Efficient Algorithms for Subtrees Comparison of Phylogenetic Trees with Applications to Two Component Systems Sequence Classifications in Bacterial Genome

Yaw-Ling Lin[*] Ming-Tat Ko[†]

Abstract

A phylogenetic tree with n leaves is a (rooted binary) tree such that each leaf node is uniquely labelled from 1 to n. The problem of whether there is a subtree in T_1 and another subtree in T_2 consisting of the same set of leaves in both trees with size of k is called *k-agreement* problem. The paper shows that it can be solved in $O(n)$ time.

The *normalized cluster distance*, $d(A, B)$, of two sets is defined by $d(A, B) = \Delta(A, B)/(|A| + |B|)$, where $\Delta(A, B)$ denotes the symmetric set difference of two sets. We show that computing all pairs normalized cluster distances between all paired subtrees of two trees can be done in $O(n^2)$ time. Since the total size of the outputs will be $\Theta(n^2)$, the algorithm is thus computationally optimal.

A nearest subtree of a subset of leaves is such a subtree that has the smallest normalized cluster distance to these leaves. Here we show that finding nearest subtrees for a collection of pairwise disjointed subsets of leaves can be done in $O(n)$ time. Furthermore, we show some hardness results concerning comparing two sets of co-evolution genes.

Several applications of these algorithms in areas of bioinformatics is considered. Among them, we discuss the 2CS (Two component systems) functional analysis and classifications on bacterial genome.

Keywords: two-component systems, bacterial genome, algorithms, phylogenetic trees, normalized cluster distance.

1 Introduction

With the rapid expansion in genomic data, the age of large-scale biomolecular sequence analysis has arrived. An important line of research in post-genome

[*]Department of Computer Science and Information Management, Providence University, 200 Chung Chi Road, Shalu, Taichung, Taiwan 433. e-mail: `yllin@pu.edu.tw`

[†]Institute of Information Science Academia Sinica, Nankang 115 Taipei, Taiwan 115. e-mail: `mtko@iis.sinica.edu.tw`

analysis is to analyze the evolution and co-evolution genes clustering of genomic sequences.

Antibiotic is the standard treatment for bacterial infections, which remains one of the leading causes of morbidity and mortality of humans in the world. However, drug-resistant bacterial strains are common that significantly limited the effectiveness of antibiotics.

The two-component signal transduction pathway (2CS) are usually composed of a sensor kinase and a response regulator. Because of its important functional roles and ubiquitous nature in most bacterial and fungal species, 2CS have been considered as very good targets in drug development [2]. The identification of the function of these 2CS would greatly facilitate not only our understanding on the basic physiology and regulatory networks of bacteria but also designing a way to prevent from causing disease in humans.

In this paper, we present algorithmic results concerning the bioinformatic applications in functional analysis and classifications of 2CS on bacterial genome.

The rest of the paper is organized as follows. Section 2 discusses the biological applications in more depth. We show that computing all pairs normalized cluster distances between all paired subtrees of two trees can be done in $O(n^2)$ time in Section 3.1. Since the total size of the outputs will be $\Theta(n^2)$, the algorithm is thus computationally optimal. We show that finding nearest subtrees for a collection of pairwise disjointed subsets of leaves can be done in $O(n)$ time, and finding whether there is a subtree in T_1 and another subtree in T_2 consisting of the same set of leaves in both trees with size of k can be solved in $O(n)$ time in Section 3.3. Furthermore, we show some hardness results concerning comparing two sets of co-evolution genes in Section 4.

2 Applications to Bacterial 2CS Sequence Analysis

Bacterial infections remain one of the leading causes of morbidity and mortality of humans in the world. Antibiotic is the standard treatment but drug-resistant bacterial strains are common that significantly limited the effectiveness of antibiotics. It is not surprising to find that many companies have exerted tremendous effort to develop novel antibiotics. So far, less than 20 classes of antibiotics and target molecules in bacteria are known. To overcome the drug resistant problem, many underutilized drug targets are being reevaluated and novel targets are in urgent demands. Bacterial components that serve as a target of drug intervention can be divided arbitrarily into several categories. These include virulence factors, gene products essential for the growth during infection, enzymes unique in bacteria, bacterial membrane transporters, bacterial two-component signal transduction pathway, product of genes unique in virulent strains of the bacterial pathogen, and product of genes conserved through evolution. Among these potential drug targets, the bacterial virulence factors are the most obvious and likely to be the most effective targets for antibacterial drug intervention.

