, 1]; see, for example, [5, 9, 3, 19, 20, 21]. This challenging problem has attracted much attention in recent years, with progress in studying various computational formulations of this problem ([12, 17, 5, 9, 10, 3]). 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 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 [15], and remains NP-hard even in the case of a binary alphabet with the metric induced by the Hamming distance [11]. Extensive recent work, both experimental and theoretical, has focused on the binary character set with the Hamming metric ([12, 5, 9, 10, 3, 22, 23, 8]). Hence, 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). If a dataset admits a perfect phylogeny, an optimal tree can be constructed in polynomial time [2] (even linear time, in the case where the alphabet is binary[12]). 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 ([17, 5, 10, 8]) and experimental settings ([23]).

The work of [17, 5, 10, 8] has given a series of randomized algorithms which find the optimal phylogeny in running time polynomial in n and d but exponential in q. When $q = O(\log d)$, these exact algorithms are tractable. However, for $q = \omega(\log d)$, these algorithms do not run in polynomial time. For such larger values of q, one could run a generic Steiner tree approximation algorithm (being careful because the hypercube has exponential size) to get an approximately-optimal solution—see, e.g., Alon et al. [3]. The best current such algorithm will yield a tree of cost at most $1.39(d + q)$ [7]. Yet for small values of q, this increase over optimal can be quite large as a function of q. In this work, we present a $\text{poly}(n, d, q)$-time algorithm that finds an approximate near-perfect phylogeny 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 $\text{poly}(n, d, q)$-time, randomized algorithm, that finds a phylogenetic tree of cost $d + O(q^2)$.*

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 the Steiner tree problem.

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 non-canonical 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 [11]. Schwartz et al. [17] 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 [9] and in polynomial time for a fixed alphabet [10]. Recent work [5] 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 [5] is exponential in the number of additional mutations. To the authors' knowledge, neither a polynomial-time (in the number of additional mutations) algorithm nor a hardness result for the near-perfect case is known.

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 [24, 4, 18, 16, 25, 19, 14, 20, 21, 7]. The more recent of these papers use a result due to Borchers and Du [6] 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. The line of work started by Robins and Zelikovsky [20, 21] which achieved an approximation ratio of 1.55 used the notion of k-restricted Steiner trees.

Alon et al. [3] 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. [7] 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. [3] 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, a labeling of the nodes mapping each node to a (unique) d-long binary string such that C is a subset of the nodes, and the labels of any two adjacent nodes differ on exactly one coordinate. One can think of this as a labeling on the edges, indicating which coordinate is flipped between the two endpoints of the edge. The cost of such a Steiner tree is the number of edges in the tree. Given a collection of datasets $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ we define the Steiner forest problem as the problem of finding a minimal Steiner tree on every $C \in \mathcal{C}$ separately.

In this work, we consider instances C whose minimum Steiner tree has cost $d + q$, and think of $q < \sqrt{d}$ (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^{opt} to be some optimal Steiner tree (abbreviated as T when context is clear). 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 in C, we define an *i-cut* as the partition $C_0 = \{x \in C : \ x_i = 0\}$ and $C_1 = \{x \in C : \ x_i = 1\}$. We now present the following basic facts which are easy to verify. (see [5] and our Appendix for proofs)

▶ Fact 2.

1. Let S be a set of coordinates that define the same cut over the terminals (up to renaming of C_0 and C_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.

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

It immediately follows from Fact 2 that for a given good coordinate i, except for at most q coordinates, one can efficiently reconstruct the endpoints of the edge in the optimal tree on which i mutates.

3 A Simple Case: Each Coordinate Determines a Unique 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 phylogenies often make this assumption and preprocess C by contracting any pair of coordinates that determine the same cut over C. In our case, however, such contractions are problematic, as we discuss in the next section. In Section 4 we remove this assumption.

