On the Complexity of Distance-based Evolutionary Tree Reconstruction

Valerie King[†*] Li Zhang [†] Yunhong Zhou[†]

Abstract

We give the first tight lower bounds on the complexity of reconstructing k-ary evolutionary trees from additive distance data. We also consider the problem under DNA-based distance estimation assumptions, where the accuracy of distance data depends on the length of the sequence and the distance. We give the first $o(n^2)$ algorithm to reconstruct trees in this context, and prove a trade-off between the length of the DNA sequences and the number of distance queries needed to reconstruct the tree. We introduce new computational models for understanding this problem, which simplify the development of algorithms. We prove lower bounds in these models which apply to the type of techniques currently in use.

1 Introduction

There has been extensive study on computational tools for evolutionary (phylogenetic) tree reconstruction using either relative or absolute distances. These algorithms access their input by a sequence of queries, each about the relationship of a few species. We model them as algorithms with access to an oracle. The extant methods use two types of oracles: the relative distance or "relation" oracle and absolute distance or "distance" oracle. A relation oracle is an oracle that takes three (for rooted trees) or four (for unrooted trees) species as input and determines which pair or pairs of leaves are closer to each other than the other leaf or leaves, i.e., it returns the topology of the induced tree on those species. A distance oracle is an oracle that takes two species and returns the distance between them in the tree. We call the methods using those oracles, respectively, the relation-based method (also known as the experiment-based method in [10] or the quartet method in the unrooted case) and the distance-based method [15] .

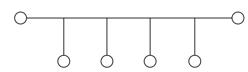

Figure 1: A caterpillar tree with six leaves

We measure complexity of such algorithms by the number of queries made to the oracle. Distance-based oracles are at least as powerful as relation-based oracles as each relation query can be simulated by a constant number of distance queries. In fact, they can be more powerful, as in the example of a caterpillar tree with n leaves (see Figure 1). For such a tree, the relation-based method requires $\Omega(n \log n)$ queries as it amounts to sorting n elements by comparisons. On the other hand, the distance-based method needs only $O(n)$ queries. While there have been tight bounds on the complexity of relation-based methods, for both binary or k-ary trees ([10, 1]), the bounds for distance-based methods are not tight.

In this paper, we first show a tight $\Omega(kn \log_k n)$ lower bound on the number of queries of distance-based methods for k-ary trees with n leaves. This matches the known upper bound of relation-based methods. Therefore, while distance-based methods provide $\log n$ factor speed-up for some trees (e.g., caterpillar trees), it does not reduce the number of queries needed in the worst case. To prove our lower bound, we first describe a partition problem and then show that an adversarial strategy for the tree reconstruction problem can be created from an adversarial strategy for that problem.

In recent years, much attention has been paid to the problem of reconstructing trees when the distance information only approximates those in the true tree. Such problems arise naturally in DNA-based distance estimation methods. The methods, pioneered by [6, 4] and others, model the evolution as a stochastic process and use the DNA sequences to derive a measurement on the mutation distance between species on the tree. One can then apply distance-based methods to reconstruct the tree based on those measured distances. The accuracy of distance obtained this way is a function of the length of the DNA sequences and the distance between the species. Earlier work studied the length of

*University of Victoria, Victoria, BC, Canada. Email: val@cs.uvic.ca

†Systems Research Center, Hewlett-Packard Labs, 1501 Page Mill Road, Palo Alto, CA 94304. Email:{valerie.king, l.zhang, yunhong.zhou}@hp.com

the DNA sequences required to reconstruct the tree with high probability. The major breakthrough made in [4] relates the bound to the so-called "edge-depth" rather than the diameter of the tree and thus reduced the DNA length needed from exponential to polynomial in terms of the number of species. A series of work [4, 5, 9, 3, 2] reduced the running time to $O(n^2)$ [3] from the early high degree polynomial bound.

In DNA-based distance estimation, if the distance between the sequences grows too large, the distance measure becomes unreliable. All the known algorithms proceed by finding pairs of leaves which are not far away in the tree. This motivates us to consider the bounded and restricted distance oracle models. In a τ-bounded distance model, the oracle returns accurate distances for only pairs within distance τ. In a (τ, ϵ)-restricted model, the oracle returns a distance with error up to ϵ for pairs within distance τ. These models offer us a much simpler view of computation in the context of DNA-based distance estimation.

We then consider the complexity of evolutionary tree reconstruction under those distance oracle models. By extending our lower bound argument for the exact distance model, we obtain lower bounds for the bounded distance model. We show that $\Omega(n\kappa(2\tau, 1/8))$ queries are required where $\kappa(2\tau, 1/8)$ is the the minimum number of balls of radius 2τ needed to cover $n/8$ leaves of the tree. This implies a worst case lower bound of $\Omega(n^2/\tau)$.

We then present algorithms for the bounded distance model. We first describe an algorithm that runs in $O(n^2)$ time for $\tau = 2g$ where g is the edge-depth of the tree. The technique in our algorithm is similar to that in [3] but is significantly simpler. Then, we describe an algorithm that runs in sub-quadratic time if we allow slightly larger τ. The running time of the algorithm is $O(\frac{1}{\epsilon_0}(n\kappa(\frac{\tau}{4} - (1 + \epsilon_0)g) + n \log n) \log g) = O((n^2/\tau + n \log n) \log g)$, where $\tau > 4(1 + \epsilon_0)g$ for any $\epsilon_0 > 0$. Our algorithm runs in two stages by first clustering the leaves and then constructing the tree incrementally by inserting those clusters according to a particular order. We then show that our algorithms for bounded oracles can be easily extended to (τ, ϵ)-restricted model for $\epsilon < 1/4$ with only a constant factor blow-up.

We also show the connection between the restricted oracle model and the DNA-based distance estimation methods. In particular, we show that an algorithm using the (τ, ϵ)-restricted oracle can be converted into an algorithm using DNA-based distance estimation method with DNA length roughly $O(\alpha^{2\tau} \log n/\epsilon^2)$ where α is a constant determined by the minimum mutation rate on each edge. On the other hand, any DNA-based distance estimation method can be converted into an algorithm using the τ-bounded oracle for $\tau = O(\log(mn)/\log \alpha)$ where m is the length of DNA sequences. Therefore, our upper bound for the restricted model and lower bound for the bounded oracle model can both be translated to the upper and lower bounds, respectively, for the DNA-based distance estimation method. This gives a tradeoff (stated in Theorem 6.3) between the length of DNA sequences and the number of queries needed for reconstruction. Our sub-quadratic algorithm shows that all pairwise distances between DNA strings do not need to be computed in order to construct the tree, which can result in substantial time savings.

The paper is organized as follows. We give some definitions in Section 2. In Section 3.1, we describe an abstract game and establish a tight bound for the game. It serves as the basis of our lower bounds arguments for both exact and bounded distance oracles, as shown in Section 3 and 4, respectively. In Section 5, we present algorithms for restricted oracle models. Section 6 establishes the connection between the restricted model and the DNA-based distance methods. Finally, we conclude with the proposal of some open problems and future work directions. Due to space constraint, we sketch some proofs in this extended abstract. For technical details, see our technical report [13].

