JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 14, Number 6, 2007
© Mary Ann Liebert, Inc.
Pp. 701–723
DOI: 10.1089/cmb.2007.R011

Efficient Enumeration of Phylogenetically Informative Substrings

STANISLAV ANGELOV,[1] BOULOS HARB,[1] SAMPATH KANNAN,[1]
SANJEEV KHANNA,[1] and JUNHYONG KIM[2]

ABSTRACT

We study the problem of enumerating substrings that are common amongst genomes that share evolutionary descent. For example, one might want to enumerate all identical (therefore conserved) substrings that are shared between all mammals and not found in non-mammals. Such collection of substrings may be used to identify conserved subsequences or to construct sets of identifying substrings for branches of a phylogenetic tree. For two disjoint sets of genomes on a phylogenetic tree, a substring is called a *tag* if it is found in all of the genomes of one set and none of the genomes of the other set. We present a near-linear time algorithm that finds all tags in a given phylogeny; and a sublinear space algorithm (at the expense of running time) that is more suited for very large data sets. Under a stochastic model of evolution, we show that a simple process of tag-generation essentially captures all possible ways of generating tags. We use this insight to develop a faster tag discovery algorithm with a small chance of error. However, since tags are not guaranteed to exist in a given data set, we generalize the notion of a tag from a single substring to a set of substrings. We present a linear programming-based approach for finding approximate generalized tag sets. Finally, we use our tag enumeration algorithm to analyze a phylogeny containing 57 whole microbial genomes. We find tags for all nodes in the phylogeny except the root for which we find generalized tag sets.

Key words: design and analysis of algorithms, phylogenetically informative substring, phylogeny, stochastic analysis, suffix tree.

1. INTRODUCTION

Genomes are related to each other by evolutionary descent. Thus, two genomes share sequence identities in regions that have not experienced mutational changes; i.e., the genomes share common subsequences. While common subsequences can also arise by chance, sufficiently long common sequences are homologous (identity by descent) with high probability. The pattern of common subsequences in a set of

[1]Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania.
[2]Department of Biology, University of Pennsylvania, Philadelphia, Pennsylvania.

genomes can be informative for reconstructing the evolutionary history of the genomes. Furthermore, since stabilizing selection for important functions can suppress fixed mutational differences between genomes, long common subsequences can be indicative of important biological function. This hypothesis has been extensively used in comparative genomics to scan genomes for novel putatively functional sequences (Bejerano et al., 2004, 2005; Siepel et al., 2005).

Typical approaches for obtaining such subsequences involve extensive pairwise comparison of sequences using BLAST-like approaches along with additional modifications. Alternatively, one can use a dictionary-based approach, scanning the genomes for presence of common k-mers (which is also the base approach for BLAST heuristics). The presence and absence of such k-mers can be also used to identify unlabeled genomes or reconstruct the evolutionary history. Detection of particular k-mers can be experimentally implemented using oligonucleotide microarrays leading to a laboratory genome identification device.

For detecting functionally important common subsequences or for identifying unlabeled genomes, it is important that k is sufficiently large to ensure homologous presence with high probability. However, the required address space increases exponentially with k. Furthermore, not all patterns of k-mer presence are informative for detecting common subsequences. If the phylogenetic relationship of the genomes is known, then the phylogeny can become a guide to delineating the most informative common subsequences. For any given branch of the tree, there will be substrings common to all genomes on one side of the branch and not present on the other side. For example, there will be a collection of common substrings unique to the mammalian lineage of the Vertebrates. Such substrings will be parts of larger subsequences that are conserved in the mammalian genomes; and, such substrings will be indicators of mammalian genomes. If we had an enumeration of all such informative common substrings, we can apply the information to efficiently detect conserved subsequences or to an experimental detection protocol to identify unlabeled genomes. In this paper, we describe a procedure to efficiently enumerate all such informative common substrings (which we call "sequence tags") with respect to a guide phylogeny. In particular, here we explore the application to the construction of an identification oligonucleotide detection array, which can be applied to high-throughput genome identification and reconstructing the tree of life.

More specifically, given complete genomes for a set of organisms S and the binary phylogenetic tree that describes their evolution, we would like to be able to detect all *discriminating* oligo tags. We will say that a substring t is discriminating at some node u of the phylogeny if all genomes under one branch of u contain t while none of the genomes under any other branch of u contains t. Thus, a set of discriminating tags, or simply tags, for all the nodes of a phylogeny allows us to place a genome *that is not necessarily sequenced* in the phylogeny by a series of binary decisions starting from the root. This procedure can be implemented experimentally as a microarray hybridization assay, enabling a rapid determination of the position of an unidentified organism in a predetermined phylogeny or classification. It is noteworthy that heuristic construction of short sequence tags has been used before for identification and classification (Amann and Ludwig, 2000), but no algorithm has been presented for data driven tag design.

Our results

- We first present an efficient algorithm for enumerating substrings common to the extant sequences under every node of a given phylogeny. The algorithm runs in linear time and space.
- We use our common-substrings algorithm to develop a near-linear time algorithm for generating the discriminating substrings for every node of the phylogeny. Specifically, if S is the set of given genomes, the discriminating-substrings algorithm runs in time $O(n|S|\log|S|)$ where n is the average length of the genomes. This improves the analysis of Angelov et al. (2006) and an earlier bound of $O(n|S|^2)$ given in Angelov et al. (2004).
- Even though all the above algorithms require linear space, due to physical memory limitations they may be impractical for analyzing very large genomic data sets. We therefore give a *sublinear* space algorithm for finding all discriminating substrings (or all common substrings). The space complexity of the algorithm is $O(n/d)$ and its running time is $O(dn|S|^2)$ assuming each genome size is $O(n)$. The tradeoff between the time and space complexities is controlled by the parameter d.
- We demonstrate the existence of tags in the prokaryotes data set of Wolf et al. (2002). The genomes represented in the data set span two of the three recognized domains of life. We find that either left or right tags exist for all nodes of the phylogeny except the root.

- Motivated by our results on the microbial genomes, we study the potential application to arbitrary scale phylogenetic problems using the Jukes-Cantor model of molecular evolution (Jukes and Cantor, 1969). We assume that the given species set S is generated according to this model. We first analyze the case where the phylogeny is a balanced binary tree with a uniform probability of change on all its edges. *We show that in this setting, if t is a tag that discriminates a set S' of species from set $\bar{S}'$, then* w.h.p. *that increases with the number of species—probability $\geq 1/\left(1 + O(\ln(n)/|S|)\right)$—$t$ is present in the common ancestor of S' (occurs early in the evolution) and is absent from the common ancestor of $\bar{S}'$ (is absent from the beginning).* Our study of the stochastic model allows us to design faster algorithms for tag generation with small error. Even when we allow arbitrary binary trees, we show that this probability is $\geq 1/2$.
- As observed in our experiments and subsequent analysis, tags are not guaranteed to exist in a given data set. We consider a relaxed notion of tags to deal with such a scenario. Given a partition $(S', \bar{S}')$ of species, we say that a set T of tags is an (α, β)-*generalized tag set* for some $\alpha > \beta$, if every species in S' contains at least an α fraction of the strings in T and every species in $\bar{S}'$ contains at most a β fraction of them. Clearly, such a tag set can still be used to decide whether a genome belongs to S' or to $\bar{S}'$. We show that the problem of computing generalized tag sets may be viewed as a set cover problem with certain "coverage" constraints. We also show that this generalization of tags is both NP-hard and LOGSNP–hard when $(\alpha, \beta) = (\frac{2}{3}, \frac{1}{3})$. However, if $|T| = \Omega(\log m)$, a simple linear programming based approach can be used to compute approximate generalized tag sets. As an example, we find $(\frac{2}{3}, \frac{1}{3})$-generalized tag sets for the root of the prokaryotes phylogeny (where we did not find tags).

2. PRELIMINARIES

Formally, the problems we consider are the following:

Hierarchical common substring problem (HCS)

Input: A set of strings $S = \{s_1, \ldots, s_m\}$ drawn from a bounded-size alphabet with total length $\sum_{i=1}^{m} |s_i| \leq nm$, where n denotes the average length of the strings; and an m-leaf binary tree P whose leaves are labeled $s_1, \ldots, s_m$ sequentially from left to right.

Goal: For all $u \in P$, find the set of *(right-)maximal* substrings common to the strings in S_u, where S_u is the set of all the input strings in the subtree rooted at u.

A substring t common to a set of strings is right-maximal if for any non-empty string α, $t\alpha$ is no longer a common substring, i.e., t is not a proper prefix of a common substring. The substring t is maximal if it is not a substring of another common substring. Given a set of strings, (right-)maximal common substrings compactly encode all common substrings. Here, we focus on finding right-maximal common substrings. The obtained result generalizes to maximal substrings in a straightforward manner (Gusfield, 1997; Angelov et al., 2004).

Discriminating substrings. A substring t is said to be a *discriminating substring* or a *tag* for a node u in a phylogeny if all strings under one branch of u contain t while none of the strings under the other branch contain t. The input to the discriminating substring problem is the same as that for the first problem.

Discriminating substring problem

Input: A set of strings S and a binary tree P.

Goal: Find sets D_u for all nodes $u \in P$, such that D_u contains all *discriminating* substrings for u.

We will also need the notion of a generalized tag set.

(α, β)-generalized tag set. Given a partition $(S', \bar{S}')$ of species, we say that a set T of tags is an (α, β)-*generalized tag set* for some $\alpha > \beta$, if every species in S' contains at least an α fraction of the strings in T and every species in $\bar{S}'$ contains at most a β fraction of them.

Suffix trees. Suffix trees, introduced in Weiner (1973), play a central role in our algorithms. A suffix tree T of a string s is a trie-like data structure that represents all suffixes of s. We adopt the following definitions from Gusfield (1997). The *path-label* of a node v in T is the string formed by following the path from the root to v. The path-labels of the $|s|$ leaves of T spell out the suffixes of s, and the path-labels of internal nodes spell out substrings of s. Furthermore, the suffix tree ensures that there is a unique path from the root, not necessarily ending at an internal node, that represents each substring of s. We also say that the path-label of node v is *the string corresponding to* v in the tree.

The algorithms we present are based on *generalized suffix trees* (Gusfield, 1997). A generalized suffix tree extends the idea of a suffix tree for a string to a suffix tree for a set of strings. Conceptually, it can be built by appending a unique terminating marker to each string, then concatenating all the strings and building a suffix tree for the resultant string. The tree is post-processed so that each path-label of a leaf in the tree spells a suffix of one of the strings in the set and, hence, is terminated with that string's unique marker.

Proposition 1 (McCreight, 1976; Ukkonen, 1995). *Given a string of length n drawn from a bounded-size alphabet, we can construct its suffix tree in $O(n)$ time and space.*

3. THE HIERARCHICAL COMMON SUBSTRING PROBLEM

Long common substrings among genomes can be indicative of important biological functions. In this section we give a linear time/linear space algorithm that enumerates substrings common to all sequences under every node of a given binary phylogeny. This is a significant improvement over naively running the linear time common substrings algorithm of Hui (1992) for every node of the phylogeny. By carefully merging sets of common substrings along the nodes of the phylogeny and eliminating redundancies we are able to achieve the desired running time. Note that for a given node, there may be quadratically many substrings common to its child sequences. The algorithm will therefore list all *right-maximal* common substrings. Such substrings efficiently encode all common substrings. The formal problem description is given in Section 2. We start with two definitions.

Definition 1. *Let C be a collection of nodes of a suffix tree. A node $p \in C$ is said to be* redundant *if its path-label is empty or it is the prefix of some other node in C.*

Definition 2. *For a tree T, let $o(v)$ be the postorder index of node $v \in T$.*

Algorithm HCS. We preprocess the input as follows: (a) Build a generalized suffix tree T for the strings in S by using two copies of every $s_i \in S$, each with a unique terminating marker: $s_1\#_{1_a}s_1\#_{1_b} \cdots s_m\#_{m_a}s_m\#_{m_b}$; (b) Process T so that lowest common ancestor (*lca*) queries can be answered in constant time; and, (c) Label the nodes of T with their postorder index.