Although new lines of antibiotic are in urgent demands, there were only 27 antibiotics under development in 1998, most of them focus on modification of existing drugs. It is expected that only a handful of them will be approved by FDA in the near future. Diagnosis of bacterial disease is a rather time-consuming process even in modern clinical laboratories. Typically 3-4 days are required to make a diagnosis for acute bacterial infections and up to 4 weeks for a chronic infection such as tuberculosis. It can take an even longer time if drug susceptibility tests are included. Therefore, it is not unusual that physicians could choose the wrong type of antibiotics, which not only delay the timing of appropriate treatment but also can result in drug resistance. How to differentiate critical groups of pathogen and together their drug susceptibility pattern within hours has become an important research direction in clinical microbiology.

These virulence genes are therefore the prime targets for development of diagnostic tool and vaccine, and for antimicrobial drug intervention. In tradition, the bacterial virulence factors were identified through a series of microbiology and immunology studies. This process has been greatly facilitated recently by the completion of genome projects of many pathogenic bacteria.

Many virulence-associated genes can be readily identified through bioinformatic approaches. Nevertheless, the BLAST programs [1] commonly used in genome analysis have their limitation. On the basis of BLAST search, it is estimated that approximately 20% of genes found in genome programs are novel sequences. Therefore, how to develop a novel annotation tool to identify the possible functional roles of these genes becomes a very important task. In addition, since the functions of these novel sequences are yet to be identified, very little attention has been drawn on these sequences.

Rapid adaptation to environmental challenge is essential for bacterial survival. To orchestrate their adaptive responses to changes in their surroundings, bacteria mainly use so-called 'two-component regulatory systems' (2CS) [7]. These systems are usually composed of a sensor kinase, which is able to detect one or several environmental stimuli, and a response regulator, which is phosphorylated by the sensor kinase and which, in turn, activates the expression of genes necessary for the appropriate physiological response. Sensor kinases (or histidine kinases) usually possess two domains: an input domain, which monitors environmental stimuli, and a transmitter domain, which auto-phosphorylates following stimulus detection. A classical response regulator contains an amino-terminally located conserved receiver domain that is phosphorylated by the sensor kinase at a strictly conserved aspartate residue, leading to activation of the carboxy-terminal effector or output domain [9, 10].

Because of its important functional roles and ubiquitous nature in most bacterial and fungal species, 2CS have been considered as very good targets in drug development [2]. In addition, 2CS also meet the following criteria for drug development. Some of 2CS are critical for bacterial growth and coordinate pathogenesis, including some problematic infection (eg. Biofilm formation). The enzymatic activity of 2CS is assayable and homology is high at active site, which lend itself to drug screening. 2CS are not found in humans that provides selective basis over mammalian targets/processes. They are surface exposed and are previously unexploited targets. Finally, there are multiple sets of 2CS in a bacterial genome and hence with low expected resistance. Analysis of complete bacterial genome sequences have shown that the number of these systems varies considerably from one species to the next, from 0 in Mycoplasma spp., 38 in the cyanobacterium Synechocystis [8], and 63 in Pseudomonas aeruginosa [11]. The functional role of most of the 2CS, however, remains elusive. For examples, among the 63 2CS in P. aeruginosa, only 10 or so have been characterized [10].

The identification of the function of these 2CS would greatly facilitate not only our understanding on the basic physiology and regulatory networks of bacteria but also designing a way to prevent from causing disease in humans.

Traditionally transcription regulators are classified according to their helix-turn-helix DNA binding motif and are assigned into families such as LysR or LuxR. Most of the genes encoding the transcription regulator are located in the upstream of their target genes and are transcribed from a divergent promoter in a direction opposite from that of targeted genes. Sequence analysis of the transcription regulators indicates that they are most likely derived from duplication events from an ancestor and was later recruited and clustered together with the target genes. Therefore, in this type of gene cluster, the transcription regulator genes were evolved independently from their target genes. In contrast, the target genes regulated by transcription (response) regulator of a 2CS are generally scattered in the genome, whereas the gene encoding response regulator and the sensor are located within an operon.

It is therefore interesting to know whether the gene encoding regulatory protein and the gene encoding the sensor kinase in a 2CS were derived by duplication from an existing 2CS (the *co-evolution*) or they were evolved independently and were assembled by recombination event later.

To address this question, we will need to know the evolutionary distance of each regulatory protein encoding gene and each sensor-encoding gene of the 63 2CS. The two trees will then be integrated and those sensor and regulator genes exhibit distinct relationship in a 2CS will be selected and analyzed further. In this case, integration of the two trees not only is the key to reveal the secret of 2CS evolution, but also a challenging question in computational biology. One way of extracting the useful clustering information that might later lead to functional classifications of these 2CS from the regulator tree and the sensor tree is to incorporate the evolutionary information from both trees.