1.1 Related work

Upper bounds: There has been extensive work on reconstruction of trees using relation and distance oracles with no error, beginning in 1977 with [15]. Optimal algorithms can be found in [10], [8], and [1].

Numerous techiques and models have been proposed to do tree reconstruction when the data is not accurate. In [7], Farach *et al.* consider the existence of tree metric that is sandwiched by two distance metrics. In [4, 5], Erdos *et al.* showed that if a distance metric is not "far" from a tree metric, it then uniquely defines a tree. All the tree reconstruction algorithms presented in [4, 5, 9, 3, 2] can be viewed as under the restricted oracle model although it is never made explicit in those papers.

Lower bounds: In the relation model, Kannan *et al.* gave an $\Omega(n \log n)$ lower bound for constructing binary trees based on information theory. In 2001, Brodal *et al.* [1] gave a $\Omega(nk \log_k n)$ lower bound for degree k trees.

In the distance model, in 1989, Hein [8] gave a $\Omega(n^2)$ lower bound for reconstructing an arbitrary tree from additive distance data. In 1999, Kao *et al.* [11] claimed a $\Omega(n \log n)$ lower bound for constructing binary trees from distance data with a flawed proof. For ultrametric distance data, they also give an $\Omega(deg(u)^2 + \sum_{v \in T \setminus u}(deg(v) - 1)^2)$ lower bound for u a node of maximum degree and a lower bound of $\Omega(n \log n + nk)$ in the case where all the leaves are children of $n/(k-1)$ vertices of degree k.

The problem of proving tighter lower bounds in the case of error does not seem to have been addressed in the literature.

2 Preliminaries

Trees Suppose that T is a unrooted tree without degree two nodes. T is semi-labeled if all the leaves of T are labeled but the internal nodes are not. Evolutionary trees are semi-labeled trees where the leaves represent species and internal nodes the evolution branching points. T is weighted if every edge e in T has a positive weight $w(e)$. In what follows, we assume that $w(e) \geq 1$ for all $e \in T$. An unweighted tree is treated as a weighted tree with unit edge length. The distance $\mathrm{d}(i,j)$ between two leaves i, j is the total weights of the edges on the path between them in T.

For any subset L of leaves, define the induced tree of L to be the weighted tree joining L. An induced tree can be obtained by deleting all the leaves not in L from T and removing all the degree two nodes by merging two edges e_1, e_2 by a single edge with weight $w(e_1) + w(e_2)$. For three leaves u, v, w, their induced tree in T is a tree with one interior node $o(uvw)$. Define $\sigma(w, uv) = \mathrm{d}(o(uvw), w) = \frac{\mathrm{d}(u,w)+\mathrm{d}(v,w)-\mathrm{d}(u,v)}{2}$. $\sigma(w, uv)$ tells us how to attach w to the path between u, v.

If we remove an edge $e = (u, v)$ from T, we obtain two subtrees, one is rooted at u and the other at v. We denote those two rooted subtrees $T_u(e)$ and $T_v(e)$, respectively. For a rooted tree, define its min-depth, denoted by $\mathrm{d}_{\min}$, to be the distance from the root to the nearest leaf in the tree. Let the edge-depth $g(e)$ be $w(e)+\max(\mathrm{d}_{\min}(T_u(e)), \mathrm{d}_{\min}(T_v(e)))$.[1] Define the edge-depth $g(T)$ of a tree T to be $\max_{e \in T} g(e)$. According to [4], for unweighted trees, $g(T) = O(\log n)$ in the worst case and $O(\log \log n)$ in average for random trees. For any node u in T, the removal of u decomposes T into one or three subtrees. By the definition of edge-depth, each of those subtrees contains a leaf that is at most $g(T)$ away from u.

Distance oracles A distance oracle $\mathcal{O}$ is an oracle that takes a pair (i, j) and returns a non-negative number. Denote by $A^{\mathcal{O}}$ an algorithm that queries distance oracle $\mathcal{O}$. These are several types of oracles that we are interested in.

Definition 2.1. *A distance oracle $\mathcal{O}$ is*

1. τ***-bounded*** *if $\mathcal{O}(i,j) = \mathrm{d}(i,j)$ when $\mathrm{d}(i,j) \leq \tau$ and $\mathcal{O}(i,j) = \phi$ otherwise;*
2. ϵ***-approximate*** *if $|\mathcal{O}(i,j) - \mathrm{d}(i,j)| \leq \epsilon$. When $\epsilon = 0$, the oracle is called exact;*
3. (τ,ϵ)***-restricted*** *if $|\mathcal{O}(i,j) - \mathrm{d}(i,j)| \leq \epsilon$ when $\mathrm{d}(i,j) \leq \tau$ and $\mathcal{O}(i,j) = \phi$ otherwise.*

Evolution models The existing tree reconstruction algorithms based on DNA sequences all model the evolution as a stochastic process. In such models, each DNA sequence is represented by a string over the character set Σ. Each edge e in the tree is associated with a mutation matrix $M(e)$ that describes the mutation probability from each character to any other character. The evolution starts at the root of the tree with an initial string and propagates towards the leaves. When propagating down along the edge e, each character is mutated independently according to the mutation matrix $M(e)$. There are many ways to define $M(e)$, from the simplest two character symmetric model (Cavender-Farris(CF) model, or two-state Neyman model [4, 12]) to the most general model ([14, 2]). In our paper, we only consider the CF model as there is no inherent difficulty to go from the simplest to the most general model if we define edge weights as in [2]. In the CF model, $\Sigma = \{0, 1\}$, and the mutation probability is symmetric, i.e. it is the same from 0 to 1 and from 1 to 0.

Tree covers Our lower and upper bounds results for bounded and restricted oracles all involve the cover of the trees. Here, we give some definitions and summarize some properties. A leaf u τ-covers a leaf v in T if the distance between u and v is at most τ. When τ is clear from the context, we also simply say that u covers v. We say that a set of nodes U *covers* a set of nodes V if every node in V is within distance τ of some node in U. A (τ, δ)-cover of a tree is a set of leaves that τ-covers at least δn leaves. When $\delta = 1$, we also call it a τ-cover or a cover when τ is clear. The leaves in a cover are called *centers*. A cover induces a partitioning of the leaves according to by which center a leaf is covered. We call each set in the partitioning a *cluster*. A cover is *minimal* if no center is covered by any other center.

The cover number $\kappa(\tau, \delta)$ is the minimum size of (τ, δ)-covers. $\kappa(\tau, 1)$ is also written as $\kappa(\tau)$. Denote by $\tilde{\kappa}(\tau)$ the *maximum* size of any minimal cover. A minimal cover can be computed using a simple greedy algorithm with running time $O(n\tilde{\kappa}(\tau))$. It is easy to verify the following:

Lemma 2.1. *$\kappa(\tau) \leq \tilde{\kappa}(\tau) \leq \kappa(\tau/2)$, and $\kappa(\tau, \delta) \leq \frac{4a\delta n}{\tau}$, where a is the maximum edge weight.*

[1] Our definition of edge-depth is an extension of the defintion in [4] to weighted trees.