1. For each node $u \in P$, build a list C_u of nodes in T with the following properties:
   - (P1) A substring t is common to the strings in S_u if and only if t is a prefix of the path-label of a node in C_u.
   - (P2) The elements of C_u are sorted based on their postorder index.
   - (P3) No element p of C_u is redundant.

   The lists are built bottom-up starting with the leaves of P:
   - (a) For each leaf $u \in P$, since $|S_u| = 1$, compute C_u by removing the redundant suffixes of $s \in S_u$.
   - (b) For each internal node $u \in P$, let $l(u)$ and $r(u)$ be the left and right children of u respectively. We compute $C_u = C_{l(u)} \sqcap C_{r(u)}$, where

$$A \sqcap B = \{p = lca(a, b) : a \in A,\ b \in B,\ p \text{ not redundant}\}.$$

2. For each $u \in P$, output C_u.

Analysis. The time and space complexities of the preprocessing phase are $O(nm)$ (see Proposition 1) (Harel and Tarjan, 1984; Schieber and Vishkin, 1988). We note that the tree T is obtained by concatenating two copies of each input string terminated with different end markers. This ensures that each suffix of an input string terminates at an internal node of T and simplifies our presentation. The construction is only conceptual and the above property can be emulated using the standard generalized suffix tree method where each string appears only once.

We now analyze Step 1. The lists C_u for the leaves of P are first simultaneously built in Step 1(a) by performing a postorder walk on T. Assuming $S_u = \{s_i\}$, node $p \neq \text{root}(T)$ is appended to list C_u if it has an outgoing edge labeled "$\#_{i_a}$." Suffix tree properties guarantee that C_u will consist of all of the $|s_i|$ suffixes of s_i. Since the list was constructed via a postorder walk on T, it will also possess P2. Property P3 is obtained by scanning each list from left to right and removing redundant nodes. Observe that if p is an ancestor of q, then p is an ancestor of all q' satisfying $o(q) \leq o(q') \leq o(p)$. Hence, we can remove redundancies from each C_u in time linear in $|s_i|$ by examining only adjacent entries in the list. Since every substring of s_i is a prefix of some suffix of s_i, and we removed only redundant suffixes, C_u possesses P1. We obtain the following lemma.

Lemma 1. *C_u possesses properties* P1, P2, *and* P3 *for each leaf $u \in P$.*

We now show how to compute the lists C_u for the internal nodes of P. We first show that the operation $\sqcap$ as defined in Step 1(b) preserves P1.

Lemma 2. *Let $u \in P$ be the parent of $l(u)$ and $r(u)$. If $C_{l(u)}$ and $C_{r(u)}$ possess* P1, *then $C_u = C_{l(u)} \sqcap C_{r(u)}$ also possesses* P1.

Proof. The string t is a common substring to the strings in S_u if and only if t is common to the strings in $S_{l(u)}$ and $S_{r(u)}$. This is equivalent to the existence of $p \in C_{l(u)}$ and $q \in C_{r(u)}$ such that t is a prefix of the path-labels of both p and q. That is, t is a prefix of the path-label of $lca(p, q)$ as required. ■

For each internal node $u \in P$, we construct a merged sorted list Y_u containing all the elements of $C_{l(u)}$ and $C_{r(u)}$ with repetitions. Let $src(a)$ be the source list of node $a \in Y_u$. When computing $C_{l(u)} \sqcap C_{r(u)}$, the following lemma allows us to only consider the lca of consecutive nodes in Y_u whose sources are different.

Lemma 3. *If $a, a', b, b' \in T$ satisfy $o(a') \leq o(a) \leq o(b) \leq o(b')$, then $lca(a', b')$ is an ancestor of $lca(a, b)$.*

Proof. By postorder properties, $lca(a', b')$ is an ancestor of both a and b so it is an ancestor of $lca(a, b)$. ■

Let $a, a', b, b' \in Y_u$, where $o(a') \leq o(a) \leq o(b) \leq o(b')$. If $src(a) \neq src(b)$ and $src(a') \neq src(b')$, then, since $lca(a', b')$ is an ancestor of $lca(a, b)$, the former is redundant. This suggests the following procedure for computing the list C_u starting from the empty list. Suppose at step $i \in \{1, \ldots, |Y_u| - 1\}$, $a = Y_u[i]$ and $b = Y_u[i + 1]$: If $src(a) = src(b)$, proceed to next step; else, let $p' = lca(a, b)$. If $p' = \text{root}(T)$ then we discard it and proceed to the next step. In order to avoid redundancies before appending p' to C_u, we compute $lca(p, p')$ where p is the last node appended to C_u. If $lca(p, p') = p'$ we discard p', and if $lca(p, p') = p$ we replace p with p'.

Each step of the above procedure requires constant time. Hence, since $|Y_u| \leq |C_{l(u)}| + |C_{r(u)}|$, the procedure runs in $O(|C_{l(u)}| + |C_{r(u)}|)$ time. The next lemma shows the correctness of the procedure.

Lemma 4. *For an internal node $u \in P$, the above procedure correctly computes $C_u = C_{l(u)} \sqcap C_{r(u)}$. Furthermore, the list C_u is sorted and $|C_u| \leq \min\{|C_{l(u)}|, |C_{r(u)}|\}$.*

Proof. Since the above procedure performs all necessary lca computations, we only need to show that it maintains P2 and P3 for the resulting list. The proof for properties P2 and P3 proceeds by induction on the steps of the procedure. Let u be an internal node of P. After the first step, $|C_u| \leq 1$, and C_u trivially possesses P2 and P3. Now assume that the two properties are maintained for all $i < k$. We prove that they also hold for $i = k$. Let p be the last node appended to C_u, and let p' be the newly computed node. Note that, by the inductive hypothesis, $o(p) \geq o(q)$, $\forall q \in C_u$. We proceed by cases.

1. $o(p') \geq o(p)$ and p' is an ancestor of p. Then, p' is redundant and we do not append to C_u.
2. $o(p') > o(p)$ and p' is not an ancestor of p. Then, by appending p' to C_u, C_u remains sorted and none of the nodes in C_u will be redundant. Assume, for contradiction, there is a node q such that $o(q) < o(p)$ and p' is an ancestor of q. Then p' is also an ancestor of p; a contradiction.
3. $o(p') < o(p)$ and p is an ancestor of p'. Then, p is redundant and is removed from the list. We now show that C_u will remain sorted after adding p'. Assume it is not. Then, there exists $q \in C_u$ such that $o(p') < o(q) < o(p)$. It follows that p is also an ancestor of q; a contradiction. Finally, since p' is descendant of p, by the inductive hypothesis, it cannot be an ancestor of any other node in C_u.
4. $o(p') < o(p)$ and p is not an ancestor of p'. This case is impossible. Assume $o(p') < o(p)$ and p is not an ancestor of p', and let $p = lca(a, b)$ and $p' = lca(a', b')$ where $a, a', b, b' \in Y_u$. Since p' is generated at a later stage than p, it follows that $o(p) > o(p') \geq \max\{o(a'), o(b')\} \geq \max\{o(a), o(b)\}$. But then p is an ancestor of p'; a contradiction.

Finally, $|C_u| \leq |C_{l(u)}|$ since $\nexists q, q' \in C_u$, $q \neq q'$ where q and q' are ancestors of the same node in $C_{l(u)}$. If they were, one would be redundant. Since P3 ensures that C_u has no redundant nodes, we have that $|C_u| \leq |C_{l(u)}|$. Similarly, we have $|C_u| \leq |C_{r(u)}|$. ■

We are now ready to state the following theorem.

Theorem 1. *The Hierarchical Common Substring Problem can be solved in $O(nm)$ time and $O(nm)$ space.*

Proof. The correctness of the algorithm follows from Lemmas 1, 2, 3, and 4. The time and space requirements for Step 1(a) are bounded by $O(nm) + \sum |s_i| = O(nm)$ since to compute all lists C_u when u is a leaf of P we need to walk T once, and postprocess each list in time proportional to $|s_i|$. For Step 1(b), we need time and space proportional to $\sum_{\text{internal } u} |C_u|$. From Lemma 4, we have $|C_u| \leq \min\{|C_{l(u)}|, |C_{r(u)}|\}$ for all internal nodes $u \in P$. Hence, $|C_u| \leq |C_v|$ where v is the rightmost leaf of u's left subtree; therefore since each leaf node accounts for at most one internal node,

$$\sum_{\text{internal } u} |C_u| \leq \sum_{\text{leaf } v} |C_v| \leq nm.$$

The theorem follows. ■

4. THE DISCRIMINATING SUBSTRING PROBLEM

In this section, we will use the phylogeny for extracting the most informative substrings common to the child sequences of every node in the phylogeny. Suppose we know that a sequence belongs to a certain subtree of the phylogeny that is rooted at u. We wish to know whether the sequence belongs to the left or right branch of u. If we knew the substrings common to the left subtree of u but not present in the right subtree (or vice versa), then we would know to which of the two subtrees the sequence belongs. Hence for a given node, the substrings that are common to its children but not present in the children of its sibling are more informative than only the common ones. Below we show two methods with certain tradeoffs for finding such discriminating substrings or tags, for every node in the phylogeny. It is easy to see that the set of tags obtained by selecting a tag from each node on a root-leaf path *uniquely distinguishes the sequence at the leaf from all other sequences in the phylogeny.*

4.1. A near-linear time algorithm

The HCS algorithm finds all common substrings for each node in P. The common substrings are encoded as the prefixes of the path-labels of the nodes in C_u for each $u \in P$. However, these substrings may not be discriminating. That is, the prefix of the path-label of a node $p \in C_{l(u)}$ (symmetrically $C_{r(u)}$) may also be a substring of one of the strings in $S_{r(u)}$ ($S_{l(u)}$). The following algorithm finds for each node in $C_{l(u)}$ its longest path-label prefix that is not discriminating.

Algorithm. Let C_u for all $u \in P$ be the output of the HCS algorithm and let T be the computed suffix tree.

1. For each $u \in P$, build a list A_u of nodes in T with the following properties:
   (P4) A string t is a substring of a string in S_u if and only if t is a prefix of the path-label of a node in A_u.
   (P5) The elements of A_u are sorted based on their postorder index.
   The lists are built bottom-up starting from the leaves of P:
   (a) For each leaf node $u \in P$, $A_u = C_u$.
   (b) For each internal node $u \in P \setminus \{\text{root}(P)\}$, compute $A_u = A_{l(u)} \cup A_{r(u)}$.
2. For each internal node $u \in P$, compute the set of discriminating substrings encoded with D_u, where,

$$D_u = \left\{ (p, w) : \; p \in C_{l(u)}, \; o(p) < o(w), \; w = lca\left(p, \; \operatorname*{arg\,min}_{q \in A_{r(u)}} [o(lca(p, q))]\right)\right\}.$$

In the above expression for D_u, w is the node in the suffix tree whose path-label is the longest proper prefix of the path-label of p that is present in some string in the right subtree of u. For all $q \in A_{r(u)}$, $\arg\min_{q \in A_{r(u)}} [o(lca(p, q))]$ finds the q that has the deepest lowest common ancestor with p, i.e., the q whose path-label shares the longest prefix with that of p. The condition $o(p) < o(w)$ guarantees that the least common ancestor found is a proper ancestor to p.

Analysis. We first show how each D_u encodes all discriminating substrings for $u \in P$.

