On Distances between Phylogenetic Trees

(Extended Abstract)

B. DasGupta* X. He† T. Jiang‡ M. Li§ J. Tromp¶ L. Zhang||

Abstract

Different phylogenetic trees for the same group of species are often produced either by procedures that use diverse optimality criteria [18] or from different genes [12] in the study of molecular evolution. Comparing these trees to find their *similarities* (*e.g.* agreement or consensus) and *dissimilarities*, *i.e.* *distance*, is thus an important issue in computational molecular biology. The *nearest neighbor interchange* (nni) distance [25, 24, 32, 4, 5, 3, 16, 17, 19, 29, 20, 21, 23] and the *subtree-transfer* distance [12, 13, 15] are two major distance metrics that have been proposed and extensively studied for different reasons. Despite their many appealing aspects such as simplicity and sensitivity to tree topologies, computing these distances has remained very challenging. This article studies the complexity and efficient approximation algorithms for computing the nni distance and a natural extension of the subtree-transfer distance, called the *linear-cost* subtree-transfer distance. The linear-cost subtree-transfer model is more logical than the (unit-cost) subtree-transfer model and in fact coincides with the nni model under certain conditions. The following results have been obtained as part of our project of building a comprehensive software package for computing distances between phylogenies.

1. Computing the nni distance is NP-complete. This solves a 25 year old open question appearing again and again in, for example, [25, 32, 4, 5, 3, 16, 17, 19, 20, 21, 23] under the complexity-theoretic assumption of $P \neq NP$. We also answer an open question [4] regarding the nni distance between unlabeled trees for which an erroneous proof appeared in [19]. We give an algorithm to compute the optimal nni sequence in time $O(n^2 \log n + n \cdot 2^{O(d)})$, where the nni distance is at most d. The algorithm allows us to implement practical programs when d is small. All above results also hold for linear-cost subtree-transfer.

2. Biological applications require us to extend the nni and linear-cost subtree-transfer models to *weighted* phylogenies, where edge weights indicate the length of evolution along each edge. We present a logarithmic ratio approximation algorithm for nni and a ratio 2 approximation algorithm for linear-cost subtree-transfer, on weighted trees.

1 Introduction

The evolution history of organisms is often conveniently represented as trees, called *phylogenetic trees* or simply *phylogenies.* Such a tree has uniquely labeled leaves and unlabeled interior nodes, can be *unrooted* or *rooted* if the evolutionary origin is known, and usually has internal nodes of degree 3. Over the past few decades, many different objective criteria and algorithms for reconstructing phylogenies have been developed, including (not exhaustively) parsimony [6, 9, 27], compatibility [22], distance [10, 26], and maximum likelihood [6, 7, 2]. The outcomes of these methods usually depend on the data and the amount of computational resources applied. As a result, in practice they often lead to different trees

*Department of Computer Science, Rutgers University, Camden, NJ 08102. Email: bhaskar@crab.rutgers.edu. Work done while the author was at University of Waterloo and was supported by an MRC/NSERC CGAT (Canadian Genome Analysis and Technology) grant.

†Supported in part by CGAT and NSF grant 9205982. Computer Science Department, SUNY-Buffalo, NY 14260. Email: xinhe@cs.buffalo.edu

‡Supported in part by NSERC Operating Grant OGP0046613 and CGAT. Department of Computer Science, McMaster University, Hamilton, Ont. L8S 4K1. Email: jiang@maccs.mcmaster.ca. Work done while visiting at University of Washington

§Supported by an MRC/NSERC CGAT grant. Department of Computer Science, Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. Email: mli@cs.cityu.edu.hk. On sabbatical leave from Department of Computer Science, University of Waterloo, Waterloo, Ont. N2L 3G1.

¶Supported by an NSERC International Fellowship and CGAT. CWI, P.O. Box 94079, 1090 GB Amsterdam. Email: tromp@cwi.nl

||BioInformatics Center, Institute of Systems Science, Heng Mui Keng Terrace, Singapore 119597. Email: lxzhang@iss.nus.sg. Work done while the author was at University of Waterloo.

on the same set of species [18]. It is thus of interest to compare phylogenies produced by different methods, or by the same method on different data, for similarity and discrepancy. Several metrics for measuring the distance between phylogenies have been proposed in the literature. Among these metrics, the best known is perhaps the *nearest neighbor interchange* (nni) distance introduced independently in [25] and [24].

An nni operation swaps two subtrees that are separated by an internal edge (u, v), as shown in Figure 1. The nni operation is said to *operate* or *perform* on this internal edge. The nni distance, $D_{nni}(T_1, T_2)$, between two trees T_1 and T_2 is defined as the minimum number of nni operations required to transform one tree into the other, as illustrated in Figure 2.

The complexity of computing the nni distance has been open for 25 years (since [25]). The problem is surprisingly subtle given the history of many erroneous results, disproved conjectures, and a faulty NP-completeness proof [32, 3, 16, 17, 19, 20, 23]. The question is open even for the simpler case where the trees are unlabeled. An erroneous NP-completeness proof for this case was published in [19].