3 Lower Bounds for Exact Oracles

In this section, we study the lower bounds of evolutionary tree reconstruction using exact distance queries. We first consider an abstract game called the Partition Problem.

3.1 The Partition Problem

We define the (n, k)-Partition Problem as follows:

(n, k)-Partition Problem: Given n elements which are partitioned into k equal sized classes, determine the elements in each class by asking a sequence of queries of the form: "Are elements a and b in the same class?" Cost is measured by the number of queries required.

We first give a lower bound for this problem.

Lemma 3.1. *Given an (n, k)-Partition Problem for $1 < k < n$, an adversary can force an algorithm to ask $\Omega(nk)$ queries whose answers are "No".*

Proof: First, we observe that for any $k > 1$, there is an obvious lower bound of $\Omega(n)$. Now we assume $k \geq 3$.

We prove a lower bound by an adversary argument. The adversary maintains a mapping of elements to classes, which is adjusted as the queries proceed. Initially, the adversary places the elements arbitrarily in k equal-sized classes. Let NQ be the set of pairs of elements asked about so far for which the adversary has answered "No", YQ be the set of pairs for which the adversary has answered "Yes." Let C be the adversary's current mapping from elements to classes. A mapping C is *consistent* with NQ and YQ if, for all pairs $(a, b) \in NQ$, $C(a) \neq C(b)$ and for all pairs $(a, b) \in YQ$, $C(a) = C(b)$.

Let $q(a)$ be the number of queries asked about a so far. A class is *fixed* by the adversary when its elements are involved in $n/3$ queries in NQ. An element a is *fixed* by the adversary when $q(a) = \lfloor k/3 \rfloor - 1$ or its class becomes fixed. The goal of the adversary is to say "No" as long as possible. If the algorithm asks a query (v, w), and $C(v) = C(w)$, the adversary swaps either v or w with an unfixed element from a different class, so as to be able to answer "No". The adversary answers "Yes" to a query pair only when both elements are fixed and in the same class. Once an element is fixed, its class is never changed. We show that such a strategy is possible as long as $|NQ| < (n/12)\lfloor k/3 \rfloor$.

Given a current mapping C, NQ, and YQ, two elements a and b are *swappable* if $C(a) \neq C(b)$ and there is a consistent mapping C', such that $C'(a) = C(b)$, $C'(b) = C(a)$, and for all $y \neq a, b$, $C'(y) = C(y)$.

It is not hard to see:

Claim 1: An element a is swappable with another element b if:
(1) there is no b' such that $C(b') = C(b)$ and $(a, b') \in NQ$ and
(2) there is no a' such that $C(a') = C(a)$ and $(a', b) \in NQ$ and
(3) neither a nor b is fixed and
(4) $C(a) \neq C(b)$.

If $|NQ| < (n/12)\lfloor k/3 \rfloor$ then it is easy to see that (A) there are fewer than $n/6$ elements with $d \geq \lfloor k/3 \rfloor - 1$ and (B) there are fewer than $\lfloor k/6 \rfloor$ classes that have been fixed. Hence,

Claim 2: If $|NQ| < (n/12)\lfloor k/3 \rfloor$, there are fewer than $n/3$ fixed elements.

We are now ready to show:

Claim 3: If any element a is not fixed and $|NQ| < (n/12)\lfloor k/3 \rfloor$, there is an element b that is swappable with a.

Proof of Claim 3: Since $q(a) < \lfloor k/3 \rfloor - 1$, no more than $\lfloor k/3 \rfloor \cdot n/k - n/k - 1 \leq n/3 - n/k - 1$ b's are eliminated because condition (1) in the claim above is not satisfied. Since a's class is not fixed, no more than $\lfloor k/3 \rfloor \cdot n/k \leq n/3 - 1$ b's are eliminated because condition (2) isn't satisfied. Since there are fewer than $n/3$ fixed elements, then fewer than $n/3$ elements are eliminated because of condition (3). Condition (4) eliminates n/k elements. In conclusion, there is at least one element b which satisfies the conditions (1), (2), (3) and (4).

Claim 3 implies that while any element a is not fixed and $|NQ| < (n/12)\lfloor k/3 \rfloor$, every query involving a can be answered "No". Furthermore, if every class has size at least 2, there are more than one partition consistent with NQ and YQ. Thus we have shown that a solution to the partition problem requires $|NQ| = \Omega(nk)$. $\square$

We can solve the partition problem by inserting elements one by one. We pick a representative for each of classes. Whenever we insert an element, we query it against those representatives. If the answer is "Yes" of one query, we find a class that contains the element. Otherwise, we create a new class that contains the element and assign the element to be the representative of that class. Each insertion takes at most $k-1$ queries. This implies an upper bound of $O(nk)$ for the Partition Problem. We conclude:

Theorem 3.1. *The Partition Problem on n elements and k classes has complexity $\Theta(nk)$.*

3.2 Lower Bounds for Exact Oracles

Suppose that the tree T has maximal degree k. Our main result in this section is the following theorem:

Theorem 3.2. *The number of queries needed to reconstruct an evolutionary tree is $\Omega(nk \log_k n)$.*

Proof: Let $n = k^m$ for some integer m. Suppose the tree to be reconstructed is a complete k-ary tree.

The idea of the proof is to show that the reconstruction algorithm can be forced to solve many partition problems. In particular, for each internal node of height h, the adversary can force the algorithm to solve the partition problem on k^h elements, each with k classes, with the worst case number of queries required for each problem. We denote the partition problem for node x by $P(x)$. Let NQ(x) denote the set of "No" queries NQ for $P(x)$.

We show that the cost of the algorithm is equal to the number of "No"'s given for the partition problem at each node of height $h > 1$ or $\Omega(\sum_{j=1}^{k} jk^{i-1}) = \Omega(k^{i+1})$. Summing over all internal nodes of height i and over each height $i = 1, 2, \cdots, m$, this yields the lower bound claimed.

We label the nodes of a complete k-ary tree top-down as follows. The root is labeled 1. The i^{th} child of a node a is labeled by a's label concatenated with i. A node of depth j is therefore labeled by a string in $\{1, 2, \cdots, k\}^j$.

We construct the Adversary from the adversaries for $P(x)$ as follows. The Adversary maintains a 1-1 mapping from species to leaves: Let C_x denote the mapping for the adversary in $P(x)$. The species a is mapped to the leaf labeled $C_{x_1}(a), C_{x_2}(a), \cdots, C_{x_m}(a)$ where x_1 is the root. Let $p(x)$ denote the label of the parent of node x. The label of x_i is given by a's class in the adversary mapping for the problem at its parent node, i.e., $C_{p(x_i)}(a)$, concatenated with the label of its parent $p(x_i)$.