Lemma 5. *A string t is discriminating for an internal node $u \in P$ if and only if $\exists (p, w) \in D_u$ such that the path-label of w is a proper prefix of t and t is a prefix of the path-label of p.*

Proof. (if) Let $(p, w) \in D_u$, and suppose t is a string such that the path-label of w is a proper prefix of t and t is a prefix of the path-label of p. By P1, t is a common substring of the strings in $S_{l(u)}$. Assume for contradiction that t is a substring of some string in $S_{r(u)}$. Then, by P4, $\exists q \in A_{r(u)}$ such that t is a prefix of the path-label of q. But then, since w is a proper prefix of t, it is a proper prefix of the path-labels of both p and q. Hence, $o(w) > o(lca(p, q))$; a contradiction.

(only-if) Suppose t is a discriminating string for u. Then, $\exists p \in C_{l(u)}$ such that t is a prefix of the path-label of p, and, by P4, $\nexists q \in A_r(u)$ such that t is a prefix of the path-label of q. Hence, the path-label of $w = lca\left(p, \arg\min_{q \in A_{r(u)}} [o(lca(p, q))]\right)$ is a proper prefix of t. ■

The next corollary, following from the definition of D_u and Lemma 3, will allow us to efficiently compute w as defined in D_u for a given $p \in C_{l(u)}$.

Corollary 1. *Given $p \in C_{l(u)}$. Let $q', q'' \in A_{r(u)}$ be such that*

$$q' = \operatorname*{arg\,max}_{q \in A_{r(u)} : o(q) \le o(p)} [o(q)], \qquad q'' = \operatorname*{arg\,min}_{q \in A_{r(u)} : o(q) > o(p)} [o(q)].$$

If there is no such q' (resp. q''), we set $q' = q''$ (resp. $q'' = q'$). Let

$$w = \operatorname*{arg\,min}_{q \in \{lca(q', p), lca(q'', p)\}} [o(q)].$$

Then, $(p, w) \in D_u$ if and only if $o(p) < o(w)$, or equivalently, $lca(q', p) \neq p$.

We next show how to compute the lists A_u for the internal nodes of P. Note that for a leaf $u \in P$, since $|S_u| = 1$ and C_u is sorted, $A_u = C_u$ trivially possesses both P4 and P5. Furthermore, for an internal node $u \in P$, the union operation maintains P4. Now merging two sorted lists of sizes N and M, with $M \le N$, requires at least $\lceil \log \binom{N+M}{N} \rceil = \Theta(M \log \frac{N}{M})$ comparisons to distinguish among the $\binom{N+M}{N}$ possible placements of the elements of the larger list in the output. We can use the results of Brown and Tarjan (1980, 1979) and Pugh (1990), for example, to match this lower bound. The analysis assumes that the A_u lists are represented as linked-level 2-3 trees (Brown and Tarjan, 1980). Conversion of these lists to 2-3 trees for the leaves is direct since they are sorted.

Lemma 6. *The lists A_u, for all $u \in P$, can be computed in $O(nm \log m)$ time.*

Proof. For the purpose of analysis we assume that the elements in the sets A_u for the leaf nodes of P are all distinct, i.e., $|A_{\text{root}(P)}| = \sum_{\text{leaf } u} |A_u| = O(nm)$. Therefore these elements define an universe of elements, call it U, and $A_u \subseteq U$, for all $u \in P$. Furthermore, note that an element $a \in A_u$, where u is a leaf of P, occurs only in lists A_v, where v is an ancestor of u.

Given an internal node $u \in P$, let S_u (resp. L_u) be the smaller (resp. larger) list of $A_{l(u)}$ and $A_{r(u)}$ breaking ties arbitrarily and define $x_u = |L_u|/|S_u|$. Since at each internal node, we merge the smaller list (S_u) into the larger one (L_u), the running time to compute the lists A_u's is proportional to

$$\sum_{\text{internal } u} |S_u| \log x_u.$$

We obtain a bound on the above quantity in terms of n and m by bounding the contribution of each element of U to the running time. That is, consider an element $a \in U$—for each node u such that $a \in S_u$, we can charge the element a, $O(\log x_u)$ credits to the running time. We compute the total charge to all elements of U as follows.

Given a leaf node $u \in P$, let X_u be the set of ancestral nodes of u in P such that $A_u \subseteq S_v$ and $|S_v| \ge n$ for all $v \in X_u$. (Note that the size of S_v only increases as the distance between v and root(P) decreases.) Let $w \in X_u$ be the closest node to u. Since $|A_v| \ge |S_v|(x_v + 1) \ge |S_v| x_v$, we have $|S_w| \prod_{v \in X_u} x_v \le |A_{\text{root}(P)}| \le nm$; hence $\prod_{v \in X_u} x_v \le m$. Therefore, the contribution of each element $a \in A_u$ to the running time with respect to X_u is proportional to

$$\sum_{v \in X_u} \log x_v = \log \left(\prod_{v \in X_u} x_v \right) \le \log m.$$

Therefore, each element $a \in U$ contributes at most $O(\log m)$ to the running time of all merges where $|S_u| \ge n$. Since $|U| = O(nm)$, the total contribution is $O(nm \log m)$.

Now, consider a node u where $|S_u| < n$. Since $|L_u| \le nm$, it takes $O(|S_u| \log \frac{nm}{|S_u|}) = O(n \log m)$ time to compute the list A_u. Since there are at most $m - 1$ internal nodes, the total contribution of such merges to the running time is $O(nm \log m)$. The lemma follows. ■

Now, we can compute D_u for an internal node $u \in P$ by finding the position of each $p \in C_{l(u)}$ in the sorted $A_{r(u)}$, determining its immediate neighbors q' and q'', and computing w as in Corollary 1. If we consider the elements of $C_{l(u)}$ in their sorted order, then by Brown and Tarjan (1980), and since $|A_{r(u)}| = O(nm)$, finding the positions of all the elements of $C_{l(u)}$ in $A_{r(u)}$ takes $O\left(|C_{l(u)}| \log(nm/|C_{l(u)}|)\right)$ time. Moreover, finding the neighbors of each one of these elements takes constant time. This leads to the following lemma.

Lemma 7. *Given the lists A_u, for all nodes $u \in P$, the lists D_u, for all internal nodes $u \in P$, can be computed in $O(nm \log m)$ time.*

Proof. The running time to compute the lists D_u, for all internal nodes $u \in P$, is proportional to

$$\sum_{\text{internal } u} |C_{l(u)}| \log(nm/|C_{l(u)}|).$$

From the proof of Theorem 1, we have that $\sum_{\text{internal } u} |C_{l(u)}| \le \sum_{u \in P} |C_u| = O(nm)$. The running time is maximized when all of the $m-1$ lists $C_{l(u)}$ are as large as possible and have equal sizes. Therefore, setting $|C_{l(u)}| = O(n)$ we obtain the stated running time. ■

Note that we can *simultaneously* compute the lists C_u, A_u, and D_u, for each internal node $u \in P$, in a bottom-up fashion discarding $A_{l(u)}$ and $A_{r(u)}$ at the end of the computation for each u. Hence, the total size of the A_u lists we store at any point is no more than $\sum_{\text{leaf } v} A_v = O(nm)$. Finally, since, by definition, $|D_u| \le |C_{l(u)}|$, the space required to store C_u and D_u for all $u \in P$ is $O(nm)$. We, therefore, obtain the following theorem.

Theorem 2. *The Discriminating Substring Problem can be solved in $O(nm \log m)$ time and $O(nm)$ space.*

Example. We illustrate the HCS algorithm and the discriminating substring algorithm presented in the previous sections through a concrete example. Consider the phylogeny P and the set of strings S given in Figure 1a. The generalized suffix tree T obtained in the preprocessing phase of the HCS algorithm is given in Figure 1b. Recall that the tree T has the property that every suffix of an input string terminates at an internal node of T.

We first describe how the lists C_v in step 1 of the HCS algorithm are computed for all nodes $v \in P$. For the leaf nodes, we have $C_1 = \{1, 7, 12, 15\}$, $C_2 = \{2, 8, 13\}$, and $C_3 = \{4, 9, 10, 14\}$. These lists are composed of indices from the suffix tree that encode the non-redundant suffixes of strings s_1, s_2, and s_3 respectively. Note that node 6 is not in C_2 because it is an ancestor (prefix) of node 2 and therefore it is redundant. To obtain $C_u = C_1 \sqcap C_2$, we first compute the union Y_u of C_1 and C_2 maintaining the source of each element: $Y_u = \{1, \underline{2}, 7, \underline{8}, 12, \underline{13}, 15\}$; here, underlined elements have C_2 as a source list. Scanning Y_u from left to right, we consider adjacent elements with different source lists and their lowest common ancestors: $lca(1, 2) = 3$, $lca(2, 7) = 16$, $lca(7, 8) = 9$, $lca(8, 12) = 16$, $lca(12, 13) = 14$, and $lca(13, 15) = 16$. We obtain $C_u = \{3, 9, 14\}$ by removing redundancies (none in this case) and the root of the suffix tree (which represents the empty string). The list C_u encodes the common substrings of s_1 and s_2; i.e., the strings ACG, CG, G, and all of their proper prefixes: AC, A, and C. Similarly, we have $Y_r = \{3, \underline{4}, 9, \underline{9}, \underline{10}, 14, \underline{14}\}$ and $C_r = \{5, 9, 14\}$.

The discriminating substrings algorithm then prunes the set of common substrings for each node as follows. In step 1, the algorithm computes the lists A_v for all nodes $v \in P \setminus \{r\}$. We have $A_1 = C_1$,

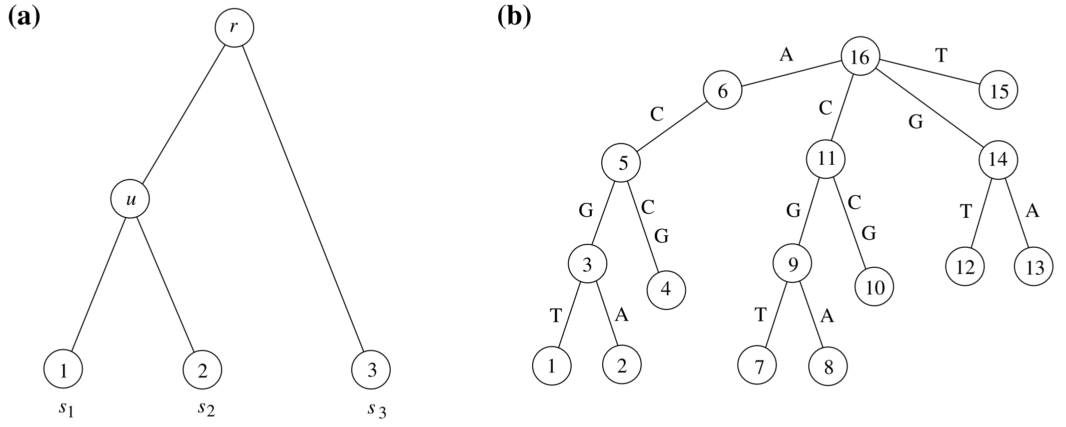

FIG. 1. An example phylogeny P with extant species $S = \{s_1, s_2, s_3\}$, and their corresponding generalized suffix tree. **(a)** The phylogeny P whose leaf nodes are associated with the strings s_1 = ACGT, s_2 = ACGA, and s_3 = ACCG. Leaf nodes are labeled with 1, 2, and 3, and the internal nodes with u and r (the root). **(b)** The generalized suffix tree T for the strings s_1, s_2, and s_3 constructed from $s_1\#_{1_a}s_1\#_{1_b}s_2\#_{2_a}s_2\#_{2_b}s_3\#_{3_a}s_3\#_{3_b}$. Leaf nodes (with incoming edges labeled $\#_{1_a}, \#_{1_b}, \#_{2_a}, \#_{2_b}, \#_{3_a}, \#_{3_b}$) are omitted for clarity of presentation. Internal nodes are labeled with their postorder index. The index of the root node is 16.