The problem of computing distance between phylogenetic trees also arises in a different context. When the data is in the form of some molecular sequences of the organisms and the sequences have been subject to events such as *recombination* or *gene conversion* during the course of evolution, the evolutionary history of the sequences cannot be adequately described by a single tree. In an attempt to solve this problem, more general evolutionary models have been proposed including the network model [30] and a model using a list of phylogenetic trees [12, 13]. In the latter, every tree corresponds to a specific region of the sequences, and each tree can be obtained from the preceding tree on the list by transferring some subtrees from one place to another. Figure 3 shows a *subtree-transfer* operation and its corresponding recombination event. The parsimony model in [12, 13] requires the computation of the subtree-transfer distance between two trees, *i.e.* the minimum number of subtrees we need to move to transform one tree into the other. [15] shows that computing the subtree-transfer distance is NP-complete and gives a simple approximation algorithm with ratio 3.

It is relevant in practice to discriminate among subtree-transfer operations as they occur with different frequencies. For example, it is reasonable to assume that sequences that have only diverged recently give rise to more recombinations than sequences that diverged many generations ago [13, 14]. In this case, we can charge each subtree-transfer operation a cost equal to the distance (number of nodes passed) that the subtree has moved in the current tree. The *linear-cost* subtree-transfer distance, $D_{st}(T_1, T_2)$, between two trees T_1 and T_2 is then the minimum total cost required to transform T_1 into T_2 by subtree-transfers.

Surprisingly, although they are studied in parallel for very different reasons, we demonstrate here that the linear-cost subtree-transfer and nni are closely related. Observe that an nni move is just a restricted subtree-transfer where a subtree is only moved across a single edge. (In Figure 1, the first exchange can alternatively be seen as moving node v together with subtree C past node u towards subtree A, or vice-versa.) On the other hand, a subtree-transfer over a distance d can always be simulated by a series of d nni moves. Hence the linear-cost subtree transfer-distance is in fact *identical* to the nni distance.

A phylogeny may also have *weights* on its edges, where an edge weight (more popularly known as *branch length* in genetics) could represent the evolutionary distance along the edge. Many phylogeny reconstruction methods, including the distance and maximum likelihood methods, actually produce weighted phylogenies. Comparison of weighted phylogenies has recently been studied in [18]. The distance measure adopted is based on the difference in the partitions of the leaves induced by the edges in both trees, and has the drawback of being somewhat insensitive to the tree topologies [8]. Both the linear-cost subtree-transfer and nni models can be naturally extended to weighted phylogenies. An nni is simply charged a cost equal to the weight of the edge it operates on, while a moving subtree is charged for the weighted distance it travels. Intuitively these measures, especially the nni distance, are more sensitive to the tree topologies than the one in [18]. Note that for weighted phylogenies, the linear-cost subtree-transfer model is more general than the nni model in the sense that we can slide a subtree along an edge with subtree-transfers. Such an operation is not realizable with nni moves.

In this paper, we study the computational complexity and efficient approximation algorithms concerning the nni distance and linear-cost subtree-transfer distance on both unweighted and weighted phylogenies. We finally settle almost all questions regarding the nni distance. We show that computing the nni distance is NP-complete. The proof is quite nontrivial and it uses the lower and upper bounds [4, 29, 23] for sorting on a tree by nni operations in an essential way. The problem is also shown to be NP-complete for *unlabeled* trees, answering another open question in [4]. We will give an efficient $O(\log n)$ approximation algorithm for computing the nni distance on weighted phylogenies, where n is the number of leaves. A special case of the result for unweighted phylogenies was recently reported in [23]. We then give an exact algorithm that runs efficiently when the nni distance is sufficiently small. Such an algorithm is useful in practice as most trees compared are quite similar. The complexity of computing linear-cost subtree-transfer distance on weighted phylogenies is presently open, but here we present an efficient approximation algorithm with ratio 2 and show that computing linear-cost subtree-transfer distance is NP-complete for labeled trees provided the labels are not required to be unique.

Unless otherwise mentioned, all the trees in this paper are *degree*-3 trees with *unique labels on leaves.* An edge of a tree is *external* if it is incident on a leaf, otherwise it is *internal.* Finally, two weighted trees are *equal* iff there is an isomorphism between them preserving topology, edge weights (and leaf labels for labeled trees). Due to space limitations, many proofs are omitted from this extended abstract.

2 Computing the Nni Distance Is NP-complete

THEOREM 2.1. *Computing the nni distance (between two labeled trees) is NP-complete.*

The proof is by a reduction from Exact Cover by 3-Sets (X3C), which is known to be NP-complete [11], to our problem. Recall that, given an instance $S = \{s_1, \ldots, s_m\}$, where $m = 3q$, and $C_1, \ldots, C_n$, where $C_i = \{s_{i_1}, s_{i_2}, s_{i_3}\}$, the X3C problem is to find disjoint sets $C_{i_1}, \ldots, C_{i_q}$ such that $\cup_{j=1}^{q} C_{i_j} = S$. We will construct two trees T_1 and T_2 with unique leaf labels, such that transforming from T_1 into T_2 requires at most N (to be specified later) nni moves iff an exact cover of S exists.

Here is an outline of our reduction. We can perform sorting with nni moves and thus view nni as a special sorting problem. A sequence $x_1 \ldots x_k$ can be represented as a linear tree as in Figure 4. For convenience, such a linear tree will be simply called a sequence of length k. Sorting such a sequence means to transform it by nni operations to a linear tree whose leaves are in ascending order.