We describe the Adversary strategy: The Adversary reveals that the tree is a rooted full k-ary tree. We model the state of revealed knowledge by placing a set of n pebbles on internal nodes of the tree, one for each species. Initially, all pebbles are placed on the root.

If query (a, b) is asked, then let x be the lowest common ancestor of the nodes on which the pebbles for a and b lie. Let i be the height of x.

Case 1: Pebbles for a and/or b lie on node x If the adversary strategy for $P(x)$ requires the adversary to say "No," the Adversary answers $\mathrm{d}(a, b) = 2i$, and (a, b) is added to $NQ(x)$. If the adversary fixes a species to class j and its pebble lies in x then the Adversary moves its pebble to the j^{th} child of x.

If the adversary strategy requires a "Yes" answer, then this implies that both pebbles are now located in the subtree rooted by the same child of x and the query is now answered according to the current position of the pebbles.

Case 2: If neither pebble for a and b lie in x then the two pebbles have already been fixed in $P(x)$. The adversary answers that the distance between them is $2i$. No pebble is moved.

Note that when a's pebble is in node x, a is essentially a "dummy" in all $P(y)$, y a descendant of a, since no queries regarding a affect these problems. So if a and b are swapped in $P(y)$ then for all z on the path from y to the leaves assigned to a or b, they are swapped in $P(z)$, with no effect on the partition problem.

The complexity of the adversary strategy is proved by induction on the height of the trees. We omit the details due to lack of space. The key idea is: Assigning the species to one of the k subtrees of the root is equivalent to solving the partition problem on n elements with k subtrees. Answering that the distance between two species is "$2m$" indicates only that two species are in two different subtrees at height m but reveals no information about the species' positions in the subtrees of height $m - 2$ or less. The queries answered "Yes" either confirm information already revealed by the adversary and/or become the equivalent of queries on a partition problem for a descendant node. □

4 Lower Bounds for Bounded Oracles

In this section, we study the lower bounds of evolutionary tree reconstruction using bounded oracles as defined in Section 2. We only consider bounded oracles as a lower bound for τ-bounded oracle is also a lower bound for a (τ, ϵ)-restricted oracle. Recall that $\kappa(\tau, \delta)$ denotes the minimum number of leaves needed to τ-cover δn leaves in the tree.

Theorem 4.1. *For $\tau > 0$, an algorithm with access to a τ-bounded oracle needs $\Omega(n\kappa(2\tau, 1/8))$ queries to reconstruct a tree with bounded degree in the worst case.*

Proof: The proof of this theorem is similar to the proof of the lower bound for the partition problem. The adversary maintains a 1-1 mapping π from the set of species to the set of leaves of the tree. Let NQ be the set of queries (a, b) for which the adversary answers ϕ and YQ be the set of triples (a, b, d) where the answer to the query (a, b) is d. A mapping π is consistent with NQ and YQ if $(a, b) \in NQ$ implies the distance $\mathrm{d}(\pi(a), \pi(b))$ between $\pi(a), \pi(b))$ is greater than τ and if $(a, b, d) \in YQ$, then $\mathrm{d}(\pi(a), \pi(b)) = d$. The adversary may *fix* a species a to its leaf, i.e., $\pi(a)$ will remain unchanged throughout the adversary strategy.

A species u *neighbor-τ-covers* a species z under a mapping π if there exists a species v such that $(u, v) \in NQ$ and $\pi(v)$ τ-covers $\pi(z)$. Two species a and b are *swappable* under π if there is a consistent mapping π' such that $\pi = \pi'$ except that $\pi'(a) = \pi(b)$ and $\pi'(b) = \pi(a)$.

The adversary maintains the invariant that species which are not fixed are not involved in any query of

YQ. The invariant implies that s and t are *swappable* iff s and t are not fixed and s does not neighbor-τ-cover t and t does not neighbor-τ-cover s.

The adversary fixes a species a to its leaf if
(1) a neighbor-τ-covers at least $n/8$ species; or
(2) a is neighbor-τ-covered by $n/2$ species or $\pi(a)$ is within a ball of radius τ of a $\pi(b)$ where b is neighbor-τ-covered by $n/2$ species.

We claim that the number of fixed species is less than $39n/112$ if $|NQ| < \kappa(2\tau, 1/8)n/56$.

Proof of Claim: Let A be the set of species fixed under (1) and B be the set of species fixed under (2). In the following we bound $|A|$ and $|B|$ separately.

If $|A| \geq n/28$, then because each element $a \in A$ neighbor-τ-covers $n/8$ species, a must be in $\kappa(\tau, 1/8)$ queries in NQ. Since $|A| \geq n/28$, $|NQ| \geq \kappa(\tau, 1/8)n/56 \geq \kappa(2\tau, 1/8)n/56$.

If $|B| \geq 5n/16$, then let b be any element of B. There are $(5n/16)(n/2)$ pairs of species in which one of the pair neighbor-2τ-covers the other. Using the fact that a species can't cover more than n other species, we see that there must be at least $n/28$ species that neighbor-2τ-cover at least $n/8$ species. Each species must have at least $\kappa(2\tau, 1/8)$ queries in NQ, so $|NQ| \geq \kappa(2\tau, 1/8)n/56$.

Because $|A| < n/28$ and $|B| < 5n/16$, thus the total number of fixed species is less than $39n/112$, and the claim is proved.

Now, let a be a species which is not fixed. If we add up the number of fixed species ($39n/112$), the number of species which neighbor-τ-covers a ($n/2$), and the number of species that a neighbor-τ-covers ($n/8$), we see that at least $3n/112 - 1$ species are left for a to be swapped with, as long as $|NQ| < \kappa(2\tau, 1/8)n/56$. So that if $|NQ|$ does not exceed that bound, the adversary can maintain its invariant. If the degree of the tree is less than $3n/112 - 1$ then there is more than one distinct evolutionary tree consistent with the queries. This implies that any algorithm needs at least $\Omega(n\kappa(2\tau, 1/8))$ queries in the worst case. □

For the caterpillar tree with n leaves, $\kappa(2\tau, 1/8) = \Theta(n/\tau)$. One direct consequence of Theorem 4.1 is:

Corollary 4.1. *$\Omega(n^2/\tau)$ queries are necessary in the worst case.*

5 Algorithms for Restricted Oracles

In this section, we describe tree reconstruction algorithms using restricted oracles. Recall that g denotes the edge depth of T. We first describe algorithms that use τ-bounded oracles. We present an $O(n^2)$ algorithm for $\tau = 2g$. The algorithm is similar to the one in [3] but is much simpler and easier to analyze. Then, we present an algorithm with running time dependent on the cover size for $\tau > 4g$. Finally, we show that a slight modification of these algorithms works under (τ, ϵ)-restricted oracles for $\epsilon < 1/4$. Throughout this section, all the trees are assumed to be binary trees.