3 Subtrees Comparison of Phylogenetic Trees

Biologists use the information contained in the DNA sequences of a collection of organisms, or taxa, to infer the evolutionary relationships among those taxa. These evolutionary relationships are generally represented by a labelled binary tree, called a *phylogenetic tree*. Here a phylogenetic tree with n leaves is a (rooted binary) tree such that all the leaf nodes are uniquely labelled from 1 to n. Given two n-leaf phylogenetic trees, we wish to explore the subtrees relationships between subtrees of the two trees.

Consider n terminal nodes that represents n abstract objects constituting the same set of leaf nodes within two (topologically different) evolutionary / phylogenic trees, say T_1 and T_2. We call these two trees as *paired trees*. Note that the deletion of any edge separates the tree into two disconnected subtrees, whose leaf nodes are exactly two subsets of the leaf nodes, forming two partitions.

We consider the following combinatorial problems:

Definition 1 (k-agreement) *Given a positive integer k and two (topologically different) n-leave phylogenetic trees T_1 and T_1, the problem is to identify whether there is an edge e_1 in T_1 and e_2 in T_2 whose deletion form the same partition of the leaf nodes in both trees and one of the two partitions has exact size of k.*

Note that it is trivial to identify 1-agreement (as well as n-agreement) nodes from any given paired trees. On the other hand, not every parameter k leads to a feasible solution. The rationale behinds the problem is the following.

Assuming that the paired trees do process a k-agreement subset (with size k) of 2CS sequences, it follows that these k sequences have a very good chance of forming a reasonable candidate for the clustering group, and hopefully these genes would be functionally related to each others. Computationally, the k-agreement problem can trivially solved by a polynomial time algorithm.

There are exactly $(n-1)$ internal nodes for a n-leaf (binary) tree; thus the number of (rooted) subtrees is exactly $n-1$. It follows that arbitrarily picking two pairs of internal nodes, each from one of paired trees, constitutes totally $O(n^2)$ possible pairings. For each pairing, a linear time algorithm can be used to double check whether these two pairs form an agreement. The total time needed for the trivial algorithm will be $O(n^3)$. More efficient algorithms are attainable for this problem.

Besides the exact match solutions for the k-agreement problem, it will be useful for biologists to have some sort of *fuzzy* measurement of two clusters of genes. Many methodologies can be used for converting raw data into a meaningful distance table. A commonly used similarity measure, which forms a distance is the (normalized) cardinality of *symmetric set difference* [5].

Let $A, B \subset S = \{1, 2, \ldots, n\}$ be two clusters of genes set S. The *symmetric set difference*, $\Delta(A, B)$, is defined as the following:

$$\Delta(A, B) = |A \cup B \setminus A \cap B| = |A \setminus B| + |B \setminus A|.$$

Further, the *normalized cluster distance*, $d(A, B)$, is defined as the following:

$$d(A, B) = \frac{\Delta(A, B)}{|A| + |B|}$$

The normalized cluster distance between two sets is considered to be a rough measurement of the *degree of difference* between them. Note that $0 \leq d(A, B) \leq 1$; $d(A, B) = 0$ if $A = B$, and $d(A, B) = 1$ if $A \cap B = \emptyset$. In other words, the smaller value of $d(A, B)$ implies a greater *similarity* of A, B.

3.1 All Pairs Subtrees Comparison

Here we discuss the problem of all pairs subtree comparison. Given a rooted leaf-labelled binary T with v being a (internal or leaf) vertex, we use $L(v)$ to denote the set of all descendent leaves of v in T. That is, $L(v) = \{x \mid x \text{ a descendent of } v\}$. Note that $L(v) = \{v\}$ if v itself is a leaf. A naive, $O(n^3)$-time algorithm of computing all the normalized distances of all pairs of subtrees of given paired sensor trees is illustrated at Figure 1.

```
NAIVE-ALL-PAIR($T_1, T_2, n, k$)
Input: A sensor kinase tree $T_1$, a (response) regulator tree $T_2$, with $n$
    leaves.
Output: A list of $k$ pairs of co-subtree $(t_1, t_2)$'s where $t_1$ ($t_2$) is a subtree
    of $T_1$ ($T_2$). These $k$ co-subtrees possess the smallest normalized
    cluster distance.
Step 1: Let the $A = \{a_1, a_2, \ldots, a_{n-1}\}$ denote the $n-1$ subtrees of $T_1$
    defined by the $n-1$ internal nodes of $T_1$. Let $B = \{b_1, b_2, \ldots, b_{n-1}\}$
    denote the $n-1$ subtrees of $T_2$. Let output list $P \leftarrow \emptyset$.
Step 2: For each $(a, b) \in A \times B$, compute the normalized cluster distance
    $d(a, b) = |L(a) \cup L(b) \setminus L(a) \cap L(b)|/(|L(a)| + |L(b)|)$.
Step 3: Select the $k$ co-subtree with smallest normalized cluster distance
    among all $d(a, b)$'s. Output these $k$ pairs in non-decreasing order.
```

Figure 1: Computing the normalized cluster distance $d(t_1, t_2)$'s within a pair of co-evolution trees.