To construct the first tree T_1, for each $s_i \in S$, we create a sequence S_i of leaves that takes a "large" number of nni moves to sort. We will make sure that S_i and S_j are "very different" permutations for each pair $i \neq j$, in the sense that we cannot hope to have the sequence S_i sorted *for free* while sorting the sequence S_j by nni moves and vice versa. Then for each set $C_i = \{s_{i_1}, s_{i_2}, s_{i_3}\}$, we create three sequences with the same permutations as the sequences $S_{i_1}, S_{i_2}, S_{i_3}$, respectively, but with distinct labels. Such n groups of sequences for $C_1, \ldots, C_n$, each consisting of three sequences, will be placed "far away" from each other and from the m sequences $S_1, \ldots, S_m$ in tree T_1. Tree T_2 has the same structure as T_1 except that all sequences are *sorted.*

Here is the connection between exactly covering S and transforming T_1 into T_2 by nni moves. To transform T_1 into T_2, all we need is to sort the sequences defined above. If there is an exact cover $C_{i_1}, \ldots, C_{i_q}$ of S, we can partition the m sequences $S_1, \ldots, S_m$ into $\frac{m}{3} = q$ groups, according to the cover. For each C_j ($j = i_1, \ldots, i_q$) in the cover, we send the corresponding group of sequences $S_{j_1}, S_{j_2}, S_{j_3}$ to their counterparts, merge the three pairs of sequences with identical permutations, sort the three permutations, and then split the pairs and transport the three sorted versions of $S_{j_1}, S_{j_2}, S_{j_3}$ back to their original locations in the tree. Thus, instead of sorting six sequences separately, we do three merges, three sortings, three splits, and a round trip transportation of three sequences. Our construction will guarantee that the latter is significantly cheaper. If there is no exact cover of S, then either some sequence S_i will be sorted separately or we will have to send at least $q + 1$ groups of sequences back and forth. The construction guarantees that both cases will cost significantly more than the previous case.

We now give more details. Apparently many difficult questions have to be answered: How can we find these m sequences $S_1, \ldots, S_m$ that are *hard* to sort by nni moves? How do we make sure that sorting one such sequence will never help to sort others? How can we ensure that it is most beneficial to bring the sequences $S_{j_1}, S_{j_2}, S_{j_3}$ to their counterparts defined for C_j to get sorted, and not the other way?

We begin with the construction of the sequences $S_1, \ldots, S_m$. Recall that each such sequence is actually a linear tree, as in Figure 4. Intuitively, it would be a good idea to take a long and difficult-to-sort sequence and break it into m pieces of equal length. But this simple idea does not work for two reasons. First, such a sequence probably cannot be found in polynomial time. Second, even we find such a sequence, because the upper bound in [4, 23] and the lower bound in [29] (see [23]) do not match, these pieces may still help each other in sorting possibly by merging, sorting together, and then splitting. The following lemma states that there exists two sequences of constant size that are hard to sort and do not help each other in sorting. We will build our m sequences using these two sequences.

LEMMA 2.1. *For any positive constant $\epsilon > 0$, there exists infinitely many k for which there is a constant c and two sequences x and y of length k such that (i) each of them takes at least $(c-\epsilon)k\log k$ nni moves to sort, (ii) each of them takes at most $ck\log k$ nni moves to sort, and (iii) it takes at least $(1-\epsilon)c(2k)\log(2k)$ nni moves to sort both of them together, i.e. the sequence xy.*

Proof. Note that for any c, k, x, y, statements (ii) and (iii) imply statement (i). So it suffices to prove the existence of a constant c and an infinite number of k's that satisfy conditions (ii) and (iii).

From the results in [4, 23, 29], we know that for each k, there exists a sequence of k leaves such that sorting the sequence takes at most $k\log k + O(k)$ nni moves and at least $\frac{1}{4}k\log k - O(k)$ nni moves. Let us define c_k, for any k, as the maximum number of nni steps to sort any sequence of length k, divided by $k\log k$. Since $\frac{1}{4} - o(1) \le c_k \le 1 + o(1)$ there must be infinitely many k satisfy $c_{2k} \ge c_k - \frac{\epsilon}{2}$. Taking x and y to be the two halves of a hardest sequence of length $2k$, for large enough such k, and taking $c = c_k$, one can see that conditions (ii) and (iii) are satisfied. ∎

Let $\epsilon = 1/2$, k a sufficiently large integer satisfying Lemma 2.1 and c, x, y the corresponding constant and sequences. Next we use x and y, each of length k, to construct m long sequences $S_1, \ldots, S_m$. Choose m distinct binary sequences in $\{0,1\}^{\lceil \log m \rceil}$. Replace each letter 0 with the sequence x^{m^3} and each letter 1 with the sequence y^{m^3}. Give each occurrence of x and y unique labels. Insert in front of every x and y block a *delimiter* sequence of length k^2 with unique labels. This results in sequences $S_1, \ldots, S_m$, all with distinct labels. We can show that these sequences have the desired properties concerning sorting. The m sequences will have specific orientations in the tree; let's refer to one end as *head* and the other end as *tail*.

We are now ready to do the reduction. From sets $S = \{s_1, \ldots, s_m\}$, and $C_1, C_2, \ldots, C_n$, we construct the two trees T_1 and T_2 as follows. For each element s_i, T_1 has a sequence S_i as defined above. For each set $C_i = \{s_{i_1}, s_{i_2}, s_{i_3}\}$, we create three sequences $S_{i,i_1}, S_{i,i_2}, S_{i,i_3}$, with the same permutations as $S_{i_1}, S_{i_2}, S_{i_3}$, respectively, but with different and unique labels (we are not allowed to repeat labels).