5.1 A quadratic algorithm for bounded oracles

We first describe a tree reconstruction algorithm using a τ-bounded oracle for $\tau = 2g$. Our algorithm works incrementally by adding the leaves into the tree in a carefully chosen order. Let T_i denote the induced tree of the first i leaves. We show that in $O(n)$ time we can find another leaf to insert into T_i and thus overall the algorithm runs in $O(n^2)$ time. Denote by L_i the set of leaves in T_i. We maintain a partitioning of all the leaves in $L \setminus L_i$: for each edge e, $S(e)$ is a subset of the leaves attached to the edge e, and the set of U contains all the leaves not in any $S(e)$. For each leaf $s \in S(e)$, we store the position $o(s)$ from which s branches out together with $\mathrm{d}(s, o(s))$. For each o on e, we keep track of the leaf with the minimum $\mathrm{d}(s, o)$. For each leaf $s \in U$, an *anchor* $a(s)$ is a leaf in L_i that is within distance τ from s. A node in U may not have an anchor.

To start, we find two nodes u, v with the minimum distance. This can be done in linear time as the underlying graph is a tree. Set $L_2 = \{u, v\}$ and T_2 the tree with a single edge $e = (u, v)$. Initially, $S(e) = \emptyset$ and $U = L \setminus L_2$. For each leaf s in U, we query its distances to u and v. If both are bounded by τ, then we move it into $S(e)$ and compute $o(s)$ and $\mathrm{d}(s, o(s))$. If only one of u, v, say u, is within distance τ from s, we then set $a(s) = u$. Otherwise, $a(s)$ is undefined.

We now describe how to find a leaf to insert to T_i and how to update $S(\cdot), a(\cdot)$. Consider those edges e's with non-empty $S(e)$. If the two endpoints p, q of e are both internal nodes or both leaf nodes, then we choose $s \in S(e)$ with the minimum $\mathrm{d}(s, o(s))$. If none of such edges exists, we consider an edge $e = (p, q)$ where p is an internal node, q a leaf node, and $S(e)$ is not empty. Among all the leaves $s \in S(e)$ with the minimum $\mathrm{d}(p, o(s))$, we choose the one with the minimum $\mathrm{d}(s, o(s))$.

Now suppose that we have chosen a leaf s from $S(e)$ where $e = (p, q)$. We insert s into T_i to form T_{i+1} by splitting e at o into two edges $e_1 = (p, o)$ and $e_2 = (o, q)$ and inserting $e_3 = (o, s)$. We assign weights to these three edges accordingly. We then update the data structures by initiating three empty sets $S(e_1)$, $S(e_2)$, and $S(e_3)$. We split $S(e)$ as follows: for a leaf w in $S(e)$, if $o(w) \neq o$, we add it into either $S(e_1)$ or $S(e_2)$; otherwise, we query the distance $\mathrm{d}(w, s)$. If the oracle returns ϕ, we add w into U with the anchor node $a(w) = o$. Otherwise, we add w to $S(e_3)$. For each leaf $w \in U$, we query $\mathrm{d}(s,w)$ too. If the oracle returns ϕ, we do nothing. Otherwise, if $a(w)$ is undefined, we update $a(w) = s$. If $a(w)$ has already been defined, we can determine the position of w on the path from $a(w)$ to s. If it happens to be in the middle of an edge e', we remove w from U and add it into $S(e')$. Otherwise if it branches out from one internal node t', we update $a(w)$ to t'.

Clearly, each step in the above algorithm takes $O(n)$ time. In what follows, we shall show that we can always find a leaf s and insert it in the tree. An edge in T_i is called *unfinished* if it is not an edge in T. An edge $e = (p,q)$ is called an internal edge if p,q are both internal nodes. Otherwise, it is an external edge. For any internal node of T_i, its removal results in three subtrees which are denoted by $T_i^1(v), T_i^2(v)$ and $T_i^3(v)$. We first show the following lemma.

Lemma 5.1. *The weight of each edge of T_i is at most g. For any internal node v in T_i and for any $1 \leq j \leq 3$, there exists a leaf u in $T_i^j(v)$ so that $\mathrm{d}(u,v) \leq g$.*

Proof: The proof is by induction. At the begining, $\mathrm{d}(u,v) \leq g$ because u,v are two leaves with minimum distance. Consider an intermediate stage. We distinguish two cases.

(i) There exists an unfinished internal edge. Assume $e = (p,q)$ where both p,q are internal nodes. By the definition of edge-depth, there exists a leaf s branching out at o with $\mathrm{d}(s,o) \leq g$. By induction, there exists a leaf node q' in the subtree $T_q((o,q))$ such that $\mathrm{d}(p,q') \leq g$. Thus $\mathrm{d}(s,q') = \mathrm{d}(s,o) + \mathrm{d}(o,q') \leq 2g = \tau$. Similarly, there exists a leaf node p' in the subtree $T_p((o,p))$ such that $\mathrm{d}(s,p') \leq \tau$. Thus $s \in S(e)$.

(ii) There does not exist any unfinished internal edge, i.e. all the unfinished edges are external edges. Assume that $e = (p,q)$ where p is an internal node, and q a leaf node. Similarly, we can prove that there exists a leaf node s that branches out on edge e and a leaf node $q' \neq q$ such that $\mathrm{d}(s,q) \leq g$ and $\mathrm{d}(s,q') \leq g$. Details can be found in the complete version [13].

In both cases, after the insertion of s, we split an edge and add one new edge (o,s). Clearly the weight of (o,s) is no more than g. It is also easy to verify that the added internal node o satisfies the property as claimed in the lemma. □

For any unfinished edge e in T_i, by the definition of edge depth, there exists a leaf node s branching out from e and is at most g away from the branching node o. By Lemma 5.1, we know that there exist two leaves p,q in T_i such that $\mathrm{d}(s,p), \mathrm{d}(s,q) \leq \tau$. Therefore, $s \in S(e)$, implying that we can always find a leaf node to insert into the tree. Thus, we have that:

Theorem 5.1. *When $\tau \geq 2g$, we can reconstruct the tree in $O(n^2)$ by using a τ-bounded oracle.*

5.2 A sub-quadratic algorithm for bounded oracles

In this section, we describe an algorithm whose running time depends on the cover size when $\tau > 4g$. The running time of our algorithm may vary between $n^2 \log g / 2^{c\tau}$ and $n^2 \log g / \tau$ where $c > 0$ is a constant. Our algorithm always runs in sub-quadratic time. When the tree has a small cover, then it may run significantly faster.

The algorithm works in two steps. In the first step, we compute a cover of the leaves. In the second step, we construct the tree incrementally by inserting the clusters formed in the first step. When we insert a node, the algorithm is similar to incremental tree construction algorithm using exact oracles. We show that if we order the clusters appropriately, we can guarantee that the distance queried in the process is within τ. Further, most queries made by the algorithm are between a leaf and a center in the cover. Therefore, we are able to bound the running time by the number of centers in the cover, i.e. the cover size.