The goal here is to compute all paired distances within $O(n^2)$ time. Note that the total size of the outputs will be $\Theta(n^2)$. An $O(n^2)$ time algorithm is thus computationally optimal. The idea here is trying to find a recurrence formula such that the normalized cluster distance of a parent node can be computed from its children in constant time. Let u be a node of a phylogenetic tree T_1, and v, v_1, v_2 be 3 nodes of another phylogenetic tree T_2, where v is the parent of v_1 and v_2. Now the target is to compute $\Delta(u, v)$ from $\Delta(u, v_1)$ and $\Delta(u, v_2)$ in constant time. Note that, for easier description, we use $\Delta(u, v)$ as a short-hand notation of $\Delta(L(u), L(v))$.

Lemma 1 (constant time $\Delta(A, B)$ calculation) *Let A, B_1, B_2 be 3 sets with $B_1 \cap B_2 = \emptyset$. It follows that $\Delta(A, B_1 \cup B_2) = \Delta(A, B_1) + \Delta(A, B_2) - |A|$.*

Proof. It is easily verified that $\Delta(X, Y) = |X| + |Y| - 2|X \cap Y|$ for any two sets X, Y. Now we have

$$\begin{aligned}
\Delta(A, B_1 \cup B_2) &= |A| + |B_1 \cup B_2| - 2|A \cap (B_1 \cup B_2)| \\
&= |A| + |B_1| + |B_2| - 2|A \cap (B_1 \cup B_2)| \\
&= |A| + |B_1| + |B_2| - 2|A \cap B_1| - 2|A \cap B_2)|
\end{aligned}$$

```
ALL-PAIR($T_1, T_2$)
Input: Two phylogenetic trees $T_1$ and $T_2$ with leaves $\{1, 2, \ldots, n\}$.
Output: All pairs $\Delta(u, v)$'s for all $u \in T_1, v \in T_2$.

1 Compute $|L(u)|$'s, $|L(u)|$'s, level($u$)'s, level($v$)'s for all $u \in T_1, v \in T_2$.
2 for each $u \in T_1$ in increasing order of level($u$) do        $\triangleright$ bottom up.
3     for each $v \in T_2$ in increasing order of level($v$) do
4         if both $u, v$ are leaf nodes then        $\triangleright$ initial condition.
5             $\Delta(u, v) \leftarrow 0$ if $u = v$; otherwise, $\Delta(u, v) \leftarrow 2$
6         else if $u$ is a leaf then let $v$ be the parent of $v_1$ and $v_2$;
7             $\Delta(u, v) \leftarrow \Delta(u, v_1) + \Delta(u, v_2) - 1$ $(= |L(u)|)$
8         else let $u$ be the parent of $u_1$ and $u_2$;
9             $\Delta(u, v) \leftarrow \Delta(u_1, v) + \Delta(u_2, v) - |L(v)|$
```

Figure 2: Computing all pairs $\Delta(T_1, T_2)$'s in $O(n^2)$ time.

$$\begin{aligned} &= |A| + |B_1| - 2|A \cap B_1| + |A| + |B_2| - 2|A \cap B_2)| - |A| \\ &= \Delta(A, B_1) + \Delta(A, B_2) - |A| \end{aligned}$$

since $B_1 \cap B_2 = \emptyset$. □

Note that Lemma 1 implies that $\Delta(u, v)$ can be calculated from $\Delta(u, v_1)$ and $\Delta(u, v_2)$ in constant time when $|L(u)|$ is precomputed. Given a pair of phylogenetic trees T_1 and T_2, we can store at each node $u \in T_1, v \in T_2$ with its associated descendants size $|L(u)|$ and $|L(v)|$. Further, for each node $u \in T_1$ we can store an array consisting of $\Delta(u, \cdot)$'s so that whenever we need to decide the value $d(u, v)$, it can be computed as $d(u, v) = \Delta(u, v)/(|L(u)| + |L(v)|)$.

Theorem 1 *Computing all paired subtree distances can be done in $O(n^2)$ time.*

Proof. We propose an $O(n^2)$ time algorithm, ALL-PAIR(T_1, T_2), shown in Figure 2. The algorithm essentially builds up all $\Delta(\cdot, \cdot)$'s in a bottom up manner. The correctness of the algorithm is easily followed by Lemma 1 and the correctness of the computation ordering.