Figure 5 outlines the structure of tree T_1. Here a thick solid line represents a sequence S_i or $S_{i,j}$ with the circled end as head; a dotted line represents a *toll* sequence of m^2 uniquely labeled leaves; a small black rectangle represents a **one-way circuit** as illustrated in Figure 6(i). The heads of m sequences at the left of Figure 5 are connected by two full binary trees connected root-to-root of depth $\log m + \log n$ to the n toll sequences, each leading to the *entrance* of a one-way circuit. The *exit* of each such one-way circuit is connected to the entrances of three one-way circuits leading finally to the three sequences corresponding to some set C_i.

A one-way circuit is designed for the purpose of giving free rides to subtrees moving first from 'a' to 'b' and then later from 'b' to 'a', while imposing a large extra cost for subtrees first moving from 'b' to 'a' and then from 'a' to 'b'. We will choose r so large (*i.e.* $r = m^4$) that it is not worthwhile to move any sequence $S_{i,j}$, corresponding to some C_i, to the left through the one-way circuits to sort and then move it back to its original location in T_1. This can be seen as follows. The counterpart of the one-way circuit in T_2 is as shown in Figure 6(ii).

In any optimal transformation of circuit (i) to (ii), the u's are paired up with the z's first and then the v's are paired with the u-z pairs. This requires u_r and v_1 to move up and out of the way. The pairing of the u's essentially provides a shortcut for u_r to reach z_r in half as many steps, and similarly for v_1.

In the following *sorting* a sequence S_i or $S_{i,j}$ means to have each of its x/y blocks sorted and then the whole sequence *flipped*. The tree T_2 has the same structure as T_1 except that

- all sequences S_i and $S_{i,j}$ are sorted.
- each circuit in Figure 6(i) is changed to (ii).

Let M be the cost for sorting a sequence $S_{i,j}$ optimally (M can be computed easily). The following lemma completes the reduction and thus the proof of Theorem 2.1.

LEMMA 2.2. (Proof omitted) *The set S has no exact cover iff $D_{nni}(T_1, T_2) \ge N + m^2/2$, where $N = q(\log m + \log n) + qm^2 + 28nm^4 - 28n + O(q) + 3nM + (k^2 + 6k)m^3 \log m + O(1)$.*

Next, we consider the hardness of computing the nni distance when both the trees have unlabeled leaves, solving an open problem mentioned in [4]. A flawed proof of Theorem 2.2 was published in [19].[1] Theo-

[1] In [19], the author reduced the Partition problem to nni by

rem 2.2 can be proved either using Theorem 2.1 or independently using a direct and much simpler reduction from the X3C problem.

THEOREM 2.2. (Proof omitted) *Computing the nni between two unlabeled trees is NP-complete.*

3 An Efficient Exact Algorithm for Small Nni Distance

In practice, the trees to be compared usually have small nni distances between them and it is of interest to devise efficient algorithms for computing the optimal nni sequence when the nni distance is small, say d. An $n^{O(d)}$ algorithm for this problem is trivial. With careful inspection, one can derive an algorithm that runs in $O(n^{O(1)} \cdot d^{O(d^2)})$ time, which can asymptotically be improved to $O(n^2 \log n + n \cdot d^{2d+o(d)})$. It turns out that by using the results in [29, 23], we could further improve the time to $O(n^2 \log n + n \cdot 2^{11d})$.

THEOREM 3.1. (Proof omitted) *Suppose that $D_{nni}(T_1, T_2) \leq d$. The optimal sequence of nni operations transforming T_1 into T_2 can be computed in $O(n^2 \log n + n \cdot 2^{11d})$ time.*

4 Approximation of Nni on Weighted Phylogenies

In this section we generalize the nni distance $D_{nni}(T_1, T_2)$ to the case when both T_1 and T_2 are weighted, the cost of an nni operation being the weight of the edge across which two subtrees are swapped. As mentioned in the introduction, many phylogeny reconstruction methods produce weighted phylogenies. Hence the weighted nni distance problem is also very important in computational molecular biology. NP-completeness of the (unweighted) nni distance problem (in Section 2) implies the NP-completeness of the weighted nni distance problem also.

We present a polynomial time algorithm with approximation ratio $O(\log n)$ for nni on weighted phylogenies, generalizing the logarithmic ratio approximation algorithm in [23]. The approximation for the weighted case is considerably more complicated. Note that nni operations can be performed only across internal edges. For feasibility of weighted nni transformation between two given weighted trees T_1 and T_2, we require in this section that the following conditions are satisfied: (1) for each leaf label a, the weight of the edge in T_1 incident on a is the same as the weight of the edge in T_2 incident on a, (2) the multisets of weights of internal edges of T_1 and T_2 are the same.

THEOREM 4.1. (Proof omitted) *Let T_1 and T_2 be two weighted phylogenies, each with n leaves. Then, $D_{nni}(T_1, T_2)$ can be approximated to within a factor of $6 + 6 \log n$ in $O(n^2 \log n)$ time.*

Note that the approximation ratio does not depend on the weights. Intuitively, the idea of the algorithm is as follows. We first identify "bad" components in the tree that need a lot of nni moves in transformation process. Then, for each bad component, we put things in correct order by first converting them into balanced shapes. But notice that we cannot afford to perform nni operations many times on heavy edges. Furthermore, not only the leaf nodes need to be moved to the right places, so do the weighted edges. The main difficulty of our algorithm is the careful coordination of the transformations so that at most $O(\log n)$ nni operations are performed on each heavy edge.