Our algorithm works by building the optimal phylogeny bottom-up, finding a good coordinate i adjacent to a leaf terminal t in the tree, removing t from the set of terminals, and adding to the set of terminals, t's parent (by flipping i, which reduces the dimensionality of the problem by 1). When all the leaves of the current tree are adjacent to bad coordinates, there is no good coordinate i adjacent to a leaf, and the bottom-up approach fails. The algorithm then randomly picks a coordinate i, and separates the problem into two sub-problems, induced from the cut determined by i. We will show that with high probability, the algorithm can determine the *unique* endpoints of the edge in the optimal tree where i is flipped. The algorithm then adds each endpoint to its appropriate subproblem, and recurs. Finally, when the dimensionality of all sub-problems together is too small (the algorithm is no longer likely to pick i such that it can determine i's endpoints), the algorithm simply runs the minimum spanning tree (MST) based 2-approximation algorithm for Steiner tree (or any poly(d, n, q) constant-approximation algorithm) on each sub-problem. The subproblems are reconnected by the previous randomly chosen coordinates. Formally, we first define the pattern-matching procedure, then describe the algorithm in detail in Figure 1.

▶ **Definition 3.** Given a set of terminals C and a coordinate i, the following procedure is called *pattern matching*: (i) find all coordinates $j \neq i$ that are fixed on at least one side of the i-cut. Denote this set by D, and let $\mathbf{b}^i$ be the vector of values of the coordinates in D (on the side where they are fixed). (ii) find the set of terminals $C_{\mathbf{b}^i} = \{x \in C : \ \forall j \in D, \ x_j = b^i_j\}$. We call D and $\mathbf{b}^i$ the *pattern* of coordinate i.

▶ **Theorem 4.** *With probability $\geq 1/2$, the algorithm in Figure 1 returns a tree whose cost is at most $d + O(q^2)$.*

To prove Theorem 4, fix an optimal phylogeny T over our initial set of terminals. We call a partition $\mathcal{C}$ *proper* if the induced subgraphs resulting from restricting T to the different components $C \in \mathcal{C}$ are edge disjoint. Observe that initially $\mathcal{C}$ contains all the terminals and

Input: A set $\mathcal{C}$, such that each component $C \in \mathcal{C}$ is a subset of the terminals. Initially, $\mathcal{C}$ is the singleton set, containing all terminals.

1. **Pluck-a-leaf step:**
   if there exists $C \in \mathcal{C}$, and a coordinate i such that i is non-constant in C and exactly one terminal $x \in C$ has $x_i = 0$ (or exactly one terminal has $x_i = 1$), then:
   - Replace x in $\mathcal{C}$ with $\bar{x}$, which is identical to x except for coordinate i, which is flipped.
   - Recur on $\mathcal{C}$.
   - In the forest returned, let $T_{\bar{x}}$ be the tree that contains $\bar{x}$. Connect T and x with coordinate i leaving $\bar{x}$ and ending with x.
2. **Split step:**
   else-if the number of coordinates (which are not constant on some $C \in \mathcal{C}$) is $\geq 60q^2$, then:
   - Pick i uniformly at random from all non-constant coordinates on $\mathcal{C}$.
   - Split the unique component C in which i resides into C_0 and C_1 and apply the **pattern-matching** procedure to i to obtain $C_{\mathbf{b}^i}$.
   - If $| C_{\mathbf{b}^i} \cap C_0 | = 1$ or $| C_{\mathbf{b}^i} \cap C_1 | = 1$, then:
     - Let x be this unique terminal in C_0(or C_1), and let $\bar{x}$ be this terminal with x_i flipped.
     - Let $C_1 = \{\bar{x}\} \cup C_1$ (or $C_0 = \{\bar{x}\} \cup C_0$).
     - Recur on $\mathcal{C} \setminus \{C\} \cup \{C_0, C_1\}$. Let T_0 and T_1 be the two trees on C_0 and C_1 resp.
     - Merge T_0 and T_1 into a single tree by connecting x and $\bar{x}$ on an edge labeled i.
   - Otherwise **Abort**.
3. **Base of recursion step:**
   else
   - For every $C \in \mathcal{C}$ find its MST, $T(C)$, and return the forest $\{T(C)\}$.