To ensure the correct computation ordering, we introduce the following notations. Given a phylogenetic tree T and a node $v \in T$, the *v-descendant subtree*, denoted by T_v, is the subtree induced of by all descendants of v in T; here we assume that v is a descendant of itself. The *level* of a node v in T, denoted by level(v), is the *height* of T_v. Thus, whenever we traverse nodes of a phylogenetic tree T in their *increasing* levels ordering, we ensure that the descendants of a node v have already been visited before v.

It is easily seen that Step 1 of All-Pair can be computed in $O(n)$ time by a bottom up computation. A commonly used post order nodes traversal of a tree suffices. Let v be the parent of v_1, v_2 in a tree T. It follows that $|L(v)| = |L(v_1)| + |L(v_2)|$ with the initial condition that $|L(v)| = 1$ when v is a leaf. Further, $\text{level}(v) = \max\{\text{level}(v_1), \text{level}(v_2)\} + 1$ with the initial condition that $\text{level}(v) = 1$ when v is a leaf.

Also, it is clear that the inner steps of Step 4 to Step 9 take constant time to compute each time. Since there are exactly $O(n^2)$ number of iterations for the double loop (Step 2 and Step 3), it follows that All-Pair(T_1, T_2) finishes in $O(n^2)$ time. $\square$

3.2 Confluent Subtrees

The *lowest common ancestor* (LCA) between two nodes u and v in a tree is the furthest node from the root node that exists on both paths from the root to u and v. Harel and Tarjan [6] have shown that any n-node tree can be preprocessed in $O(n)$ time such that subsequently LCA queries can be answered in constant time.

Let T be a phylogenetic tree with leaf nodes L. Given $S \subset L$, define set $\textsc{Lca}(S) = \{\textsc{Lca}(x, y) \mid x \neq y \in S\}$ as the collection of all (proper) lowest common ancestors defined over S. It is easily verified (by induction) that $|\textsc{Lca}(S)| = |S| - 1$ since T is a binary tree.

Definition 2 (confluent subtree) *Let T be a phylogenetic tree with leaf nodes L. Given $A \subset L$, the* confluent subtree *of A in T is a phylogenetic tree, denoted by $T_{\uparrow A}$, with leaf nodes A and internal nodes* $\textsc{lca}(A)$. *Further, $u \in \textsc{Lca}(A)$ is a parent of v in $T_{\uparrow A}$ if and only if u is the lowest ancestor of v in T comparing to any other node in* $\textsc{Lca}(A)$.

```
CONFLUENT$(T, A)$
Input: A phylogenetic trees $T$ with leaves $L = \{1, 2, \ldots, n\}$, $A \subset L$.
Output: The confluent subtree of $A$ in $T$, $T_{\uparrow A}$.

Preprocessing: Compute the tree ordering of $L$ on $T$, and perform the LCA
constant time queries preprocessing [6].
Notations: $p[\cdot, T'], left[\cdot, T'], right[\cdot, T']$: parent, left, right children links.

1 Let $A = \langle v_1, v_2, \ldots, v_k \rangle$ be nodes of $A$ in the tree ordering.
2 Create a dummy node $\lambda$ and let $level[T, \lambda] \leftarrow +\infty$ ; PUSH$(S, \lambda)$
3 for $i \leftarrow 1$ to $k-1$ do        $\triangleright$ visit each $v_i$'s.
4     $x \leftarrow$ LCA$(v_i, v_{i+1})$ ; $y \leftarrow v_i$
5     while $level[x, T] > level[top[S], T]$ do $y \leftarrow$ POP$(S)$
6     PUSH$(S, x)$ ; $p[v_{i+1}, T'] \leftarrow x$ ; $p[y, T'] \leftarrow x$ ; $p[x, T'] \leftarrow top[S]$
7     $left[x, T'] \leftarrow y$ ; $right[x, T'] \leftarrow v_{i+1}$ ; $right[top[S], T'] \leftarrow x$
8 $root[T'] \leftarrow right[\lambda, T']$ ; return $T'$ as $T_{\uparrow A}$;
```

Figure 3: Computing the confluent subtree.

Let T be a phylogenetic tree with leaf nodes L. A post-order (pre-order, or in-order) traversal of nodes of L within T defines a *tree ordering* of nodes on L.

Lemma 2 *Let T be an n-node phylogenetic tree with leaf nodes L. The following subsequent operation can be done efficiently after an $O(n)$ time preprocessing. Given a query set $A \subset L$, the confluent subtree $T_{\uparrow A}$ can be constructed in $O(|A|)$ time if A is given in sorted tree ordering; otherwise, $T_{\uparrow A}$ can be constructed in $O(|A| \log\log |A|)$ time.*

Proof. We propose an $O(|A|)$ time algorithm, CONFLUENT(T, A), shown in Figure 3. The algorithm requires $O(n)$ time preprocessing phase for building up the tree ordering of L on T, and perform the LCA constant time queries preprocessing [6]. Further, the input $A \subset L$ is assumed to be listed according to the tree ordering of L; otherwise, we can use the data structure of van Emde Boas [12] for sorting these finite ranged integers in $O(|A| \log\log |A|)$ time.