$A_2 = C_2$, $A_3 = C_3$, and $A_u = C_1 \cup C_2$. To reduce the space requirement, once A_u is computed, lists A_1 and A_2 are discarded. Now to compute the discriminating substrings for, say, node r, we need to prune C_u using A_3 to obtain the left tags, and we need to prune C_3 using A_u to obtain the right tags. The calculations for the left tags are as follows. For each element of C_u we efficiently find its immediate neighbors in A_3. For node 3, the neighbor is 4; for node 9, the neighbors are 9 and 10; for node 14, the neighbor is 14. Applying the calculations in Corollary 1, we obtain $D_u = \{(3, 5)\}$. The pair $(3, 5)$ encodes all prefixes of ACG (node 3) that are longer than AC (node 5); i.e., it identifies the single discriminating tag ACG.

4.2. *A sublinear space algorithm*

The above algorithms are optimal or near optimal in terms of their running times and they require only linear space. For very large genomes, even linear space might not fit in primary memory. It is important to further reduce the algorithms' space requirements for such situations to avoid expensive access to secondary storage. Intuitively, it seems that we should be able to run the algorithms on chunks of the data at a time in order to reduce the space complexity. Below we describe such a *sublinear* space algorithm, with a time-space tradeoff, for finding all discriminating substrings (or all common substrings). The precise tradeoff is stated in Theorem 3. The running time of the algorithm will also depend on the underlying structure of the input phylogeny, specifically its *height*. Recall that the height of a tree P is equal to the maximum number of edges on a simple root-leaf path.

In this section, we assume that the maximum genome length is $O(n)$ where n is the average genome length. Such an assumption is reasonable when dealing with genome data from relatively close species. When this condition does not hold, the running time of the stated algorithm increases by an additional factor proportional to the ratio of the maximum to average genome length in the input.

Algorithm outline and analysis. The algorithm proceeds by considering each node of the phylogeny P independently. For a node $u \in P$, we can find the set of discriminating substrings D_u by using the matching statistics algorithm introduced in Chang and Lawler (1994). Given strings s and s', the algorithm computes the length $m(s, j, s')$ of the longest substring of s starting at position j in s and matching some substring of s'. This is done by first constructing the suffix tree for s' and then walking the tree using s. The algorithm requires $O(|s'|)$ time and space for the construction of the suffix tree, and $O(|s|)$ additional time and space to compute and store $m(s, j, s')$ for all j.

Let S_u be the union of two disjoint sets L_u and $R_u = S_u \backslash L_u$ where L_u and R_u are the sets of strings under the two branches of $u \in P$. A substring starting at position j of $s \in L_u$ is discriminating for u if and only if

$$\min_{s' \in L_u} m(s, j, s') - \max_{s' \in R_u} m(s, j, s') > 0. \qquad (1)$$

That is, there exists a substring of s starting at j that is common to all strings in L_u and is sufficiently long so that it does not occur in any string in R_u. If a position j satisfies (1), then a substring t starting at j such that $\max_{s' \in R_u} m(s, j, s') < |t| \leq \min_{s' \in L_u} m(s, j, s')$ is discriminating. Clearly, all tags are substrings of s and thus the outlined procedure computes D_u. The running time of the algorithm is $\sum_{s' \in S_u} O(|s| + |s'|) = O(nm)$ and requires $O(\max_{s' \in S_u} |s'|) = O(n)$ space. Computing the set of tags for all nodes of P with tree height h requires $O(nmh)$ time and $O(n)$ space matching the running time of Angelov et al. (2004). By limiting the maximum allowed length of a tag, we can obtain a tradeoff between the running time and memory required by the algorithm as stated in the following theorem.

Theorem 3. *The Discriminating Substring Problem for tags of length $O(n/d)$, for some threshold parameter d, $1 \leq d \leq n$, can be solved in time $O(dnmh)$ and $O(n/d)$ space, where h is the height of P.*

Proof. We will show the required modifications to the above algorithm in order to include the desired time-space tradeoff parameter d. Given the length of tags of interest is at most n/d, the algorithm can be adapted to use $O(n/d)$ memory and $O(dnm)$ time to compute the set of all discriminating tags D_u for a node u of P by virtually chopping each input string in $O(d)$ overlapping segments of length $2n/d$. For a string $s \in S_u$ and an integer $i \in [0, |s|/d)$, let $s(i)$ denote the segment that starts at position $\frac{n}{d}i + 1$.

Note that the overlap between consecutive segments $s(i)$ and $s(i+1)$ ensures that each substring of s with length at most n/d is contained in some segment. Since tags must occur in all strings in L_u, we pick a representative string $s \in L_u$ and find tags contained within each of its segments. For a segment $s(i)$, we can find all its discriminating tags by computing $m(s(i), j, s') = \max_{s'(k)} m(s(i), j, s'(k))$ instead of $m(s, j, s')$ in expression (1) above. Computing $m(s(i), j, s')$ for all positions j in $s(i)$ takes $O(n)$ time since there are $O(d)$ segments in each string s' each with length $O(n/d)$. This implies that the computation of expression (1) for each $s(i)$ takes $O(nm)$ time as there are $O(m)$ strings in L_u and R_u. Once we process the current segment of s and produce the tags that are contained within it, we add that segment to the set R_u to avoid generation of duplicate tags. We then repeat the procedure for the remaining segments of s. The total running time per node is then $O(dnm)$ as, again, there are $O(d)$ segments in s.

Since we need to process only two segments at a time, we need only $O(n/d)$ space. ■

Remark. Since a tag may occur more than once in a segment, we can further eliminate duplicates by maintaining the relevant matching statistics information at the nodes of the suffix tree for the current segment. Then, non-redundant tags can then be output in a bottom-up fashion. Note that the generalized suffix tree of two strings can be obtained by augmenting the suffix tree of the first sequence (Ukkonen, 1995). This allows for quickly identifying the correspondence between the nodes of the two trees. Therefore, the modification does not affect the asymptotic time and space requirements.

5. DISCRIMINATING TAGS UNDER A STOCHASTIC MODEL OF EVOLUTION

We now analyze statistical properties of tags using a simplified assumption of molecular evolution. We make the first steps toward understanding the capability of tags to place new sequences in a given phylogeny and their application to arbitrary scale phylogenetic problems. We show that there is a *primary mechanism* for generating tags which suggests that tags are indicative of shared evolutionary history.

We use the *Jukes-Cantor* model (Jukes and Cantor, 1969) for our analysis. In this model each position in the genome evolves independently according to an identical stochastic process where the probability of a mutation occurring per unit time is given by a parameter λ. Further, it is assumed that the probability λ of change is distributed equally between the 3 possible changes at a site. Thus if a site currently has the nucleotide A, then it has probability $\lambda/3$ of changing to C, for example, in unit time. When branching occurs at a node in a phylogeny, then the two branches start with identical sequences but evolve independently according to the stochastic process. Finally, we assume that the sequence at the root of the phylogeny is a random sequence of length n. Since we only allow substitutions all genomes will have the same length. Given the actual time durations between evolutionary events, it is possible to represent the Jukes-Cantor model by specifying the probabilities of change along each edge in the phylogeny where these probabilities depend on the time duration represented by the edge (e.g., if an edge is infinitely long, the probability of change is 3/4).

All of the current phylogeny models are some version of continuous time homogeneous Markov chain models. From an event point of view, these are all Poisson counting processes with event rates determined as some function of the model parameters and the different models only determining the marginal probability of state transitions. For our purposes the main determinants of the tags are where in the trees the events occur so the analysis on the Jukes-Cantor model should be reflective of the general cases.[1] Even with this simple model, obtaining a closed-form representation of tag length distribution as a function of the probabilities of change along each edge is a complex task. We therefore start with a simplifying assumption—the phylogeny is a complete binary tree and the probability of change along each edge is p. We let h be the height of our tree and label the sequences at its leaves with $s_1, \ldots, s_{2^h}$. We label the sequence at the root with r. We will focus on tags present in the left subtree of the root, which we call *left tags*. Similar analysis holds for right tags and for other nodes in the tree. In Section 5.3, we generalize the analysis to arbitrary binary tree topologies and probabilities of change along the edges.

[1]Our simulation results can be reviewed at: *www.cis.upenn.edu/~angelov/phylogeny/experiments/simulation/*.

5.1. The primary mechanism for generating tags

Given the stochastic model of evolution we show that there is a dominant process by which tags are generated. We first prove that if the probability of change p along an edge is more than $\ln(n)/(2^{h-2}k)$, we do not expect tags to be generated. Using this bound on p, we show in Theorem 4 that the primary mechanism by which a tag t that discriminates a set S' of species from set $\bar{S}'$ arises is one where t is present in the common ancestor of the species in S' and is absent from the common ancestor of those in $\bar{S}'$. In particular, if we let T denote the set of all tags and T' the set of tags generated by the primary mechanism, then we show that $|T| \leq |T'|(1 + O(\frac{\ln n}{|S' \cup \bar{S}'|}))$. Thus the error term decays inversely in the number of species. We start with the following two lemmas bounding the minimum tag length and the maximum probability of change p.

Lemma 8. *Tags have length greater than* $(1-\epsilon)\log_4 n$ w.h.p. *where* $0 < \epsilon < 1$.

Proof. Consider a sequence s in the right subtree of r. The sequence s is uniformly distributed since it evolved from a random sequence under the Jukes-Cantor model. Now let $k \leq (1-\epsilon)\log_4 n$ and consider a k-mer t. If we partition s into strings of length k, then the probability that t does not appear in s is at most $(1 - 4^{-k})^{n/k} \leq e^{-\frac{n^{\epsilon}}{(1-\epsilon)\log_4 n}}$. Summing this probability over all possible k-mers, we get that the probability some k-mer does not appear in s is upper bounded by $e^{\frac{-n^{\epsilon}}{(1-\epsilon)\log_4 n} + (1-\epsilon)\ln n}$, which is negligible. ■

Lemma 9. *If* $p > \frac{3\ln n}{k(2^h - 2)}$, *the expected number of tags of length* k *is* < 1.

Proof. The left subtree has 2^{h-1} leaves. Consider the character at ith position of the leaf s_1. The probability that a leaf s_j, $j \neq 1$, has the same character at position i is upper bounded by $1 - 2p/3$. The probability that all leaves in the left subtree agree in the ith position is thus bounded by

$$(1 - 2p/3)^{k(2^{h-1}-1)} \leq \exp(-2pk(2^{h-1} - 1)/3),$$

which is less than $1/n$ when $p > 3\ln n/(k(2^h - 2))$. ■

Henceforth, we will assume that $p \leq 3\ln(n)/(k(2^h - 2))$. Let A_i for $1 \leq i \leq n - k + 1$ be the event that position i in the root sequence, r, is *good*. Position i is said to be good if the k-mer starting at i in the left child of r differs from that in the right child. Therefore, $\Pr[A_i = 1] = 1 - (p^2/3 + (1-p)^2)^k$. If the event A_i results in a tag being generated, we will say that this tag is a *type–I* tag. The following theorem shows that type–I tags are dominant.

Theorem 4. *Let t be a sequence that either does not occur at the left child of the root or occurs at the right child of the root. Then the probability that any such t is a left tag is negligible compared to the probability of type–I tags.*

Proof. Suppose that t is a left tag that appears in the ith position of all the sequences at the leaves of the root's left subtree. Let t_l and t_r be the ith k-mers in the root's left and right children respectively.

First, we will show that $\Pr[t = t_l \mid t \text{ is a left tag}]/\Pr[t \neq t_l \mid t \text{ is a left tag}] \geq 3/(8p)$. We will assume that when $t \neq t_l$, the two k-mers differ in exactly one position. (The above ratio only gets better if the number of differing positions is more than one.) Hence, independent of the value of the ith k-mer in the root, $\Pr[t = t_l]/\Pr[t \neq t_l] \geq p/3$.