We first prove a useful structural property about trees. Suppose that $C = \{c_1, c_2, \cdots, c_m\}$ is a d_1-cover of T. We form a graph $G = (C,E)$ where $(c_i,c_j) \in E$ if $\mathrm{d}(c_i,c_j) \leq d_2$. We claim that:

Lemma 5.2. *If $d_1 \geq 0$ and $d_2 \geq 2d_1 + 2g$, then G is connected.*

Proof: For any two centers $c_1, c_2 \in C$, consider the path $q_1, q_2, \cdots, q_k$ ($q_1 = c_1, q_k = c_2$) between them in T. For any i, the removel of the edge (q_i, q_{i+1}) results in two subtrees, one rooted at q_i and the other at q_{i+1}. By the definition of edge depth, there exists a leaf v_{i+1} in the subtree rooted at q_{i+1} such that $\mathrm{d}(q_i, v_{i+1}) \leq g$. Clearly, $\mathrm{d}(q_{i+1}, v_{i+1}) \leq g$ as well. Denote by u_i the center in C that covers the leaf v_i. We then have that $\mathrm{d}(u_i,q_i) \leq \mathrm{d}(u_i,v_i) + \mathrm{d}(v_i,q_i) \leq d_1 + g$, and $\mathrm{d}(u_{i+1},q_i) \leq \mathrm{d}(u_{i+1},v_{i+1}) + \mathrm{d}(v_{i+1},q_i) \leq d_1 + g$. Therefore, $\mathrm{d}(u_i,u_{i+1}) \leq \mathrm{d}(u_i,q_i) + \mathrm{d}(u_{i+1},q_i) \leq 2(d_1 + g) \leq d_2$. That is, there is an edge between u_i and u_{i+1} in G. c_1, c_2 thus are connected. □

Once we have obtained G, we can reorder the centers to $c_1, c_2, \cdots, c_m$ such that for any $i > 1$, there exists $j < i$ such that the edge $(c_j, c_i) \in G$. This can be done by a depth-first search in G. Given the order of the centers, we now reconstruct the tree incrementally by inserting the corresponding clusters in the order.

Suppose that we have constructed the tree T_i for the leaves in the sets $C_1, C_2, \cdots, C_i$ where C_j denotes the set of leaves covered by c_j. We shall show how to add the leaves in C_{i+1} to the tree. We first describe the process for inserting c_{i+1}.

By the order of the centers, there exists c_j, $j \leq i$, such that $\mathrm{d}(c_j, c_{i+1}) \leq d_2$. We first root T_i at c_j. For any node $u \neq c_j$, denote by $p(u)$ the parent of u in the rooted tree and $T_i(u)$ the tree obtained by adding the edge $(u, p(u))$ to the subtree in T_i rooted at u. $T_i(u)$ is called *free* if it does not contain any center. Otherwise, we define the *lead* $\ell(u)$ of $T_i(u)$ to be the center in $T_i(u)$ nearest to u. For every free subtree, we construct a data structure that allows insertion in $O(\log |T_i(u)|)$ time using exact distance oracle. In the following discussion, our strategy is that we pretend to work under the exact distance oracle model, and we show that the distances queried are always within the given bound τ if we choose d_1, d_2 appropriately.

The insertion of c_{i+1} is done recursively. Suppose that we have decided that c_{i+1} is in $T_i(u)$, and we have computed $\mathrm{d}(s, p(u))$. Initially, u is c_j, the root of the tree. If $T_i(u)$ is free, we insert c_{i+1} in $O(\log n)$ time. Otherwise, we query $\mathrm{d}(c_{i+1}, \ell(u))$ to compute $\sigma(c_{i+1}, u\ell(u))$. If there is no subtree incident at the node $o(c_{i+1}u\ell(u))$, we can attach c_{i+1} to the tree that node. Otherwise, we recurse on subtree $T(o(c_{i+1}u\ell(u)))$ until we are able to insert the leaf or reach a free subtree. Now, we bound the distance ever queried by the above procedure.

We claim that

Lemma 5.3. *1. For any u, if $T_i(u)$ is not free, then $\mathrm{d}(\ell(u), p(u)) \leq d_2$.*

2. The height of a free subtree is at most d_1.

Proof: 1. Suppose otherwise $\mathrm{d}(\ell(u), p(u)) > d_2$. Since $\ell(u)$ is the nearest to u among all the centers in $T_i(u)$, for any center v in $T_i(u)$, $\mathrm{d}(v, p(u)) > d_2$. There then does not exist a path between any center in $T_i(u)$ and the centers not in $T_i(u)$, contradicting with that $c_1, \cdots, c_i$ are connected in G.

2. Otherwise, it would contradict with that every leaf is covered by a center. □

By the above two properties, we have that:

Lemma 5.4. *If we have ever queried $\mathrm{d}(c_{i+1}, v)$, then $\mathrm{d}(c_{i+1}, v) \leq 2d_2$.*

Proof: We distinguish two cases:

(i) when $v = \ell(u)$ for some u, $\mathrm{d}(c_{i+1}, \ell(u))$ is queried only if c_{i+1} is in the subtree $T_i(u)$. Or, $p(u)$ is on the path from c_{i+1} to the root c_j. Therefore $\mathrm{d}(c_{i+1}, \ell(u)) \leq \mathrm{d}(c_{i+1}, p(u)) + \mathrm{d}(p(u), \ell(u)) \leq d_2 + \mathrm{d}(c_{i+1}, c_j) \leq 2d_2$ by Lemma 5.3.1.

(ii) when v is in a free tree, similarly by Lemma 5.3.2, we have that $\mathrm{d}(c_{i+1}, v) \leq d_1 + \mathrm{d}(c_{i+1}, c_j) \leq d_1 + d_2 \leq 2d_2$. □

After we have inserted c_{i+1}, we reroot the tree at c_{i+1} and recompute the leads. Then, we insert all the other nodes in C_{i+1} by the same process. Similar to Lemma 5.4, we have that

Lemma 5.5. *If we have ever queried $\mathrm{d}(s, v)$ for some $s \in C_{i+1}$, then $\mathrm{d}(s, v) \leq d_1 + d_2 \leq 2d_2$.*

Now, we are able to state the following:

Theorem 5.2. *For any $\epsilon_0 > 0$, suppose that $\tau > 4(1+\epsilon_0)g$. Then the above algorithm uses $O(n)$ space and runs in $O(\frac{1}{\epsilon_0}(n(\tilde{\kappa}(\frac{\tau}{4} - (1+\epsilon_0)g) + \log n) \log g)$ time.*

Proof: Suppose for now that we know the edge-depth of the tree. By Lemma 5.4 and 5.5, as long as $2d_2 \leq \tau$, the distance queried in the algorithm never exceeds τ. By Lemma 5.2, we can set $d_1 = \frac{\tau}{4} - g$ and $d_2 = 2d_1 + 2g = \tau/2$. Computing the centers takes $O(n\tilde{\kappa}(d_1))$ time and $O(n)$ space. The order of the centers can be computed in time $O(\tilde{\kappa}(d_1)^2)$ and in space $O(\tilde{\kappa}(d_1))$ by an implicit depth-first search on G. Each re-rooting and computing of leads takes time $O(n)$. The insertion of a leaf takes time $O(\tilde{\kappa}(d_1) + \log n)$ while the first term accounts for the time spent on recursion and the second for the time on insertion in a free tree. Therefore, in total, the time complexity is $O(n\tilde{\kappa}(d_1) + n\log n) = O(n\tilde{\kappa}(\frac{\tau}{4} - g) + n\log n)$, and the space complexity is $O(n)$.