The correctness of the algorithm can be shown by an inductive argument. It is easily verified that CONFLUENT computes the correct confluent tree when $k = 2$. For $k \geq 3$, it suffices to consider the last vertex v_k inserting into a correctly computed confluent tree $T_{\uparrow(A \setminus v_k)}$. Let $i = k - 1$; the situation is

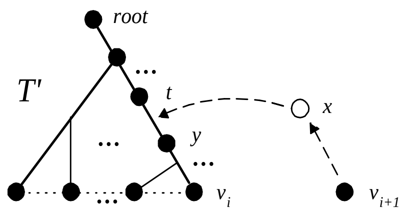

Figure 4: Inserting one node into a confluent tree.

illustrated in Figure 4. Note that the vertex $x = \textsc{Lca}(v_i, v_{i+1})$ must be a newly created node since the given phylogenetic tree T is a full binary tree; i.e., each internal node of T has exactly two children. Observe that the stack ordering of S essentially encodes the ordering of the *right fringe* of $T_{\uparrow(A\setminus v_k)}$. Since the inserted nodes are given in the tree ordering of L, v_{i+1} and x must be the newly admitted members of the right fringe of $T_{\uparrow A}$; that is, x shall be inserted into the stack S. It follows that Step 5 locates the correct position that x belongs in. Finally, Step 6 and Step 7 perform the corresponding operations setting up the tree links in T'.

In order to simplify the algorithm, a dummy node λ is set up so that the empty stack (S) situation, when x becomes the new root of T', is avoided. The final correct confluent tree is obtained after the correction of the final Step 8.

To justify that $\textsc{Confluent}(T, A)$ takes $O(|A|)$ time, here we show that Step 3 to Step 7 can be done in $O(|A|)$ time. The total number of operations of the algorithm is clearly bounded by $O(|A|)$ except for the while-loop body of Step 5. This can be shown by a simple amortized analysis [3]. In the following, we show that the amortized cost of the while-loop is a constant. Therefore, the overall time required by the loop is $O(|A|)$.

We define the *potential function* of S after the jth iteration of the for-loop (*i.e.* Steps 3 to 7) to be $\Phi(j)$, *i.e.* the numbers of elements within the stack S. Let us compute the amortized cost of the operations done by Step 5 in this jth iteration. Suppose that the number of elements in S is decreased by t_j in this period. Then the actual cost of the operations is $t_j + 1$. Observing that

$\Phi(j) = \Phi(j-1) - t_j + 1$, the change of the potential of S during the jth iteration is

$$\Phi(j) - \Phi(j-1) = 1 - t_j.$$

The amortized cost is therefore calculated as

$$\hat{t}_j = t_j + 1 + \Phi(j) - \Phi(j-1) = 2.$$

In other words, we deposit a credit (as a unit of the potential of S) whenever we push an element into S (Step 6). Later on, when the algorithm needs to pop elements of S at the while-loop, the cost can be charged to the pre-deposited credits. Since exactly $|A|$ credits would be deposited in the entire process, the while-loop spends at most overall $O(|A|)$ time. $\square$

3.3 Nearest Subtrees and Total agreement subtrees

Here we discuss the problem of Relative Nearest Subtrees.

Definition 3 (nearest subtree) *Let T be a phylogenetic tree with leaf nodes L. Given $A \subset L$, a node (internal or leaf) $v \in T$ (inducing T_v) is the* nearest subtree *of A in T if $(\forall x \in T)\,(d(L(v), A) \leq d(L(x), A))$.*

By utilizing Lemma 2, we can efficiently solve the nearest subtrees problem.

Theorem 2 *Let T be an n-leaf phylogenetic tree with leaf nodes L. Given a collection of pairwise disjointed subsets of L, $S = \{A_1, A_2, \ldots, A_j\}$, the nearest subtrees of all A_i's on T can be found in totally $O(n)$ time.*