Figure 1 Algorithm for the simple case

hence is proper. The proof of Theorem 4 follows from showing that the first and the second steps of the algorithm return a proper partition with high probability.

▶ Claim 5. If $\mathcal{C}$ is a proper partition, then the pluck-a-leaf step of the algorithm returns a proper partition.

▶ Claim 6. If $\mathcal{C}$ is a proper partition, then with probability $\geq 1 - (8q)^{-1}$, the split step of the algorithm returns a proper partition. Furthermore, the split step is executed at most $4q$ times.

Proof of Theorem 4. The proof follows from the above two claims. Using the union bound, the probability that any execution of the split step either aborts or returns a non-proper partition is at most $1/2$. So, with probability $\geq 1/2$ the algorithm reaches step 3 of the algorithm with a proper partitioning of the tree. By the time the algorithm reaches step 3 the algorithm has reduced the dimensionality of the problem to $d' \leq 60q^2$, so the cost of the optimal Steiner forest is at most $d' + q$. Therefore, our MSTs have total 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 $d+q-d'$ edges, the tree it outputs is of overall cost at most $d - d' + q + 2(d' + q) = d + O(q^2)$. ◀

Proof of Claim 5. Recall that the pluck-a-leaf step picks coordinate i from the non-constant coordinates of C because there was exactly one terminal $x \in C$ with $x_i = 0$ (without loss

of generality). Let $T[C]$ denote the restriction of the optimal tree T to the terminals in C. Clearly, i must appear in $T[C]$, since it is non-constant over the terminals in C. Furthermore, i must cross the cut $(x, C - x)$.

Assume i crosses this cut more than once. Removing all edges in $T[C]$ labeled i splits the tree into several subtrees, where on the $i = 0$ side, there is only a single terminal, x. Furthermore, as $\mathcal{C}$ is proper, this does not traverse any terminal x' that belongs to some $C' \neq C$. Imagine removing the (at least two) edges labeled i, setting the i-th coordinate of all vertices on the $i = 1$ side of $T[C]$ to 1, and connecting x and to $\bar{x}$ via a single edge labeled i, and $\bar{x}$ to the original parent of x in T. This gives a tree of cost strictly smaller than $T[C]$, thus, a tree of cost strictly smaller than T. This contradicts the optimality of T.

It follows both that i crosses the cut once, and that i is adjacent to a leaf in $T[C]$. Hence, by replacing x with its parent $\bar{x}$ and removing i, the algorithm maintains a proper partition of the terminals. ◀

Proof of Claim 6. Since $\mathcal{C}$ is proper, the forest induced by $T[\mathcal{C}]$ contains disjoint components. We call any vertex in this forest of degree ≥ 3 an *internal split*. Suppose we remove all internal split vertices, s.t. by removing a vertex v, we break the component into $deg(v)$ disjoint components. This results in a collection of disjoint paths. Over this set of paths we created, we now remove all edges labeled by a bad coordinate, leaving us only with paths over good coordinates. We call the result of these operations as the *path decomposition* of the forest.

Let us upper bound the number of paths in the path decomposition. The split step of the algorithm is executed when the pluck-a-leaf step fails. It follows from Claim 5 that all leaves in the forest over the component in $\mathcal{C}$ are adjacent to bad coordinates. As a result, we have a forest of $l \leq 2q$ leaves. Observe that the number of internal split vertices is at most l, and that the number of paths we create by removing the internal split vertices is at most $2l \leq 4q$. Then, by removing at most $2q$ edges labeled with a bad coordinate, where each removal splits one path into two, we end up with at most $8q$ disjoint paths overall.

Imagine that by picking a coordinate uniformly at random, the algorithm picked i which is associated with an edge that resides on a path of length ≥ 2. Let j be a neighboring coordinate of i on this path. Since j is a good coordinate, j is fixed on one side of the i-cut. Furthermore, as j is a neighbor of i, it follows that exactly one node on the other side of the i-cut (where j is not fixed) has its j-th coordinate different than all other coordinates. Since we assume throughout this section that i induces a distinct cut from j, this node must be a terminal. Pattern matching then finds the unique terminal that can be adjacent to i. Observe also that the recursion now continues by treating this unique terminal as a leaf, so the next step the algorithm performs (after the split) is the pluck-a-leaf step.