5 Linear-cost Subtree-Transfer Distance

In this section we investigate the linear-cost subtree-transfer model on weighted phylogenies. Recall that the linear-cost subtree-transfer distance is identical to the nni distance on unweighted phylogenies. Below we formalize the linear-cost subtree-transfer model.

Consider binary unrooted trees in which each edge e has a weight $w(e) \geq 0$. To ensure feasibility of transforming a tree into another, we require the total weight of all edges to equal one. A subtree-transfer is defined as follows. Select a subtree S of T at a given node u and select an edge $e \notin S$. Split the edge e into two edges e_1 and e_2 with weights $w(e_1)$ and $w(e_2)$ ($w(e_1), w(e_2) \geq 0$, $w(e_1) + w(e_2) = w(e)$), and move S to the common end-point of e_1 and e_2. Finally, merge the two remaining edges e' and e'' adjacent to u into one edge with weight $w(e') + w(e'')$. The cost of this subtree-transfer is the total weight of all the edges over which S is moved. Figure 7 gives an example. The subtree S is transferred to split the edge e_4 to e_6 and e_7 such that $w(e_6), w(e_7) \geq 0$ and $w(e_6) + w(e_7) = w(e_4)$; finally, the two edges e_1 and e_2 are merged to e_5 such that $w(e_5) = w(e_1) + w(e_2)$. The cost of transferring S is $w(e_2) + w(e_3) + w(e_6)$.

THEOREM 5.1. (Proof omitted) *Let T_1 and T_2 be two weighted trees with (not necessarily uniquely) labeled leaves. Then, computing $D_{st}(T_1, T_2)$ is NP-complete.*

THEOREM 5.2. *For any two weighted phylogenies T_1 and T_2, $D_{st}(T_1, T_2)$ can be approximated to within a factor of 2 in $O(n^2 \log n)$ time.*

In the rest of this section, we prove Theorem 5.2.

constructing a tree of i nodes for a number i.

We first define the notion of good edge pairs. Next, we devise an approximation algorithm for the case when T_1 and T_2 share no good edge pairs. Finally, we show how to apply the algorithm to the general case.

First, we introduce some notation. For any tree T, let $E(T)$ (resp. $V(T)$) denote the edge set (resp. node set) of T and $L(T)$ denote the set of leaf nodes of T. An external edge of T incident on a leaf node a is denoted by $e_T(a)$. Let $E_{int}(T)$ and $E_{ext}(T)$ denote the set of internal and external edges of T, respectively. For a subset $E' \subseteq E(T)$, define $w(E') = \sum_{e \in E'} w(e)$. Define $W_{int}(T) = w(E_{int}(T))$ and $W_{ext}(T) = w(E_{ext}(T))$. Next, we define the notion of good edge pairs:

DEFINITION 1. Let $e_1 \in E_{int}(T_1)$ and $e_2 \in E_{int}(T_2)$. Let T_1' and T_1'' be the two subtrees of T_1 partitioned by e_1. Let T_2' and T_2'' be the two subtrees of T_2 partitioned by e_2. e_1 and e_2 are called a *good pair* of T_1 and T_2 iff the following two conditions hold:

1. $L(T_1') = L(T_2')$ and $L(T_1'') = L(T_2'')$.

2. Either $w(E(T_1')) \leq w(E(T_2')) < w(E(T_1')) + w(e_1)$, or $w(E(T_2')) \leq w(E(T_1')) < w(E(T_2')) + w(e_2)$.

We say that nodes connected by 0-weight edges are equivalent and call the resulting equivalence classes *super-nodes*. Let $e_1, \ldots, e_k$ be all positive weight edges incident to a super-node o. With 0 cost, we can reconnect the edges $e_1, \ldots, e_k$ by any subtree, consisting of only 0 weight edges. In particular, the following observation will be useful in the description of our algorithm.

Observation. Let o be a super-node of T. Let $e_1, \ldots, e_k$ be all positive weight edges incident on o. Pick any e_i and e_j. We can assemble $\{e_1, \ldots, e_k\} - \{e_i, e_j\}$ into a single subtree S with 0 cost; and then transfer S along e_i by a distance $d \leq w(e_i)$. The effect of this operation is that the edges $e_1, \ldots, e_k$ are still incident on a super-node, and a portion of e_i of length d is *moved* into e_j. The total cost of this operation is d. We denote this operation by $move(e_i, d, e_j)$. This operation can be implemented in $O(k)$ time using the adjacency-list representation of the tree (where the weight of the edge is also stored in the adjacency list).

Figure 8 shows an example of this operation. In the figure, the thin lines denote 0 weight edges and heavy lines denote positive weight edges.

A tree T is called a *super-star* if all of its internal edges have 0 weight. In other words, all external edges of a super-star T are incident to a single super-node.

We are now ready to describe our algorithm. First, we consider the special case when T_1 and T_2 do not have any good edge pairs. Algorithm DST, as described below, approximates $D_{st}(T_1, T_2)$ to within a factor of 2. The algorithm transforms T_1 into a super-star T_1' (by moving the weight of internal edges into external edges). Similarly, the algorithm transforms T_2 into a super-star T_2'. The transformations are chosen to make T_1' coincide with T_2'. To transform T_1 to T_2, we first transform T_1 to $T_1'(= T_2')$ and then transform this to T_2. Let T_1' (resp. T_2') denote the tree during the transformation of T_1 (resp. T_2).

Algorithm DST:

Step 0. Initialize $T_1' = T_1$ and $T_2' = T_2$.