Since we do not know the edge depth beforehand, we can guess g by trying $1, (1+\epsilon_0), (1+\epsilon_0)^2, \cdots$ until we reconstruct the tree successfully. There are two types of failures: one is when the graph G is not connected and the other is when we query a distance while inserting a node, the bounded distance oracle returns ϕ. We abort whenever a failure happens and try a larger g. The bound follows from that we never try a value greater than $(1+\epsilon_0)g$, and we only try $\log g / \log(1+\epsilon_0) = O(\frac{1}{\epsilon_0} \log g)$ times. □

Combining the above theorem with Lemma 2.1, we can show that the following worst case bound.

Corollary 5.1. *Suppose that a is the maximum edge weight of T. Then $O(\frac{an^2 \log g}{\tau})$ queries are sufficient for $\tau > 5g$.*

Proof: Set $\epsilon_0 = 1/5$ in the above theorem. Then the running time of the algorithm is $O((n\tilde{\kappa}(\frac{\tau}{4} - (1+\epsilon_0)g) + n\log n)\log g) = O(n(\tilde{\kappa}(\frac{1}{100}\tau) + \log n)\log g) = O(an^2 \log g/\tau)$ by $\tilde{\kappa}(\tau) = O(an/\tau)$. □

The above bound is to illustrate that our algorithm always runs in sub-quadratic time if $\tau = \omega(1)$. However, the bound is quite conservative as we use $\tilde{\kappa}(\tau) = O(an/\tau)$ in deriving the bound. In a typical case, we expect that $\tilde{\kappa}(\tau)$ is much smaller than an/τ, and the running time of our algorithm is significantly lower than quadratic.

5.3 Extension to restricted oracles

We now extend the above algorithms to restricted oracles. When the distance oracle is (τ, ϵ) restricted, the distance queried is no longer exact. When ϵ is big, we may not be able to construct a unique tree. However, as shown in [4, 5], if ϵ is small enough, certain algorithms do guarantee to reconstruct a unique tree. We can apply similar argument and show that after slight modification to our algorithms, we can reconstruct the tree in the same bound as long as $\epsilon < 1/4$.

The primitive of the previous algorithms is to determine where to insert a leaf u to a path vw by computing $\sigma(u, vw)$. The following is immediate given all the edge length is ≥ 1:

Lemma 5.6. *Suppose that $\hat{\sigma}$ is computed using an ϵ-approximate oracle where $\epsilon < 1/4$. For any four leaves u_1, u_2, v, w, we have that*

$$o(u_1 vw) = o(u_2 vw) \text{ iff } |\hat{\sigma}(u_1, vw) - \hat{\sigma}(u_2, vw)| < 1/2\,.$$

The above lemma suggests the following modification to the algorithm. Whenever we try insert a leaf u to a path vw, we compute $\hat{\sigma}(u, vw)$. If there already is an interior node o' on the path vw so that $\hat{o}(u, vw)$ is within distance $1/2$ to o', we treat $o' = o(uvw)$ and recurse on that subtree. Otherwise, we create a new interior node o and attach u to o. Denote the modified algorithm $\hat{A}$. By Lemma 5.6, we can prove the following by induction.

Theorem 5.3. *If A reconstructs a tree successfully with a τ-bounded oracle, $\hat{A}$ reconstructs the same tree successfully with a $(\tau, 1/4)$-restricted oracle.*

6 Connection to DNA-based Distance Model

In this section, we establish the connection of the restricted oracle model to the DNA-based distance estimation method. The technique is mostly adapted from the previous work on DNA-based distance methods. We only consider the CF model as described in Section 2. The extension to more general mutation models should not be hard if we define edge weights as in [2].

We show that if all the mutation probabilities are the same, then the DNA-based distance estimation resembles a bounded distance oracle, and if the mutation probabilities are different, then the DNA-based method resembles a restricted distance oracle. We will focus on the former case and the discussion can be extended to the latter case easily.

We assume that the mutation probability is p, where $0 < p < 1/2$. Set $\alpha = \frac{1}{1-2p}$. We treat T as a unweighted tree. Suppose that D_i is the DNA sequence corresponding to the leaf i, and the length of D_i's is m. Denote by h_{ij} the Hamming distance between D_i and D_j and H_{ij} the probabilistic distribution on h_{ij}. Denote by $H(m, \delta)$ the binomial distribution with probability δ. Then it is well-known that H_{ij} is identical to $H(m, \frac{1}{2} - \frac{\alpha^{-\mathrm{d}(i,j)}}{2})$ [6]. For two probabilistic distribution H_1, H_2, their distance distance $\Delta(H_1, H_2)$ is defined as $\Delta(H_1, H_2) = \frac{1}{2}\sum_z |\Pr\{H_1(x) = z\} - \Pr\{H_2(x) = z\}|\,.$

Given two DNA sequences D_i, D_j, define $\hat{\mathrm{d}}(i, j) = \lfloor -\frac{\log(1-2h_{ij}/m)}{\log \alpha} + 1/2 \rfloor$, if $h_{ij} < 1/2$, and undefined otherwise. We can simulate a distance oracle using the DNA sequences under the CF model: for a query pair (i, j), compute and return $\hat{\mathrm{d}}(i, j)$. Denote by $\mathcal{D}_m$ the simulated distance oracle by DNA sequences with length m. $\mathcal{D}_m$ behaves very similar to a bounded distance oracle for T according to the following lemma.

Lemma 6.1. *If the evolution follows the CF model with uniform mutation probability p, then we have that*

1. *If $\mathrm{d}(i, j) \leq \tau$, then $\Pr\{\hat{\mathrm{d}}(i, j) = \mathrm{d}(i, j)\} \geq 1 - 2e^{-2mp^2/\alpha^{2\tau}}$.*
2. *If $\mathrm{d}(i, j) > \tau$, then $\Delta(H_{ij}, H(m, 1/2)) \leq m\alpha^{-\tau}$.*

Proof: 1 follows from Lemma 2.2 in [3], and 2 follows from $\Delta(H(m, 1/2 - \epsilon), H(m, 1/2)) \leq m\epsilon$. □

Intuitively, the above lemma states that for given DNA length, if $\mathrm{d}(i, j)$ is small then $\hat{\mathrm{d}}$ approximates d very well; and if $\mathrm{d}(i, j)$ is too big then $\hat{\mathrm{d}}$ tells us almost no information as H_{ij} is not distinguishable from a random distribution. The following theorem is the formal statement.