Therefore, by picking an edge on a good path of length ≥ 2, the split-step of the algorithm does not abort and breaks C into two proper sub-components. We know the forest contains $> 60q^2$ edges, out of which $\leq 2q$ are bad, and $\leq 4q - 1$ are good, but may reside on paths of length 1. Therefore, w.p. $> \frac{60q^2-6q}{60q^2} = 1 - \frac{1}{10q}$ the split-step does not abort and returns a proper partition of the terminals.

To complete the proof, observe that whenever the split step is executed and a good coordinate i is picked, then the algorithm removes the good path (in the path decomposition) on which the edge labeled i resides. This follows from the fact that immediately after removing i, at least one of its endpoints now serves as a leaf in the subcomponent created. So the algorithm plucks that leaf, and inductively, all of its parents along the path (which is composed solely out of good coordinates). As a result, between any two consecutive

executions of the split step, the number of paths in the path-decomposition we have, strictly decreases. Therefore, the algorithm cannot execute the split step more than $4q$ times. ◀

4 The General Case

Before describing the general case, let us briefly discuss why the conventional way of initially contracting all coordinates that define the same cut and applying the algorithm from the previous section might result in a tree of cost $d+\omega(q^2)$. The analysis of the first two steps of Algorithm 1 still holds. The problem lies in step 3, where the algorithm runs the MST-based 2-approximation, when it is left with $< 60q^2$ contracted coordinates. These contracted coordinates correspond to $\tilde{d} \gg q^2$ true coordinates. So by using any constant approximation on this entire forest, we may end with a tree of cost $d+\Omega(\tilde{d})$.

Our revised algorithm does not contract edges. Instead, let us define a *simple* coordinate as one for which the split-step of the original algorithm is successful. Formally,

▶ **Definition 7.** Given a set of terminals C and a non-constant coordinate i on C, we say that i is a *simple* coordinate if the pattern matching procedure finds a unique terminal in C_0 or a unique terminal in C_1.

We detail the new algorithm in Figure 2 below, and prove its correctness in the following theorem.

▶ **Theorem 8.** *The algorithm in Figure 2 returns a tree of cost* $d+O(q^2)$.

The proof of Theorem 8 follows the same outline as the proof of Theorem 4. Observe that Claims 5 and 6 still hold[1]. Therefore, with probability $\geq 1/2$, the algorithm enters step 3 with a proper partition. Thus, by the following claim, the algorithm outputs a tree of cost $d+O(q^2)$.

▶ Claim 9. Assume that when step 3 is executed, $\mathcal{C}$ 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. Let $\mathcal{C}^{\text{post}}$ denote the instance which results after splitting on all coordinates with weight greater than q. The proof of the claim reduces to showing that there exists a forest over $\mathcal{C}^{\text{post}}$ 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 $\mathcal{C}$ we get has cost $\leq d'+O(q^2)$.

The set of all terminals in all components of $\mathcal{C}^{\text{post}}$ is composed of terminals that belong to the original $\mathcal{C}$, and to new terminals, added by coordinates of weight more than q which are *not* simple. We construct a forest of low cost over $\mathcal{C}^{\text{post}}$ by (i) taking an optimal Steiner forest $\mathcal{T}$ over $\mathcal{C}$, (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 2 that the good and bad coordinates cannot define the same cut). Therefore,

[1] The split step 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 set $\mathcal{C}$, s.t. each element $C \in \mathcal{C}$ is a subset of the terminals. Initially, $\mathcal{C}$ is the singleton set, containing all terminals.