Step 1. While T_1' is not a super-star yet and there is an external edge $e_{T_1'}(a) = (a, u)$ in T_1' such that $w(e_{T_1'}(a)) < w(e_{T_2'}(a))$, do:

- Let e_1 be any positive weight internal edge of T_1' incident on the super-node containing u. Let $d = \min\{w(e_1), [w(e_{T_2'}(a)) - w(e_{T_1'}(a))]\}$.
- Perform the operation $move(e_1, d, e_{T_1'}(a))$ in T_1'. (Note: after this move operation, either the entire length of e_1 is moved into $e_{T_1'}(a)$ or $w(e_{T_1'}(a)) = w(e_{T_2'}(a))$).

(Note: after the loop terminates, either T_1' is a super-star or $w(e_{T_1'}(a)) \geq w(e_{T_2'}(a))$ for all leaf nodes a. Also we perform subtree-transfer only on internal edges of T_1).

Step 2. Similar to Step 1, with the roles of T_1' and T_2' swapped.

Step 3. We transform T_1' and T_2' into two super-stars such that $w(e_{T_1'}(a)) = w(e_{T_2'}(a))$ for all leaf nodes a. There are two possible cases as follows.

Case 3.1. $w(e_{T_1'}(a)) = w(e_{T_2'}(a))$ for all leaf nodes a. Perform the following loop to transform both T_1' and T_2' into super-stars. During the execution of the loop, we maintain the condition $w(e_{T_1'}(a)) = w(e_{T_2'}(a))$ for all leaf nodes a (this condition implies that T_1' is a super-star iff T_2' is a super-star).

Repeat

Pick any edge $e_{T_1'}(a) = (a, u_1)$ in T_1'. Suppose that the corresponding edge $e_{T_2'}(a)$ in T_2' is (a, u_2). Let e_1 be any positive weight internal edge of T_1' incident on the super-node containing u_1. Let e_2

be any positive weight internal edge of T_2' incident on the super-node containing u_2. Let $d = \min\{w(e_1), w(e_2)\}$. In T_1', perform the operation $move(e_1, d, e_{T_1'}(a))$. In T_2', perform the operation $move(e_2, d, e_{T_2'}(a))$. (After this, we have moved the entire length of either e_1 or e_2 into external edges.)

Until both T_1' and T_2' are super-stars.

(Note: during this step, we perform subtree-transfer only on internal edges of T_1 and T_2).

Case 3.2. There exists a leaf node a such that $w(e_{T_1'}(a)) \neq w(e_{T_2'}(a))$. This can happen only if both T_1' and T_2' are super-stars already. We need to make $w(e_{T_1'}(a)) = w(e_{T_2'}(a))$ for all leaf nodes a. This is done as follows. Partition $L(T_1')$ into three subsets A, B, and C as follows: A (resp. B, C) is the set of leaf nodes a (resp. b, c) such that $w(e_{T_1'}(a)) = w(e_{T_2'}(a))$ (resp. $w(e_{T_1'}(b)) < w(e_{T_2'}(b))$, $w(e_{T_1'}(c)) > w(e_{T_2'}(c))$).

Repeat

Pick any edge $e_{T_1'}(b)$ with $b \in B$ and $e_{T_1'}(c)$ with $c \in C$. Let $d = \min\{[w(e_{T_1'}(c)) - w(e_{T_2'}(c))], [w(e_{T_2'}(b)) - w(e_{T_1'}(b))]\}$. In T_1', perform $move(e_{T_1'}(c), d, e_{T_1'}(b))$. Then:

- If $d = w(e_{T_2'}(b)) - w(e_{T_1'}(b))$, remove b from B and put b into A.
- If $d = w(e_{T_1'}(c)) - w(e_{T_2'}(c))$, remove c from C and put c into A.
- If $d = w(e_{T_1'}(c)) - w(e_{T_2'}(c)) = w(e_{T_2'}(b)) - w(e_{T_1'}(b))$, remove b from B; remove c from C; put both b and c into A.

Until $B = C = \emptyset$.

Step 4. Now both T_1' and T_2' are super-stars and $w(e_{T_1'}(a)) = w(e_{T_1'}(a))$ for all leaf nodes a. We adjust the topology of the super-nodes of T_1' and T_2' so that T_1' and T_2' are identical.

The following lemma shows an upper bound on the performance ratio of algorithm DST.

LEMMA 5.1. (Proof omitted) *Assume that T_1 and T_2 do not share any good edge pairs. Then, algorithm DST approximates $D_{st}(T_1, T_2)$ to within a factor of 2 in $O(n^2)$ time.*

Next, we consider the general case. It is easy to find the set of all good edge pairs in $O(n^2 \log n)$ time using an algorithm similar to described in the proof of Lemma 3.1. Let K be the number of good edge pairs in T_1 and T_2. Our algorithm is by induction on K. If $K = 0$, algorithm DST works by Lemma 5.1. Suppose $K > 0$. Let $e_1 = (u_1, v_1) \in E(T_1)$ and $e_2 = (u_2, v_2) \in E(T_2)$ be a good pair. Let T_1' and T_1'' be the two subtrees of T_1 partitioned by e_1. Let T_2' and T_2'' be the two subtrees of T_2 partitioned by e_2, where $L(T_1') = L(T_2')$ and $L(T_1'') = L(T_2'')$.