The tag t can be generated by two processes, one starting with $t = t_l$ and the other starting with $t \neq t_l$. In the latter case, consider the position j that causes t to differ from t_l, i.e., $t_l(j) \neq t(j)$. Let E be the set of maximal edges (closest to the root) such that for each edge $e \in E$, the jth position becomes equal to $t(j)$ for the first time at the node below e. Now let $N(i)$ be the number of ways of having such i maximal changes. We know that $N(2) = 1$ and $N(3) = 2$. In general, $N(i) = 2N(i-1) + \sum_{j=2}^{i-2} N(j)N(i-j)$.

Hence, the probability that the jth position in every leaf is equal to $t(j)$ is at most $\sum_i \left(\frac{p}{3}\right)^i N(i)$. Note that $N(i+1)$ is the ith Catalan number C_i (see, for example, Stanley [1999]); therefore,

$$\sum_i \left(\frac{p}{3}\right)^i N(i) = \frac{p}{3}\left(\sum_i \left(\frac{p}{3}\right)^i C_i - 1\right) = \frac{1-\sqrt{1-4p/3}-2p/3}{2} \leq \frac{4p^2}{9}.$$

The corresponding probability for the case when $t_l = t$ is lower bounded by $(1-p)^{2^h-2}$, which is the probability of no changes in the left subtree to the jth position. Assuming t has length $\Omega(\ln n)$ and using Lemma 9 we find that this lower bound is at least $1/2$. Hence, the desired ratio is at least $(p/3)\frac{1/2}{4p^2/9} = \frac{3}{8p}$.

It remains to show a similar result for the right side; namely, that $\Pr[t \neq t_r \mid t \text{ is a left tag}]/\Pr[t = t_r \mid t \text{ is a left tag}] = O(2^h)$. We will start with the assumption that $t_l = t$ since we showed that this is the predominant way for generating left tags. Let p' be the probability of some change along an edge in a given k-mer. That is, $p' = 1-(1-p)^k \leq pk$. Now, $\Pr[t \neq t_r]/\Pr[t = t_r] \geq 1-((1-p)^2 + p^2/3)^k \geq 1-(1-p)^k = p'$. Using an argument similar to that above, we have that when $t = t_r$, the probability t does not appear in the sequences at the leaves of the root's right subtree is at most $4p'^2$. Further, when $t \neq t_r$, the probability that the discriminating position is preserved is at least $(1-p)^{2^h-2} \geq 1/2$ by making the same assumption on the length of t. Hence, the desired ratio is at least $p'\frac{1/2}{4p'^2} \geq 1/(8pk)$. ■

Expected number of length k tags. Define B_i for $1 \leq i \leq n-k+1$ to be the event that the ith k-mer at each leaf of the left subtree agrees with that at the root of the left subtree. A lower bound on $\Pr[B_i = 1]$ is obtained when there are no changes in the left subtree. That is,

$$\Pr[B_i = 1] \geq (1-p)^{\#\{\text{edges in the left subtree of } r\}\cdot k} = (1-p)^{(2^h-2)k}$$

One way a type–I tag is generated is if A_i occurs, the k-mer does not change anywhere in the left subtree and a position that changed due to the occurrence of A_i remained unchanged in the right subtree. Let the random variable X_i indicate if a type–I tag of length k occurs at position i. Then,

$$E[X_i] \geq \Pr[A_i = 1]\Pr[B_i = 1](1-p)^{2^h-2}.$$

Finally, let the random variable X equal the number of tags of length k. Then, $X \geq \sum_{i=1}^{n-k+1} X_i$, implying that,

$$E[X] \geq (n-k+1)E[X_i].$$

5.2. *A sampling based approach*

Consider the phylogeny described above, and suppose event B_i occurred. That is, suppose that the k-mer starting at position i is common to all the sequences in the left branch of the root. Call this k-mer t_i. Let $R = \{s_{2^{h-1}+1}, \ldots, s_{2^h}\}$ be the set of sequences at the leaves of the right subtree of r. For t_i to be discriminating, it should not occur in any of the sequences in R. Instead of testing the occurrence of t_i in every one of those sequences, we will only test a sample of those sequences. Let $\mathcal{M}$ be the sample we pick. We will consider t_i to be a tag if it does not occur in any of the sequences in $\mathcal{M}$. If t_i is a tag, then our test will succeed. However, we need to bound the probability that we err. Specifically, we bound the ratio of the expected number of false positive tags to the expected number of tags our algorithm produces.

Algorithm. We use the sampling idea to speed up our tag detection algorithm:

1. Run the HCS algorithm to compute C_u for all u in our phylogeny P.
2. For each $u \in P$,
   - Pick a set $\mathcal{M}_u$ of sequences from the right subtree of u.
   - For each $s \in \mathcal{M}_u$, trim $C_{l(u)}$ as in Step 2 of the algorithm in Section 4.1.

Assuming $\mathcal{M}$ is the sample of maximum size, then the running time of the above algorithm is $O(nm|\mathcal{M}|)$.

Sampling error. How well does the sampling based approach work? Even with a sample of constant size, the probability that we err decreases with the tag size k. Theorem 4 shows that if t occurs at the right child of the root, then t is not a left tag *w.h.p.* Hence, assuming that the k-mer t is a left tag at position i, we need only consider the case when the right child of the root contains a k-mer $t' \neq t$ at position i. We do not err when a differentiating bit in t' is preserved in the right subtree which is at most $(1-p)^{2^h-2}$ implying the following theorem.

Theorem 5. *The sampling algorithm errs with probability* $< 1/2$ *for* $k = \Omega(\ln n)$.

5.3. General tree topologies

We generalize our stochastic analysis to arbitrary binary topologies and probabilities of change along edges of the phylogeny. Given a phylogeny P with root r, let L (resp. R) be the total length of the edges in the left (resp. right) subtree of r, and let E be the total length of the two edges incident on r. Recall that at a given site a nucleotide changes to one of the three remaining nucleotides with a rate of λ per year. Hence, the position i will experience x number of mutations on a branch of length ℓ with probability $e^{-\lambda\ell}(\lambda\ell)^x/x!$. Again, we will focus on left tags occurring at homologous sites. The following is the analog of Lemma 9.

Lemma 10. *Let* $k > \zeta \log_4 n$ *where* $\zeta < 2$ *is a constant. If* $\lambda L > \frac{\zeta \log_4 n}{k}$, *the expected number of left tags of length* k *is less than 1.*

Proof. A necessary condition for a left tag of length k to exist at position i is the agreement of all the ith k-mers at the leaves of the left subtree of r. Clearly, the results of this section hold for right tags also. Let A_i be the event that the leaves of the left subtree agree at position i. Then,

$$\Pr[A_i] \leq e^{-\lambda L} + \frac{1}{3}(1-(1+\lambda L)e^{-\lambda L}) = \hat{P}(A_i).$$

The first term of the upper bound $\hat{P}(A_i)$ is the probability the ith position will not experience any changes in the left subtree, while the second term is the probability that it will experience at least two changes that result in agreement. Note that two or more changes lead to agreement with probability at most $1/3$. Since the left subtree has more than one branch, one change in the subtree cannot result into an agreement. If $(\hat{P}(A_i))^k$, is bounded from above by $1/n$, then $(\Pr[A_i])^k < 1/n$ implying that we do not expect any tags. Hence, we wish to find the range of λL such that

$$\left(\hat{P}(A_i)\right)^k < \frac{1}{n}.$$

Let $\zeta = \left(\log_4\left(\frac{3e}{1+e}\right)\right)^{-1} \simeq 1.77$. We know that k, the length of the tag, satisfies,

$$k = c \log_4 n, \tag{2}$$

where $c > 1$. If we restrict $c > \zeta$, and if $\lambda L \geq \zeta/c$, then $\hat{P}(A_i) \leq 1/\sqrt[c]{4}$ implying from (2) that $(\hat{P}(A_i))^k < 1/n$, i.e., we do not expect tags. ■

Assuming that both left and right tags occur in the given phylogeny, we show that type–I tags constitute the majority of tags if $\lambda E = \Omega(1/k)$.

Theorem 6. *The probability that a tag* t *is of type–I is* $> 1/2$ *if* $\lambda E = \Omega(1/k)$.

Proof. Suppose both left and right tags exist in the phylogeny P. Consider a left tag t of length k occurring at position i, and let t_l and t_r be the ith k-mers in the root's left and right children respectively. We show that with probability greater than $1/2$, $t = t_l$ and $t \neq t_r$.

In order to show the desired probability we will show that under certain assumptions on λE,

$$\frac{\Pr[(t \neq t_l \vee t = t_r) \wedge t \text{ is a left tag}]}{\Pr[t = t_l \wedge t \neq t_r \wedge t \text{ is a left tag}]} \leq 1. \tag{3}$$

We call the k-mer starting at position i of a leaf sequence in the left subtree a *possible* left tag (*p-left tag*) if it is present at the ith position in every leaf of the left subtree. Now let,

$$R_1 = \frac{\Pr[t \neq t_l \mid t \text{ is a p-left tag}]}{\Pr[t = t_l \mid t \text{ is a p-left tag}]}, \quad \text{and}$$

$$R_2 = \frac{\Pr[t = t_r \wedge t \text{ is a left tag} \mid t = t_l \wedge t \text{ is a p-left tag}]}{\Pr[t \neq t_r \wedge t \text{ is a left tag} \mid t = t_l \wedge t \text{ is a p-left tag}]}.$$

Then, in order to show (3), it suffices to show,

$$\frac{1}{\Pr[t \neq t_r \wedge t \text{ is a left tag} \mid t = t_l \wedge t \text{ is a p-left tag}]} R_1 + R_2 \leq 1. \tag{4}$$

We will assume that when $t \neq t_l$, the two k-mers differ in exactly one position. The ratio R_1 only gets better if the number of differing positions is more than one. Hence,

$$R_1 \leq \frac{(1/3)(1 - e^{-\lambda L/2})^2}{e^{-\lambda L}}, \quad \text{and} \quad R_2 \leq \frac{e^{-\lambda E k}(1 - e^{-\lambda R k/2})^2}{(1 - e^{-\lambda E k})\, e^{-\lambda R}}.$$

Substituting into (4) we obtain

$$\lambda E \geq k^{-1} \ln \left[\frac{e^{-\lambda R} + (1 - e^{-\lambda R k/2})^2}{e^{-\lambda R} - (1/3)(e^{\lambda L/2} - 1)^2} \right].$$

The right-hand side of the above lower bound is maximized when λL and λR are maximized. Setting $\lambda L = \lambda R = 1$, we get $\lambda E \geq k^{-1} \ln[(e^{-1} + 1)/(e^{-1} - (\sqrt{e} - 1)^2/3)] \simeq 1.794/k$. This lower bound on λE is a constant factor away from the best possible bound if $n \geq 2$. The upper bound on the logarithmic term is minimum when λL and λR equal their least upper bounds. That is, setting $\lambda L = \lambda R = (\zeta \log_4 n)/k$, we have $\lambda E > 0.2/k$ when evaluated at $n = 2$. ■

6. GENERALIZED TAG SETS

The stochastic analysis in Section 5 shows that tags may not always exist even in data sets generated by stochastic evolutionary processes. When tags are not present, we can relax the definition of discriminating substrings and still be able to distinguish if a genome comes from a node's left or right subtree. Recall that given a partition $(S', \bar{S}')$ of species, we say that a set T of tags is an (α, β)-generalized tag set for some $\alpha > \beta$, if every species in S' contains at least an α fraction of the strings in T and every species in $\bar{S}'$ contains at most a β fraction of them. Hence, given a tag set T, we can determine whether a species s belongs to S' or to $\bar{S}'$ by counting the number of tags in T which s contains. We next show that the problem of computing generalized tag sets may be viewed as a set cover problem with certain "coverage" constraints. *W.l.o.g.* assume we are computing generalized tag sets at the root.