1. **Pluck-a-leaf step:**
   if exists a leaf, pluck it, just as in the original algorithm.
2. **Split step:**
   else-if $\sum_{C\in\mathcal{C}} |\{\text{simple coordinates of } C\}| \geq 8q^2$, then:
   - Pick i u.a.r from $\bigcup_{C\in\mathcal{C}}\{\text{simple coordinates of } C\}$, then continue as in the original algorithm.
3. **Base of recursion step:**
   else
   - Contract all coordinates that define the same terminal-cut. For a coordinate i in the contracted instance, let $w(i)$ be the number of coordinates in the original instance that i represents.
   - For every i with $w(i) > q$ and the component C in which i resides (we later show C is unique),
     - Apply pattern matching to i in C. Let D be the set of coordinates fixed on either side of the i cut, and let $\mathbf{b}^i$ be their corresponding value.
     - **if** i is simple, split on C into C_0 and C_1 just as the original algorithm.
     - **else**
       * Define the node $y(i)$ as the node where $y_i = 0$, every coordinate $j \in D$ is set to b^i_j, and every coordinate $j \notin D$ is set to 0.
       * Add $y(i)$ to its appropriate side of the cut, and $\overline{y(i)}$ ($y(i)$ with coordinate i flipped) to its appropriate side of the cut.
       * Replace C with the two sets C_0 and C_1.
   - For every $C \in \mathcal{C}$ find its MST $T(C)$, and retrieve the forest $\{T(C)\}$.
   - For every i with $w(i) > q$: add an edge labeled i between its corresponding terminals on both sides of its cut (either the unique terminals that match its pattern, or between $y(i)$ and $\overline{y(i)}$), thus merging two trees into one.
   - Expand all contracted coordinates to their original set of coordinates by replacing i with a path of length $w(i)$.

Figure 2 Algorithm for the general case

since $\mathcal{C}$ is proper, the coordinate i resides in a single component. Second, observe that step 3 is run when the pluck-a-leaf step fails. Therefore, as in the proof of Claim 6, $\mathcal{T}$ is a forest with at most $2q$ leaves, so its path decomposition contains $8q$ paths. Our next observation is the following proposition.

▶ Proposition 10. 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 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 2). Therefore, we may assume i is adjacent to x. Furthermore, as x is not an endpoint, there exists a good coordinate $j \neq i$ adjacent to x 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 Claim 6, i is simple, where x is the unique terminal on one side of its cut matching i's pattern. ◀

Proposition 10 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 coordinates of weight more than q is at most $4q$. Furthermore, the number of edges on any path of length no more than q with no terminal on them is at most $4q^2$. Finally, since step 3 is executed only when the split-step fails to execute, the number of edges on paths that do have a terminal on them is at most $8q^2$. It follows that the path decomposition of $\mathcal{T}$ contains at most $12q^2$ 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 $2q$ bad edges; at most $12q^2$ 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 $4q \cdot 2 \cdot q = 8q^2$. Thus $\mathcal{T}^{\text{post}}$ contains no more than $22q^2$ edges. Thus completes the proof. ◀

Runtime analysis: Step 1 of the algorithm can be implemented in time linear in the size of the dataset, i.e. $O(nd)$. Counting the number of simple coordinates takes time $O(nd^2)$. Splitting on one of them takes time $O(nd)$. In step 3, a naive implementation of the contraction procedure takes time $O(nd^2)$ 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 the pluck-leaf step and the split step of the algorithm construct the optimal tree and we can uniquely identify endpoints of the edges for the coordinates flipped at those locations. Even for step 3 of the algorithm, we can declare every edge of weight $> q$ to be good, and 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.

Several open problems arise out of this work. Can one achieve an additive approximation of $d + O(q)$? The algorithm presented in the work of Blelloch et al. [5] runs in polynomial time for $q = O(\log d)$. It would be interesting to see if our techniques can be used to get a

polynomial time algorithm which constructs the optimal tree for $q = O((\log d)^2)$. It would also be interesting to extend our results to larger alphabet sizes.

References

1 The international hapmap project. *Nature*, 426(6968):789–96, 2003.

2 R. Agarwala and D. Fernandez-Baca. A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 140 –147, nov 1993.

3 Noga Alon, Benny Chor, Fabio Pardi, and Anat Rapoport. Approximate maximum parsimony and ancestral maximum likelihood. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 7:183–187, January 2010.