Assume $w(E(T_1')) \leq w(E(T_2')) < w(E(T_1')) + w(e_1)$. (The other case can be handled in a similar way). Add a new edge (u_1, x) to T_1' and assign $w((u_1, x)) = w(E(T_2')) - w(E(T_1'))$. Add a new edge (x, v_1) to T_1'' and assign $w((x, v_1)) = w(e_1) - w((u_1, x))$. Add a new edge (u_2, x) to T_2' and assign $w((u_2, x)) = 0$. Add a new edge (x, v_2) to T_2'' and assign $w((x, v_2)) = w(e_2)$. (See Figure 9). Note that the weights of all new edges are non-negative.

Clearly, $L(T_1') = L(T_2')$ and $w(T_1') = w(T_2')$. We can normalize the weights of T_1' and T_2' such that their sum is 1. By induction hypothesis, we can transform T_1' to T_2' with cost at most $2D_{st}(T_1', T_2')$. Similarly, we can transform T_1'' to T_2'' with cost at most $2D_{st}(T_1'', T_2'')$. Combining the two transfer sequences, we can transform T_1 to T_2 with cost at most $2D_{st}(T_1, T_2)$. The complete algorithm takes $O(n^2 \log n)$ time. This completes the proof of Theorem 5.2.

6 Conclusion

These results have been obtained as a part of our larger project of building a comprehensive software package for comparing phylogenetic trees. It will include programs for computing nni, subtree-transfer, linear-cost subtree-transfer, edit, rotation, and contraction-decontraction distances. Part of these have already been implemented. Several open questions remain:

1. Can we approximate nni with a better ratio (on weighted or unweighted phylogenies)? It seems that to obtain a ratio better than $\log n$, we have to be able to prove superlinear lower bounds for sorting sequences on trees with nni moves.

2. Nni is similar to and slightly more powerful than rotation distance [4, 28]. Is rotation distance NP-complete? Can we approximate the rotation distance better than (the trivial ratio) 2? This question turns out to be subtler than it appears to be.

7 Acknowledgments

We thank J. Felsenstein and J. Hein for explaining to us the biological motivations for comparing weighted phylogenies and studying the linear-cost subtree-transfer distance, respectively, S. Yu for implementing the user interface, V. King and M. Waterman for explaining to us the nni problem, its history and relevant literatures and K. Zhang and T. Yokomori for useful discussions.

References

[1] M.A. Armstrong, *Groups and Symmetry*, Springer Verlag, New York Inc., 1988.

[2] D. Barry and J.A. Hartigan, Statistical analysis of hominoid molecular evolution, *Stat. Sci.*, 2(1987), 191-210.

[3] R. P. Boland, E. K. Brown and W. H. E. Day, Approximating minimum-length-sequence metrics: a cautionary note, *Math. Soc. Sci.*, 4(1983), 261–270.

[4] K. Culik II and D. Wood, A note on some tree similarity measures, *Inform. Proc. Let.*, 15(1982), 39–42.

[5] W. H. E. Day, Properties of the nearest neighbor interchange metric for trees of small size, *Journal of Theoretical Biology*, 101(1983), 275–288.

[6] A.W.F. Edwards and L.L. Cavalli-Sforza, The reconstruction of evolution, Ann. Hum. Genet., 27(1964), 105. (Also in *Heredity* 18, 553.)

[7] J. Felsenstein, Evolutionary trees for DNA sequences: a maximum likelihood approach. *J. Mol. Evol.*, 17(1981), 368-376.

[8] J. Felsenstein, personal communication, 1996.

[9] W.M. Fitch, Toward defining the course of evolution: minimum change for a specified tree topology, *Syst. Zool.*, 20(1971), 406-416.

[10] W.M. Fitch and E. Margoliash, Construction of phylogenetic trees, *Science*, 155(1967), 279-284.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[12] J. Hein, Reconstructing evolution of sequences subject to recombination using parsimony, *Math. Biosci.*, 98(1990), 185-200.

[13] J. Hein, A heuristic method to reconstruct the history of sequences subject to recombination, *J. Mol. Evol.*, 36(1993), 396–405.

[14] J. Hein, personal email communication, 1996.

[15] J. Hein, T. Jiang, L. Wang, and K. Zhang, On the complexity of comparing evolutionary trees, *Proc. 6th Combinatorial Pattern Matching Conf.*, Helsinki, 1995.

[16] J. P. Jarvis, J. K. Luedeman and D. R. Shier, Counterexamples in measuring the distance between binary trees, *Mathematical Social Sciences*, 4(1983), 271–274.

[17] J. P. Jarvis, J. K. Luedeman and D. R. Shier, Comments on computing the similarity of binary trees, *Journal of Theoretical Biology* 100(1983), 427–433.

[18] M. Kuhner and J. Felsenstein, A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Mol. Biol. Evol.* 11(3), 1994, 459–468.

[19] M. Křivánek, Computing the nearest neighbor interchange metric for unlabeled binary trees is NP-complete, *Journal of Classification* 3(1986), 55–60.

[20] V. King and T. Warnow, On Measuring the nni distance between two evolutionary trees, *DIMACS mini workshop on combinatorial structures in molecular biology*, Rutgers University, Nov 4, 1994.

[21] S. Khuller, Open Problems: 10, *SIGACT News*, 24:4(Dec., 1994), p.46.

[22] W.J. Le Quesne, The uniquely evolved character concept and its cladistic application, *Syst. Zool.*, 23(1974), 513-517.

[23] M. Li, J. Tromp, and L.X. Zhang, Some notes on the nearest neighbor interchange distance, *2nd COCOON*, Hong Kong, June 17-19, 1996.