Theorem 6.1. *Suppose that T is an evolutionary tree under the CF model with uniform mutation probability p, then*

1. *If $A^{\mathcal{O}}$ reconstructs the tree T with probability $1 - \delta_1$ using any τ-bounded oracle $\mathcal{O}$, then there exists B with the same complexity so that $B^{\mathcal{D}_m}$ reconstructs T with probability $1 - \delta_1 - \delta$ for any $m \geq \alpha^{2\tau} \log \frac{n}{\delta} / p^2$.*
2. *If $B^{\mathcal{D}_m}$ reconstructs T with probability $1 - \delta_1$, then there exists A with the same complexity so that $A^{\mathcal{O}}$ reconstructs T with probability $1 - \delta_1 - \delta$ where $\mathcal{O}$ is a τ-bounded oracle for any $\tau \geq 2 \log \frac{mn}{\delta} / \log \alpha$.*

Proof: 1. B simulates $A^{\mathcal{O}}$ by querying $\hat{\mathcal{D}}_m$. When $\hat{\mathcal{D}}_m$ returns a distance greater than τ, B treats it as ϕ. According to Lemma 6.1.1, if $m \geq c_1 \alpha^{2\tau} \log \frac{n}{\delta} / p^2$, the estimated distance $\hat{d}(i, j)$ equals the true distance $d(i, j)$ with probability at least $1 - 2\delta/n^2$. Since there are no more than $n^2/2$ pairs, the error probability is upper-bounded by δ.

2. A simulates $B^{\mathcal{D}_m}$ by querying $\mathcal{O}$. When $\mathcal{O}$ returns ϕ, A generates a number according to the distribution $H(m, 1/2)$ and returns a distance according to the

definition of $\hat{d}$. By Lemma 6.1.2, $\Delta(H_{ij}, H(m, 1/2)) \leq m\alpha^{-\tau} \leq \frac{\delta}{n^2}$ if $\tau \geq 2\log\frac{mn}{\delta}/\log\alpha$. Therefore, the total error probability is upper-bounded by δ as well. □

So far, we have shown that if the mutation probability is uniform, then the DNA-based distance method is similar to a bounded oracle. When the probability is not uniform, the method is then similar to a restricted oracle. Suppose that T is an evolutionary tree in CF model where all the edge mutation probabilities are in an interval $[a, b]$, for some $0 < a \leq b < 1/2$. Set $\beta = \frac{1}{1-2a}, \gamma = \frac{1}{1-2b}$, we form a weighted tree T_w as follows. T_w has the same topology as T. For an edge $e \in T$ with mutation probability p_e, we assign a weight $-\frac{\log(1-2p_e)}{\log\beta}$ to the corresponding edge of e in T_w. Then, we can show that:

Theorem 6.2. *If $A^{\mathcal{O}}$ reconstructs the tree T_w with probability $1-\delta_1$ using any (τ, ϵ)-restricted oracle $\mathcal{O}$, then there exists B with the same complexity so that $B^{\mathcal{D}_m}$ reconstructs T with probability $1-\delta_1-\delta$ for $m \geq c\beta^{2\tau}\log\frac{n}{\delta}/\epsilon^2$ for some constant $c > 0$.*

Proof: The proof is similar to the uniform mutation probability case and is omitted in this abstract. □

By combining Theorem 5.3 and 6.2, we obtain a tradeoff between the length of the DNA sequences and the number of distance queries needed to reconstruct the tree.

Theorem 6.3. *Let g be the edge-depth of T when considered as a unweighted tree. There exists a constant $c_1, c_2 > 0$ such that if $m \geq c_1\gamma^{10g}\log\frac{n}{\delta}$, then the algorithm reconstructs the tree with probability $1-\delta$ and runs in time $O(n(\kappa(\tau)+\log n)\log g)$ where $\tau = c_2(\log m - \log\log\frac{n}{\delta})/\log\gamma$.*

7 Conclusion

In this paper, we first present a tight lower bound for distance-based evolutionary tree reconstruction. Then, we study the evolutionary tree reconstruction by using the bounded and restricted oracles. We present both lower and upper bounds in these models. We also show that the models we proposed are closely related to the popular DNA-based distance estimation methods. To our knowledge, this paper is the first to study evolutionary tree reconstruction in these models.

Both of our lower and upper bounds are related to the tree cover size: the minimum number of leaves needed to cover all or a constant fraction of leaves. Our work therefore may motivate the study of the tree cover size of evolutionary trees. In particular, it is very interesting to know what is the tree cover size of a typical evolutionary tree. We plan to investigate this question both in practice and in appropriate probabilistic models on the distribution of evolutionary trees.

Our sub-quadratic algorithm requires τ to be greater than $4g$. One open question is that if we can achieve subquadratic algorithm with smaller τ. Our algorithms are more efficient and significantly simpler than the previous DNA-based estimation algorithms and are easy to implement. One future work is to implement and test our algorithms in practice.

References

[1] G. S. Brodal, R. Fagerberg, C. N. S. Pedersen, and A. Östlin. The complexity of constructing evolutionary trees using experiments. In *ICALP*, pages 140–151, 2001.

[2] M. Csürös. Fast recovery of evolutionary trees with thousands of nodes. In *Proceedings of 5th Annual International Conference on Computational Molecular Biology*, pages 104–113, 2001.

[3] M. Csürös and M.-Y. Kao. Provably fast and accurate recovery of evolutionary trees through harmonic greedy triplets. *SIAM Journal on Computing*, 31(1):306–322, 2001.

[4] P. Erdös, M. Steel, L. Székély, and T. Warnow. A few logs suffice to build almost all trees - I. *Random Structures and Algorithms*, 14:153–184, 1999.

[5] P. Erdös, M. Steel, L. Székély, and T. Warnow. A few logs suffice to build almost all trees - II. *Theoretical Computer Science*, 221:77–118, 1999.

[6] M. Farach and S. Kannan. Efficient algorithms for inverting evolution. In *STOC*, pages 230–236, 1996.

[7] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13(1):155–179, 1995.

[8] J. J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of Mathematical Biology*, 51(5):597–603, 1989.

[9] D. Huson, S. Nettles, and T. Warnow. Disk-covering, a fast converging method for phylogenetic tree reconstruction. *J. of Comput. Biology*, 6(3):369–386, 1999.

[10] S. Kannan, E. Lawler, and T. Warnow. Determining the evolutionary tree. In *SODA*, pages 475–484, 1990.

[11] M. Kao, A. Lingas, and A. Ostlin. Balanced randomized tree splitting with applications to evolutionary tree constructions. In *STACS*, pages 184–196, 1999.

[12] J. Kim and T. Warnow. Tutorial on phylogenetic tree estimation. In *Intelligent Systems for Molecular Biology*, 1999.

[13] V. King, L. Zhang, and Y. Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *HP Labs Technical Report HPL-2002-267*, 2002.

[14] M. A. Steel. Recovering a tree from the leaf colourations it generates under a markov mode. *Applied Mathematic Letters*, 7:19–24, 1994.

[15] M. Waterman, T. F. Smith, M. Singh, and W. A. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64:199–213, 1977.