(α, β)–Set Cover problem

Input: A universe $U = U' \cup U''$ of m elements and a collection $\mathcal{S}$ of subsets of U.

Goal: Find a minimum size subcollection $\mathcal{C}$ of $\mathcal{S}$ such that each element of U' is contained in at least $\alpha|\mathcal{C}|$ sets in $\mathcal{C}$, and each element of U'' is contained in at most $\beta|\mathcal{C}|$ sets in $\mathcal{C}$.

In the problem definition above, the set U corresponds to the m input strings each of length n, with U' and U'' being the strings in the left and right subtrees of the root of the given phylogeny. Each $S_i \in \mathcal{S}$

represents the set of strings that share a substring t_i drawn from a suitable collection of substrings with cardinality $O(n^2m)$. In Angelov et al. (2004), it was shown how to efficiently compute and represent the corresponding sets of all substrings in $O(nm^2)$ time and space with the help of a generalized suffix tree. A biologically motivated pruning sub-step may be applied to reduce their number (Matveeva et al., 2003). Note that the pruning should be performed on the input rather than the output since removing elements from the solution set may decrease its ability to discriminate. We also note that the Discriminating Substring Problem corresponds to the $(1, 0)$–Set Cover Problem when the objective is to maximize the size of $\mathcal{C}$ since we find all tags.

The next theorem follows via a reduction from the decision version of Set Cover. In the reduction, the size of all feasible subcollections $\mathcal{C}$ is the same, hence the result holds even for the *existence and maximization* versions of the problem. The reduction relies on the construction of a collection $\mathcal{Q}$ of subsets of U'' such that for each proper subcollection of $\mathcal{Q}$, there is an element that appears in more than $(\frac{\beta}{\alpha})$ fraction of the sets while each element occurs in exactly $(\frac{\beta}{\alpha})$ fraction of the sets in $\mathcal{Q}$. By suitably padding $\mathcal{Q}$ with elements of U', we are able to show that any feasible solution should include all of $\mathcal{Q}$ together with a set cover of the original instance; thus, we obtain a guarantee on the solution size. For ease of presentation, the theorem is shown for $(\alpha, \beta) = (2/3, 1/3)$. The analysis extends in a natural way for rational α and β such that $\alpha = 1 - \beta$ and $\beta = 1/c$ for a fixed integer $c > 2$.

Theorem 7. *$(\frac{2}{3}, \frac{1}{3})$-Set Cover is NP-hard.*

Proof. Let $\langle \mathcal{S}, k\rangle$ be an instance of Set Cover. Let U denote the universe and let $a \notin U$ be an additional element. We construct a collection $\mathcal{S}' = \mathcal{S} \cup \mathcal{Q}$ where $\mathcal{Q}$ is a collection $\mathcal{Q} = \{Q_0, \ldots, Q_{q-1}\}$ of size $q = \frac{2/3}{1-2/3}k = 2k$. Assume *w.l.o.g.* that k is a power of 2 and therefore $q = 2^z$ for some integer $z \geq 1$. Let $U' = U \cup \{a\}$ be the set of *positive* elements and let U'' with size $|U''| = \sum_{i=1}^{z} 2^i = 2^{z+1} - 2$ be the set of *negative* elements. Collection $\mathcal{Q}$ is such that $Q_i \cap U' = U \cup \{a\}$, for $i > 0$, and $Q_0 \cap U' = \{a\}$. Furthermore, each negative element is distributed in exactly $q/2$ sets of $\mathcal{Q}$ such that for each non-empty proper subcollection $\mathcal{Q}' \subset \mathcal{Q}$, there is a negative element contained in $> |\mathcal{Q}'|/2$ of the sets of $\mathcal{Q}'$. We obtain an instance of the $(\frac{2}{3}, \frac{1}{3})$-Set Cover problem $\langle \mathcal{S}', k + q\rangle$.

Any solution to the constructed instance must include $q' \geq 1$ sets of $\mathcal{Q}$ since $a \in U'$ is only present in the sets of $\mathcal{Q}$. Thus, any solution has size no more than $\frac{3}{2}q'$. Since for any $q' < q$ there is a negative element present in more than $q'/2$ of the sets of $\mathcal{Q}'$, a feasible solution must include all of $\mathcal{Q}$. Finally, each element of U is covered by $q - 1 = \frac{2}{3}(q + k) - 1$ sets of $\mathcal{Q}$; therefore, the $(\frac{2}{3}, \frac{1}{3})$-Set Cover instance is a YES instance if and only if there is a Set Cover of U with at most k sets of $S = \mathcal{S}' \setminus \mathcal{Q}$.

It remains to show how the negative elements of U'' are distributed among the sets of $\mathcal{Q}$. Recall that $q = 2^z$. Denote the negative elements by b_{ij}, for $1 \leq i \leq z$ and $0 \leq j < 2^i$. For a given i, we add element b_{ij} to sets of $\mathcal{Q}$ with indices (Table 1),

$$j + (2^i x + y) \bmod q,$$

for $x \in [0,\ 2^{z-i})$, and $y \in [0,\ 2^{i-1})$. In other words, the element b_{ij} is included in sets of $\mathcal{Q}$ with indices that span 2^{z-i} index intervals each of length 2^{i-1}. The offset of the first interval of b_{ij} is j and the distance between the consecutive intervals is equal to 2^{i-1}. Note that each b_{ij} is contained in exactly $q/2$ of the sets.

Claim 1. *Let $\mathcal{Q}$ be constructed as above. For any non-empty $\mathcal{Q}' \subset \mathcal{Q}$, there exists a negative element present in more than $|\mathcal{Q}'|/2$ sets of $\mathcal{Q}'$.*

TABLE 1. DISTRIBUTION OF THE NEGATIVE ELEMENTS IN THE COLLECTION $\mathcal{Q}$ OF THEOREM 7 (FOR $z = 2$)

$Q_0 \cap U'' = \{$	$b_{10},$		$b_{20},$			b_{23}	$\}$
$Q_1 \cap U'' = \{$		$b_{11},$	$b_{20},$	$b_{21},$			$\}$
$Q_2 \cap U'' = \{$	$b_{10},$			$b_{21},$	$b_{22},$		$\}$
$Q_3 \cap U'' = \{$		$b_{11},$			$b_{22},$	b_{23}	$\}$

Proof. Let $q' = |Q'|$. We show by reverse induction on i, $1 \leq i \leq z$, $0 \leq j < 2^i$, that if every negative element is present in at most $\lfloor q'/2 \rfloor$ sets then $Q' = Q$. The inductive hypothesis for i states that if we pick a set $Q_p \in Q'$ then we should also pick sets with indices $p + 2^{i-1}x \bmod q$ for $x \in (0, 2^{z-i+1}]$. Therefore at $i = 1$, we must pick 2^z sets, or all of $\mathcal{Q}$.

Hereonafter, all arithmetic on indices is modulo q. For the base case set $i = z$. It is easy to see that $q' \equiv 0 \pmod 2$. Furthermore, the element b_{zj}, for all j, should be picked in exactly $q'/2$ sets since the elements b_{zj} and $b_{z(j+2^{z-1})}$ partition the sets of $\mathcal{Q}$ (and therefore $\mathcal{Q}'$) into two equal parts. Now look at the elements b_{zp} and $b_{z(p+1)}$ where p is the index of a picked set. The interval of indices they span overlap except for p and $p + 2^{z-1}$. Since Q_p is picked, it follows that $Q_{p+2^{z-1}}$ is also picked in order $b_{z(p+1)}$ to be covered by the same number of sets as b_{zp}. Suppose the inductive hypothesis is true for some $i > 1$, then we show that it also holds for $i - 1$ by a similar argument considering the overlapping intervals of the elements $b_{(i-1)j}$. ■

The theorem follows. ■

Theorem 8. *$(\frac{2}{3}, \frac{1}{3})$-Set Cover is LOGSNP-hard.*

Proof. We show that the $(\frac{2}{3}, \frac{1}{3})$-Set Cover Problem restricted to solutions of size $O(\log m)$ or $O(\log n)$, where m is the size of the universe and n is the number of sets, is LOGSNP-hard by reduction from Tournament Dominating Set (Megiddo and Vishkin, 1988; Papadimitriou and Yannakakis, 1996). Given a tournament and an integer k, the Tournament Dominating Set Problem asks if there is a dominating set of size at most k. The problem can be restricted to $k \leq \lceil \log n \rceil$—where n is the number of vertices—since a greedy strategy always returns a solution of size at most $\lceil \log n \rceil$.

We first show how to encode an instance of the Tournament Dominating Set Problem as a Set Cover instance which would imply the reduction to the $(\frac{2}{3}, \frac{1}{3})$-Set Cover Problem. Let $G = (V, E)$ be the given tournament where $V = [n]$. Denote by $\Gamma(i)$ the set of vertices dominated by $i \in V$. We create a collection $\mathcal{S} = \{S_1, \ldots, S_n\}$ over the universe $U = V$, where $S_i = \Gamma(i) \cup \{i\}$. Since our construction of the $(\frac{2}{3}, \frac{1}{3})$-Set cover instance is restricted to k being power of 2 we add a sufficient number of dummy sets to $\mathcal{S}$ each containing a distinct element. We now construct the $(\frac{2}{3}, \frac{1}{3})$-Set Cover Instance $\langle S', 3k' \rangle$ as in Theorem 7, where k' is a power of 2 and $k \leq k' < 2k$. Instance $\langle G, k \rangle$ of the Tournament Dominating Set Problem is a YES instance if and only if $\langle \mathcal{S}', 3k' \rangle$ is a YES instance of the $(\frac{2}{3}, \frac{1}{3})$-Set Cover Problem. Since the number of sets in $\mathcal{S}'$ is at most $(n + k) + 2k' = \Theta(n)$ and the number of elements (positive and negative) is at most $n + k + 2(k' - 1) = \Theta(n)$, it follows that the $(\frac{2}{3}, \frac{1}{3})$-Set Cover Problem restricted to solutions of size $O(\log m)$ or $O(\log n)$ is LOGSNP-hard. ■

The (α, β)–Set Cover problem can be formulated as an Integer Linear Program in a straightforward manner. When there exists an optimal solution of size $\Omega(\log m)$, standard randomized rounding of the fractional solution can be used to derive from it an (α', β')-cover where $\alpha' \geq (1 - \epsilon)\alpha$ and $\beta' \leq (1 + \epsilon)\beta$ for some small ϵ.

6.1. LP based approach

We describe a natural LP relaxation to the (α, β)-Set Cover Problem. For each $S_i \in \mathcal{S}$, we introduce a binary variable x_i that is 1 if S_i is chosen in the cover, and 0 otherwise. Let $|\mathcal{S}| = n$ and $|U| = m$.

$$\text{minimize} \quad \sum_i x_i$$

$$\text{subject to} \quad \sum_{i:a \in S_i} x_i \geq \alpha \sum_i x_i \qquad \forall a \in U' \tag{5a}$$

$$\sum_{i:b \in S_i} x_i \leq \beta \sum_i x_i \qquad \forall b \in U'' \tag{5b}$$

$$\sum_{i:a \in S_i} x_i \geq 1 \qquad \forall a \in U' \tag{5c}$$

$$x_i \in \{0, 1\} \qquad \forall i. \tag{5d}$$

The LP relaxation is obtained by substituting the integrality constraint (5d) with $0 \le x_i \le 1$ for all i. Theorem 8 shows that we do not know how to find small solutions in polynomial time. Hence, we would like to ensure that the solution returned by the linear program is of size $\Omega(\log m)$. This can be done by adding the condition

$$\sum_i x_i \ge k = \Omega(\log m). \tag{5c$'$}$$

Since α is a constant that does not depend on m, condition (5c′) subsumes (5c).