Proof. We propose an $O(n)$ time algorithm, NEAREST(T, S), shown in Figure 5. For each A_i, the algorithm essentially computes all values of $d(A_i, T_v)$'s and find one with the smallest value. The correctness of the algorithm follows from the fact that any node $v \notin$ LCA(A_i) of T can not have the smallest cluster distance to A_i. The reason is that, if $v \notin$ LCA(A_i), one of two subtrees of T_v (with parent v) contains leaves that are completely disjointed with A_i. That is, the other subtree shall have a smaller distance. Thus, it suffices to consider only nodes of LCA(A_i) or CONFLUENT(T, A_i).

```
NEAREST$(T, S)$
Input: A phylogenetic trees $T$ with leaves $L$, $S = \{A_1, A_2, \ldots, A_j\}, A_i \subset L$.
Output: The nearest subtrees of $A_i$'s in $T$.

1 Compute the tree ordering of $L$ on $T$ and the LCA queries preprocessing.
2 Sort nodes of $A_i$'s in the tree ordering.
3 Compute the subtree sum $s[v, T]$'s for each $v \in T$ ; $Q \leftarrow \emptyset$
4 for $i \leftarrow 1$ to $|S|$ do        $\triangleright$ visit each $A_i$'s.
5       $T_i \leftarrow$ CONFLUENT$(T, A_i)$
6       Compute the subtree sum $s[v, T_i]$'s for each $v \in T_i$.
7       $\triangleright$ Find the node $v$ in $T_i$ with the maximum $d(A_i, T_v)$.
        $v_i \leftarrow \arg\max\{d(A_i, T_v) = 1 - 2s[v, T_i]/(s[v, T] + |A_i|) \mid v \in T_i\}$
8       Add $v_i$ into output queue $Q$
9 return $Q$, the nearest subtrees of $A_i$'s in $T$
```

Figure 5: Computing the nearest subtrees.

The algorithm requires $O(n)$ time preprocessing phase for building up the tree ordering of L on T, and perform the LCA constant time queries preprocessing [6]. Further, Step 2 sort each $A_i \subset L$ into the tree ordering of L. We describe briefly how the Step 2 can be done in total $O(n)$ time by using an indexed bucket array. Maintain an integer array $a[1..n]$ with size n. Initially, all $a[\cdot] \leftarrow 0$, storing zeros. Then, for each vertex v of A_i, we store in $a[id[v]] \leftarrow i$. Finally, for each $a[i]$, APPEND$(A_{a[i]}, i)$. It is clear that the process correctly sort the elements of A_i in $O(n)$ time. Step 3 computes the subtree sum of T, which can be easily done by a linear traversal of T in $O(n)$ time. That is, node x being the parent of y, z, let $s[x] \leftarrow s[y] + s[z]$; a leaf node x is initialized by $s[x] \leftarrow 1$.

Step 5 finds for each A_i its corresponding confluent subtree in T in $O(|A_i|)$ time by Lemma 2. Step 6 computes the subtree sum of T_i in $O(|A_i|)$ time. Step 7 finds for each set A_i its corresponding nearest subtree by examine each vertex of the confluent subtree T_i in $O(|A_i|)$ time with smallest cluster distance. Note that the cluster distances, $d(A_i, T_v)$, is exactly $\Delta(A_i, L(T_v))/|A_i|+|L(T_v)|$. However, recall that $\Delta(X, Y) = |X| + |Y| - 2|X \cap Y|$ for any two sets X, Y. It follows that $d(A_i, T_v) = 1 - 2|A_i \cap L(T_v)|/(|A_i| + |L(T_v)|)$, which is just $1-2s[v, T_i]/(s[v, T]+|A_i|)$. Thus $d(A_i, T_v)$ is just a constant time computation.

It follows that the total time spent in the for loop body (Step 5 to Step 8) is bounded by $\sum_{i=1}^{|S|} O(|A_i|) = O(n)$. □

With the required technical details at hand, we are ready to show that the k-agreement problem can be solved in $O(n)$ time.

Corollary 3 *Given a positive integer k and two n-leaf phylogenetic trees T_1 and T_1, finding whether there are subtrees of T_1 and subtrees of T_2 with same set of leaves having a exact size of k can be done in $O(n)$ time.*

Proof. As discussed in the proof of Theorem 2, we can calculate the subtree sums for nodes of T_1 in $O(n)$ time. Let $v_1, v_2, \ldots, v_m$ be nodes with subtree sum exactly k. Note that, if such nodes can not be found, the algorithm completes and reports null answer. Note that $S = \{L_{T_1}(v_1), L_{T_1}(v_2), \ldots, L_{T_1}(v_m)\}$ is a collection of pairwise disjointed subsets of leaves of T_2. By Theorem 2, we can find their nearest subtrees in T_2 in $O(n)$ time. Note that a node v_i with its paired nearest subtree constitutes a total agreed pair if their distance is zero. □

4 Co-evolution Genes Classfication

Gene duplication event is commonly occurred in bacteria that generate many gene families. These gene duplication events raise a very interesting question: does gene duplication tend to occur within a relative short distance on a bacterial genome? Despite this is a reasonable assumption; there has not been solid evidence to support this notion presumably due to lack of suitable testing systems. In bacteria, the number of member in a bacterial gene family is low and the repetitive sequences are too conserved, both are non-informative in genetic analysis. With more than 60 different 2CS in P. aeruginosa genome, it provides a great opportunity for us to test this interesting hypothesis. In this study, a dot-matrix plot will be created, with the X-axis being the physical distance, and Y-axis being the evolutionary distance, between two comparing 2CS.