4 Piotr Berman and Viswanathan Ramaiyer. Improved approximations for the steiner tree problem. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, SODA '92, pages 325–334, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics.

5 Guy E. Blelloch, Kedar Dhamdhere, Eran Halperin, R. Ravi, and Srinath Sridhar. Fixed parameter tractability of binary near-perfect phylogenetic tree reconstruction. In *International Colloquium on Automata, Languages and Programming*, pages 667–678. Springer, 2006.

6 Al Borchers and Ding-Zhu Du. The k-steiner ratio in graphs. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, STOC '95, pages 641–649, New York, NY, USA, 1995. ACM.

7 Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved lp-based approximation for steiner tree. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 583–592, New York, NY, USA, 2010. ACM.

8 Peter Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theor. Comput. Sci.*, 351:337–350, February 2006.

9 Zhihong Ding, Vladimir Filkov, and Dan Gusfield. A linear-time algorithm for the perfect phylogeny haplotyping (pph) problem. In *In International Conference on Research in Computational Molecular Biology (RECOMB*, pages 585–600. Springer, 2005.

10 David Fernández-Baca and Jens Lagergren. A polynomial-time algorithm for near-perfect phylogeny. *SIAM J. Comput.*, 32:1115–1127, May 2003.

11 L. R. Foulds and R. L. Graham. The Steiner problem in phylogeny is NP-complete. *Adv. Appl. Math. 3*, 1982.

12 Dan Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, New York, NY, USA, 1997.

13 David A. Hinds, Laura L. Stuve, Geoffrey B. Nilsen, Eran Halperin, Eleazar Eskin, Dennis G. Ballinger, Kelly A. Frazer, and David R. Cox. Whole-genome patterns of common dna variation in three human populations. *Science*, 307(5712):1072–1079, 2005.

14 Stefan Hougardy and Hans Jurgen Promel. A 1.598 approximation algorithm for the steiner problem in graphs. In *IN: PROCEEDINGS OF THE TENTH ANNUAL ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS, SODA*, pages 448–453, 1999.

15 Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, New York, 1972.

16 Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the steiner tree problems. *Journal of Combinatorial Optimization*, 1:47–65, 1995.

17 Navodit Misra, Guy E. Blelloch, R. Ravi, and Russell Schwartz. Generalized buneman pruning for inferring the most parsimonious multi-state phylogeny. In Bonnie Berger, editor,

RECOMB, volume 6044 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2010.

18 Hans Pramel and Angelika Steger. Rnc-approximation algorithms for the steiner problem. In RÃ¼diger Reischuk and Michel Morvan, editors, *STACS 97*, volume 1200 of *Lecture Notes in Computer Science*, pages 559–570. Springer Berlin / Heidelberg, 1997. 10.1007/BFb0023489.

19 Gabriel Robins and Alexander Zelikovsky. Improved steiner tree approximation in graphs. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 770–779, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

20 Gabriel Robins and Alexander Zelikovsky. Improved steiner tree approximation in graphs, 2000.

21 Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph steiner tree approximation. *SIAM Journal on Discrete Mathematics*, 19:122–134, 2005.

22 C. Semple and M.A. Steel. *Phylogenetics*. Oxford lecture series in mathematics and its applications. Oxford University Press, 2003.

23 Srinath Sridhar, Kedar Dhamdhere, Guy Blelloch, Eran Halperin, R. Ravi, and Russell Schwartz. Algorithms for efficient near-perfect phylogenetic tree reconstruction in theory and practice. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 4:561–571, October 2007.

24 H. Takahashi and A. Matsuyama. An approximate solution for the steiner problem in graphs. *Mathematica Japonica*, 24:573–577, 1980.

25 Alexander Zelikovsky. Better approximation bounds for the network and euclidean steiner tree problems. Technical report, Charlottesville, VA, USA, 1996.

A Appendix

Here, we give short proofs of the basic properties presented in Fact 2. Many of these results are also found in other work [5].

1. Let S be a set of coordinates that define the same cut over the terminals (up to renaming of C_0 and C_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. ◀