A natural idea for rounding an optimal fractional solution is to view the fraction x_i as the probability that set $S_i \in \mathcal{S}$ is chosen. Let $\text{OPT}_f = \sum_i x_i$ be the size of the optimal fractional solution, and let $\mathcal{C}$ be the collection of sets we choose after the randomized rounding. We wish to show that $|\mathcal{C}|$ is very close to OPT_f and that conditions (5a) and (5b) are roughly satisfied. More specifically, we will show (*w.h.p.*) that for some $0 < \delta < 1$,

$$||\mathcal{C}| - \text{OPT}_f| \le \delta \text{OPT}_f, \tag{6a}$$

$$\forall a \in U', \quad |\mathcal{C}_a| \ge \frac{(1-\delta)}{(1+\delta)}\alpha|\mathcal{C}|, \text{ and} \tag{6b}$$

$$\forall b \in U'', \quad |\mathcal{C}_b| \le \frac{(1+\delta)}{(1-\delta)}\beta|\mathcal{C}|, \tag{6c}$$

where $\mathcal{C}_a$ and $\mathcal{C}_b$ are the collections of chosen sets that cover $a \in U'$ and $b \in U''$ respectively. Note that condition (5c′) ensures $\text{OPT}_f = k$. We will determine the exact value k in the course of the analysis.

Analysis. The expected size of $\mathcal{C}$ is,

$$E[|\mathcal{C}|] = \sum_i \Pr[S_i \text{ is picked}] = \sum_i x_i = \text{OPT}_f.$$

Let $\phi = \{||\mathcal{C}| - \text{OPT}_f| \ge \delta\text{OPT}_f\}$ be the event that $|\mathcal{C}|$ exceeds our desired bound (6a); by the Chernoff-Hoeffding bound,

$$\Pr[\phi] \le \Pr\left[|\mathcal{C}| \ge (1+\delta)\text{OPT}_f] + \Pr[|\mathcal{C}| < (1-\delta)\text{OPT}_f\right] = e^{-\delta^2 k/3} + e^{-\delta^2 k/2} \tag{7}$$

Next, in order to compute the probabilities of an element $a \in U'$ being covered by at least an α fraction of the chosen sets and an element $b \in U''$ being covered by at most $\beta|\mathcal{C}|$ sets, we will bound the probabilities of the bad events:

$$\phi_a = \{|\mathcal{C}_a| < (1-\delta)\alpha\text{OPT}_f\}, \text{ and } \phi_b = \{|\mathcal{C}_b| > (1+\delta)\beta\text{OPT}_f\},$$

for all a and b. Note that if ϕ_a does not occur for some a and if ϕ does not occur, then

$$|\mathcal{C}_a| \ge (1-\delta)\alpha\text{OPT}_f \ge \frac{(1-\delta)}{(1+\delta)}\alpha|\mathcal{C}|$$

and the bound (6b) will hold for a.

It is clear that $E[\mathcal{C}_a] \ge \alpha\text{OPT}_f$ and $E[\mathcal{C}_b] \le \beta\text{OPT}_f$. Now, for any $a \in U'$,

$$\Pr[\phi_a] = \Pr[|\mathcal{C}_a| < (1-\delta)E[|\mathcal{C}_a|]] \le e^{-\delta^2 E[|\mathcal{C}_a|]/2} \le e^{-\delta^2 \alpha k/2}. \tag{8}$$

Let $\delta' = \frac{(1+\delta)\beta\text{OPT}_f}{E[|\mathcal{C}_b|]} - 1$. For any $b \in U''$,

$$\Pr[\phi_b] = \Pr[|\mathcal{C}_b| > (1+\delta')E[|\mathcal{C}_b|]] \le \begin{cases} \exp\left(\dfrac{-(\delta')^2 E[|\mathcal{C}_a|]}{(2+\delta')}\right) \le e^{-\delta\beta k/3} & \text{if } \delta' \ge 1 \\ \exp\left(\dfrac{-(\delta')^2 E[|\mathcal{C}_a|]}{3}\right) \le e^{-\delta^2\beta k/3} & \text{otherwise} \end{cases}$$

where the second case follows from the fact that $(\delta')^2 E[|\mathcal{C}_b|] \geq \delta^2 \beta \text{OPT}_f$. Since $\delta < 1$,

$$\Pr[\phi_b] \leq e^{-\delta^2 \beta k/3}. \tag{9}$$

Using the Union bound, inequalities (7), (8), and (9) show that the probability of *some* bad event occurring is at most

$$m(e^{-\delta^2 \beta k/3} + e^{-\delta^2 \alpha k/2}) + e^{-\delta^2 k/3} + e^{-\delta^2 k/2}.$$

Finally, setting $k = \frac{6 \log m}{\delta^2 \alpha \beta}$ we have that (6a), (6b), and (6c) hold *w.h.p.* as desired.

Multiset solutions. One caveat of the linear program is that no feasible solution of size $k = \Omega(\log m)$ may exist (even when there exists a small solution). We can overcome this problem if we relax constraint (5d) further so that for all i, x_i can be any non-negative real number. Now if a small solution (of size $O(\log m)$) exists, the linear program can always scale this solution to satisfy condition (5c$'$). Again, this condition will be satisfied tightly, so we have that for all i, $x_i \leq k$. Hence, in order to perform the randomized rounding as above, we can create k new variables x_{i_j}, $1 \leq j \leq k$, for each x_i such that,

$$x_{i_j} = x_i / k, \quad \forall j.$$

The analysis above now holds; however, the resulting solution $\mathcal{C}$ will be a multiset. The number of times S_i is added to $\mathcal{C}$ is the number of coins whose outcome is 1 out of the k coins that are flipped with biases $x_{i_1}, \ldots, x_{i_k}$. We can think of the multiplicity of S_i as the weight of the set S_i in the solution.

6.2. Heuristics

The LP approach given above might be impractical when the number of sets induced by candidate substrings is large. In this section, we briefly outline two natural heuristics that can be used to find generalized (left) tag sets for a given node which we assume, *w.l.o.g.*, to be the root r of the phylogeny P.

The first approach is to find all substrings that appear in at least an α fraction of the sequences in the left subtree of r and at most a β fraction of the sequences in the right subtree of r. This can be done in linear time by slightly modifying the algorithms given in Angelov et al. (2004). We can then sample from these substrings to construct a generalized tag set T. The hope would be that each substring selected in T appears in a random subset of the input strings with the above constraints. Then, we expect that each sequence in the left subtree will contain at least an α fraction of the substrings in T and each sequence in the right subtree will contain at most a β fraction of the strings in T. Therefore, given a sufficiently large sample, the set T will be an approximate generalized tag set.

Alternatively, we can construct multiple instances of the discriminating substring problem by taking independent random samples of an α fraction of the sequences in the left subtree and a $(1 - \beta)$ fraction of the sequences in the right subtree of r. If we find discriminating tags in most of the instances, by taking one discriminating tag from each instance, we can construct an approximate generalized tag set. In fact, this is how we obtained the generalized tag sets in Section 7 for the Prokaryotes data set (Table 2). The running time of the method is also linear assuming that the number of instances is constant.

We note that neither approach is guaranteed to produce a solution, even if one exists, since a generalized tag set may include substrings that are present in less than α fraction of the left sequences or in more than β fraction of the right sequences. Consider the following example: Let $T = \{t_1, t_2, t_3\}$ be a $(\frac{2}{3}, \frac{1}{3})$-generalized tag set for $S' = \{s_1, s_2, s_3\}$ and $\bar{S}' = \{s_4, s_5\}$. The set T is such that s_1 and s_2 contain only t_1 and t_2, and s_3 contains t_1 and t_3. On the other hand, both s_4 and s_5 contain t_3 but not t_1 or t_2. Clearly, T is a generalized tag set even though t_3 is present in only one sequence of S' and in all sequences of $\bar{S}'$.

7. EXPERIMENTAL RESULTS

The existence of tags in small (relative to the full genomes) homologous data sets was demonstrated on the CFTR data set (Thomas et al., 2003) and the RDP-II data set (Maidak et al., 2001) in Angelov et al. (2004). However, we are interested in finding tags in whole genomes. Further, the existence of tags in

TABLE 2. LEFT AND RIGHT ($(\frac{2}{3}, \frac{1}{3})$)-GENERALIZED TAG SETS FOR THE ROOT OF THE PROKARYOTES TREE SHOWN IN WOLF ET AL. (2002)

Left tag set	*Right tag set*
CCGGGATTTGAACC	CCAACTGAGCTA
GTTCAAATCCCGGC	GTACGAGAGGAC
GGGATTTGAACCCG	TGCTTCTAAGCC

Tags in the left (resp. right) tag set have length 14 (resp. 12). *Left tag set*: Nine genomes in the left clade contain all 3 left tags and 2 genomes contain 2 tags, while 3 genomes in the right clade contain 1 tag from the set and the remaining 43 contain no left tags. *Right tag set*: 21 genomes in the right clade contain all right tags and 25 genomes contain 2 of the tags, while 5 genomes in the left clade contain 1 tag and the remaining 6 genomes contain no tags.

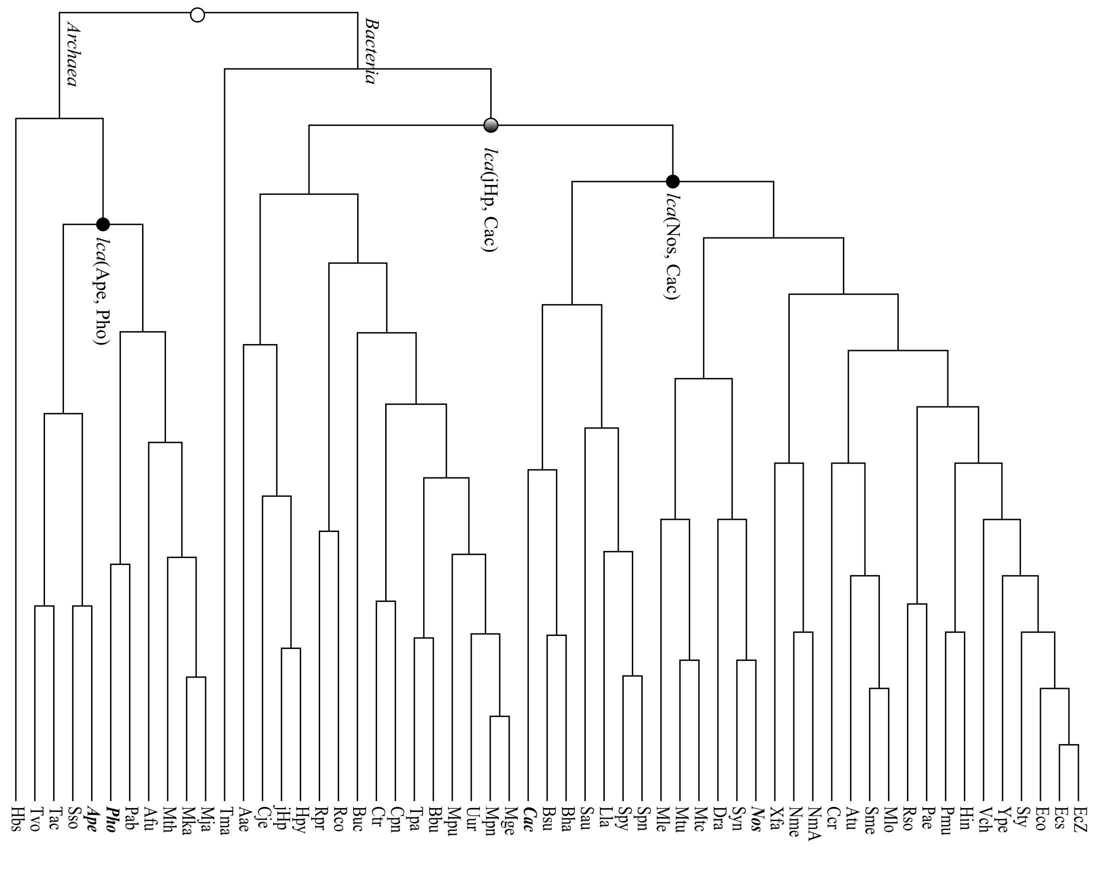

FIG. 2. Phylogeny of prokaryotes. Here edges do not represent actual distance. (For a detailed tree see Wolf et al. [2002].)