It is possible that, instead of all 2CS whose sequences possess a correlation between their physical and evolutionary distances. Some subset of 2CS, presumably functionally related, could possess the correlation between their physical and evolutionary distances. Identifying these measurement-correlated groups could be a computational consumption problem. The following combinatorial optimization problem is considered:

Definition 4 (k-correlation) *Consider n nodes being associated with two different distance measurements M_1 and M_2; an $n \times n$ squared distance matrix represents each measurement. These two measurements are called* dual measures. *The difference of two squared matrix can be defined by $d(M_1, M_2) = |M_1 - M_2|$, or some other bio-meaningful functions. Note that a selection of k nodes, $A \subset S$, from the n-set S, produces an induced measurement from the given distance matrix; the induced matrix is denoted by M^A. The problem is, for a given parameter k, to identify the k-node, $A \subset S$, such that the difference $d(M_1^A, M_2^A)$ is minimized.*

Here we show that this combinatorial problem is intractable even when the distances measurements consist of only two different real numbers.

Theorem 4 *The decision version of k-correlation problem is NP-complete.*

Proof. The decision version of k-correlation problem is clearly in NP. Here we show that it is NP-hard by a reduction from the independent set (IndSet) [4] of simple graphs to this problem.

Given an instance of IndSet, $G = (V, E)$, we construct two distance matrices (M_1, M_2) such that (M_1, M_2) has k-node, $A \subset S$ with difference $d(M_1^A, M_2^A) = 0$ if and only if G has an independent set of size k.

The construction is straightforward. Let M_1 be the 0-1 adjacent matrix of G, and M_2 be an all zeros matrix. It is readily verified that $d(M_1^A, M_2^A) = 0$ if and only if G has an independent set of size k. $\square$

It will be interesting to know whether the problem can be reasonably approximated under some special constraints. Note that our k-agreement problem can be a good candidate of approximating the k-correlation problem if we first construct two phylogenetic trees from the two given measurements and then apply the k-agreement algorithm for finding probable k's.

5 Concluding Remarks

We thank Tsan-sheng Hsu for helpful conversations concerning the all pairs distance problem in Section 3.1.

In this paper, we present algorithmic results concerning the bioinformatic applications in functional analysis and classifications on bacterial genome.

We show that computing all pairs normalized cluster distances between all paired subtrees of two trees can be computationally optimally done in $O(n^2)$ time in Theorem 1. By using the concept of confluent subtree, we are able to show that finding nearest subtrees for a collection of pairwise disjointed subsets of leaves can be done in $O(n)$ time in Theorem 2. As a corollary, the k-agreement problem is solved in $O(n)$ time. Furthermore, we show that the k-correlation problem is NP-complete in Theorem 4.

Several interesting topics to be discussed in the future include identifying novel 2CS in other bacteria genomes as well as in eucaryotic genomes, clustering analysis of 2CS for functional prediction of uncharacterized genes, and 2CS coevolutionary analysis.

References

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tools. *J. Mol. Biol.*, 215:403–410, 1990.

[2] J. F. Barret and J. A. Hoch. Two-component signal transduction as a target for microbial anti-infective therapy. *Antimicrob. Agent. Chemo.*, 42:1529–36, 1998.

[3] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[4] M.R. Garey and D.S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[5] D. Gilbert, D. Westhead, N. Nagano, and J. Thornton. Motif–based searching in tops protein topology databases, 1999.

[6] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[7] J.A. Hoch and T.J. Silhavy. *Two-Component Signal Transduction*. ASM Press, 1995.

[8] T. et al. Mizuno. Compilation of all genes encoding bacterial two-component transducers in the genome of the cyanobacterium, synechocystis sp. strain PCC 6803. *DNA Res.*, 3:407–414, 1996.

[9] J.S. Parkinson and E.C. Kofoid. Communication modules in bacterial signalling proteins. *Annu. Rev. Genet.*, 26:71–112, 1992.

[10] A. Rodrigue, Y. Quentin, A. Lazdunski, V. Méjean, and M. Foglino. Two-component systems in pseudomonas aeruginosa: why so many? *Trends Microbiol.*, 8:498–504, 2000.

[11] C.K. et al. Stover. Complete genome sequence of pseudomonas aeruginosa PAO1, an opportunistic pathogen. *Nature*, 406:959–964, 2000.

[12] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6:80–82, 1977.