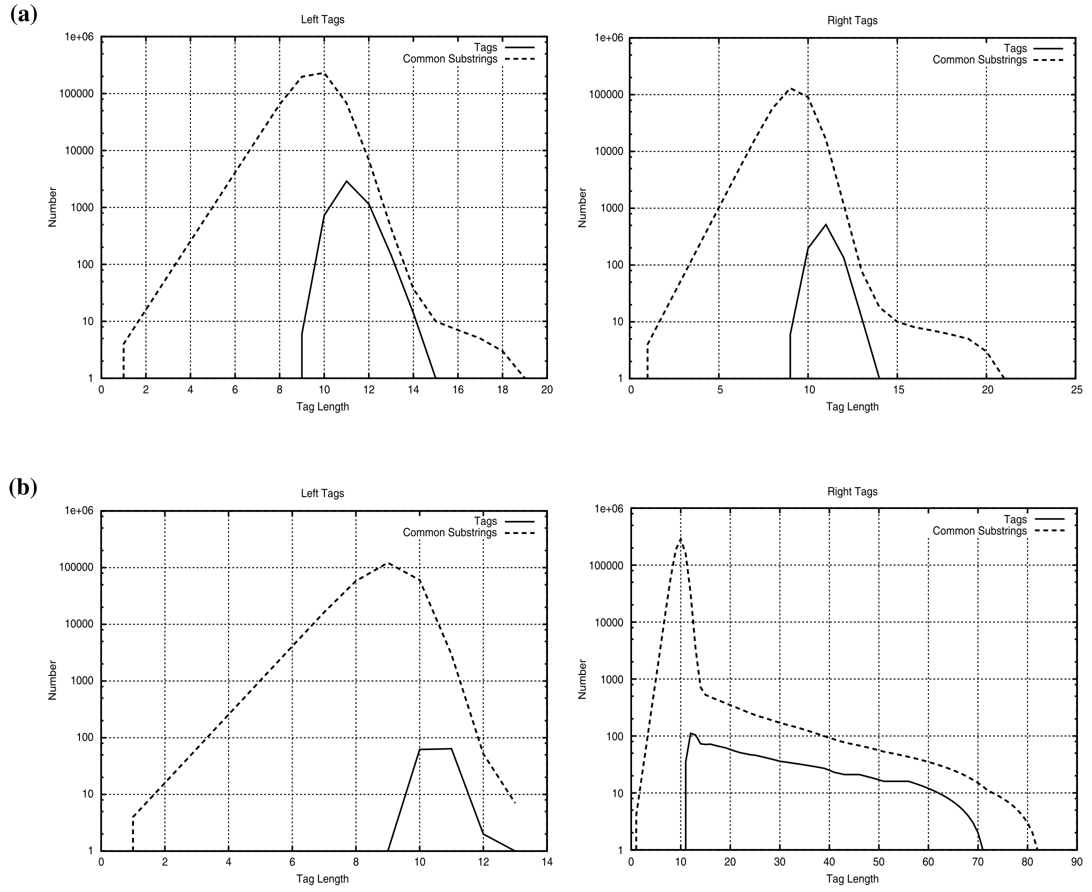

FIG. 3. Length (log-scale) distribution of tags and common substrings for two nodes of the prokaryotes phylogeny of Wolf et al. (2002). The left (right) panel displays discriminating tags present in the left (right) subtree of the corresponding node. **(a)** Tags and common substrings for the lowest common ancestor of Ape (*Aeropyrum pernix*) and Pho (*Pyrococcus horikoshii*). **(b)** Tags and common substrings for the lowest common ancestor of Nos (*Nostoc sp. PCC 7120*) and Cac (*Clostridium acetobutylicum*).

subsequences of a given set of strings does not necessarily imply the existence of tags in the strings. Here the existence of tags in data sets containing whole genomes is confirmed on the prokaryotes phylogeny obtained from Wolf et al. (2002).[2] The genomes represented in the data set span a broad evolutionary distance, at the level of one of the three recognized domains of life. But these genomes are also some of the smallest (1000-fold smaller than the human genome), allowing less sampling space for tags. Thus, they represent cases on the hard extremes of potential applications. We find left and right tags for all nodes of the phylogeny except for the root and the lowest common ancestor of Cac (*Clostridium acetobutylicum*) and jHp (*Helicobacter pylori*), where for the latter node only left tags are found. Relaxing the definition of a tag set as in Section 2, we show two $(\frac{2}{3}, \frac{1}{3})$-generalized tag sets for the root as examples. The prokaryotes phylogeny consists of 57 genomes where the average genome length is roughly 2.75 Mbp and the total length is about 157 Mbp. There are 11 sequences in the root's left subtree (Archaea) and 43 sequences in its right subtree (Bacteria). For convenience, Figure 2 illustrates the structure of the phylogeny (see Fig. 1 in Wolf et al. [2002]). We implemented the sublinear space algorithm described in Section 4.2 to find all tags for every node in the tree if they exist. Figure 3a shows both left and right tags for the lowest common ancestor of Ape (*Aeropyrum pernix*) and Pho (*Pyrococcus horikoshii*), and Figure 3b shows the

[2]Our experimental results can be found at: *www.cis.upenn.edu/∼angelov/phylogeny*.

tags for lowest common ancestor of Nos (*Nostoc sp. PCC 7120*) and Cac (*Clostridium acetobutylicum*). As mentioned earlier, we did not find tags for the root of the phylogeny. Hence we generated two $(\frac{2}{3}, \frac{1}{3})$-generalized tag sets for the root using a heuristic approach described in Section 6.2. Table 2 displays those two sets. We also enumerated the common substrings for this phylogeny as shown in Figure 3. As expected, longer common substrings are also discriminating tags; i.e., the longer the shared substrings, the more likely they are shared by evolutionary descent (what we call *type–I* tags in the analysis below). The experimental data suggests that, at least for this range of diversity, our approach will be successful at recovering informative substrings.

8. DISCUSSION

The data-driven approach to choosing discriminating oligonucleotide sequences appears to be novel. In this paper we have described how such sequences can be chosen given a "complete" data set consisting of a phylogeny where all the input sequences are present at the leaves. In this situation when our algorithms produce tags we can use them for high-throughput identification of an unlabeled sequence which is known to be one of the sequences in the input. Each tag found (at any node in the phylogeny) identifies an exactly conserved sequence shared by a clade. Such conserved segments can be used as seeds (in a BLAST-like fashion) to identify longer segments with high-similarity multiple alignments. When our algorithm fails to find tags, or even sufficiently long, shared sequences this is also informative. We learn that there is no strong conservation of segments within the clade. A natural extension of the problem considered here is to the situation where our knowledge is less complete. For example, how can one generalize to the case when the phylogeny is not fully known? If we attempt to place a new sequence in the phylogeny using the tags to guide us, how good is the placement as a function of the position of the new sequence in the phylogeny *vis a vis* the sequences from which the tag set was built? These are some of the directions that we plan to explore.

ACKNOWLEDGMENTS

We would like to thank Sheng Guo for his help with the simulation experiments and the anonymous referees for valuable comments and suggestions. This work was supported in part by NIGMS (grant 1P20GM69121).

REFERENCES

Amann, R., and Ludwig, W. 2000. Ribosomal RNA-targeted nucleic acid probes for studies in microbial ecology. *FEMS Microbiol. Rev.* 24, 555–565.

Angelov, S., Harb, B., Kannan, S., et al. 2004. Genome identification and classification by short oligo arrays. *Lect. Notes Comput. Sci.* 3240, 400–411.

Angelov, S., Harb, B., Kannan, S., et al. 2006. Efficient enumeration of phylogenetically informative substrings. *Lect. Notes Comput. Sci.* 3909, 248–264.

Bejerano, G., Pheasant, M., Makunin, I., et al. 2004. Ultraconserved elements in the human genome. *Science* 304, 1321–1325.

Bejerano, G., Siepel, A., Kent, W., et al. 2005. Computational screening of conserved genomic DNA in search of functional noncoding elements. *Nat. Methods* 2, 535–545.

Brown, M.R., and Tarjan, R.E. 1979. A fast merging algorithm. *J. ACM* 26, 211–226.

Brown, M.R., and Tarjan, R.E. 1980. Design and analysis of data structures for representing sorted lists. *SIAM J. Comput.* 9, 594–614.

Chang, W.I., and Lawler, E.L. 1994. Sublinear approximate string matching and biological applications. *Algorithmica* 12, 327–344.

Gusfield, D. 1997. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, New York.

Harel, D., and Tarjan, R.E. 1984. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.* 13, 338–355.

Hui, L. 1992. Color set size problem with applications to string matching. *Lect. Notes Comput. Sci.* 644, 227–240.

Jukes, T.H., and Cantor, C. 1969. *Mammalian Protein Metabolism.* Academic Press, New York.

Maidak, B.L., Cole, J.R., Lilburn, T.G., et al. 2001. The RDP-II (ribosomal database project). *Nucleic Acids Res.* 29, 173–174.

Matveeva, O.V., Shabalina, S.A., Nemtsov, V.A., et al. 2003. Thermodynamic calculations and statistical correlations for oligo-probes design. *Nucleic Acids Res.* 31, 4211–4217.

McCreight, E.M. 1976. A space-economical suffix tree construction algorithm. *J. ACM* 23, 262–272.

Megiddo, N., and Vishkin, U. 1988. On finding a minimum dominating set in a tournament (note). *Theoret. Comput. Sci.* 61, 307–316.

Papadimitriou, C.H., and Yannakakis, M. 1996. On limited nondeterminism and the complexity of the V-C dimension. *J. Comput. Syst. Sci.* 53, 161–170.

Pugh, W. 1990. A skip list cookbook [Technical Report CS-TR-2286.1]. University of Maryland Institute for Advanced Computer Studies.

Schieber, B., and Vishkin, U. 1988. On finding lowest common ancestors: simplifications and parallelization. *SIAM J. Comput.* 17, 1253–1262.

Siepel, A., Bejerano, G., Pedersen, J., et al. 2005. Evolutionarily conserved elements in vertebrate, insect, worm, and yeast genomes. *Genome Res.* 15, 1034–1050.

Stanley, R.P. 1999. *Enumerative Combinatorics. Volume 2*. Cambridge University Press, Cambridge, UK.

Thomas, J., Touchman, J., Blakesley, R., et al. 2003. Comparative analyses of multi-species sequences from targeted genomic regions. *Nature* 424, 788–793.

Ukkonen, E. 1995. On-line construction of suffix trees. *Algorithmica* 14, 249–260.

Weiner, P. 1973. Linear pattern matching algorithms. *Proc. 14th IEEE Symp. Switching Automata Theory*, 1–11.

Wolf, Y.I., Rogozin, I.B., Grishin, N.V., et al. 2002. Genome trees and the tree of life. *Trends Genet.* 18, 472–479.

Address reprint requests to:
Dr. Stanislav Angelov
Department of Computer and Information Science
University of Pennsylvania
3330 Walnut St.
Philadelphia, PA 19104

E-mail: angelov@cis.upenn.edu