[24] G. W. Moore, M. Goodman and J. Barnabas, An iterative approach from the standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets, *Journal of Theoretical Biology* 38(1973), 423–457.

[25] D. F. Robinson, Comparison of labeled trees with valency three, *Journal of Combinatorial Theory, Series B,* 11(1971), 105–119.

[26] N. Saitou and M. Nei, The neighbor-joining method: a new method for reconstructing phylogenetic trees, *Mol. Biol. Evol.*, 4(1987), 406-425.

[27] D. Sankoff, Minimal mutation trees of sequences, *SIAM J. Appl. Math.*, 28(1975) 35-42.

[28] D. Sleator, R. Tarjan, W. Thurston, Rotation distance, triangulations, and hyperbolic geometry, *J. Amer. Math. Soc.*, 1(1988), 647-681.

[29] D. Sleator, R. Tarjan, W. Thurston, Short encodings of evolving structures, *SIAM J. Discr. Math.*, 5(1992), 428–450.

[30] A. von Haseler and G.A. Churchill, Network models for sequence evolution, *J. Mol. Evol.*, 37(1993), 77-85.

[31] M. S. Waterman, *Introduction to computational biology: maps, sequences and genomes*, Chapman & Hall, 1995.

[32] M. S. Waterman and T. F. Smith, On the similarity of dendrograms, *Journal of Theoretical Biology*, 73(1978), 789–800.

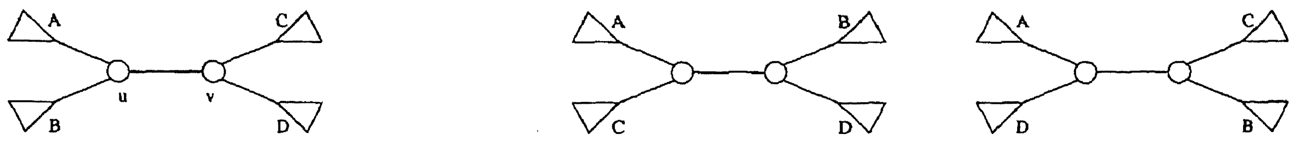

Figure 1: The two possible nni operations on an internal edge (u, v): exchange $B \leftrightarrow C$ or $B \leftrightarrow D$

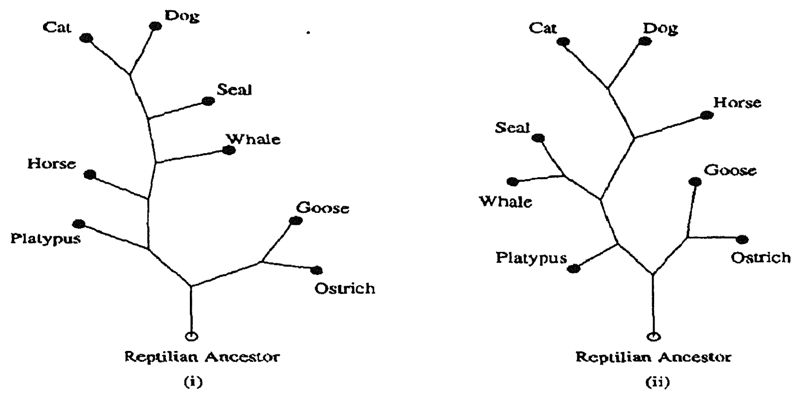

Figure 2: The nni distance between (i) and (ii) is 2.

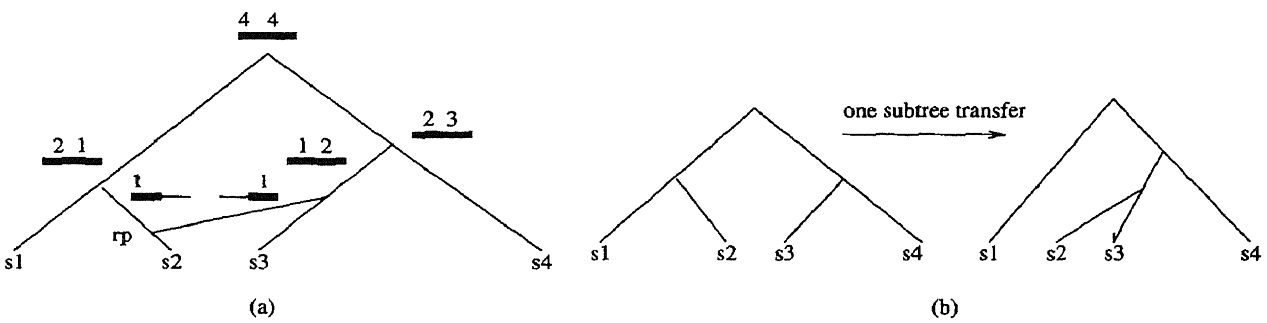

Figure 3: Recombination event at point rp in (a) corresponds to transferring subtree s2 in (b). The genetic material (thick lines), that is in one sequence after recombination, was in two sequences just before the recombination. The two sets of numbers (on the thick lines) correspond to the two evolutionary histories (as shown in (b)) of two parts of the sequences. For example, in the evolutionary tree for the second parts of the sequences (rightmost tree in (b)), a common ancestor of s2, s3, s4 is found going back in time; hence the second number of the thick line in second row is 3.

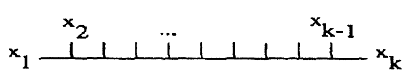

Figure 4: A linear tree with k leaves.

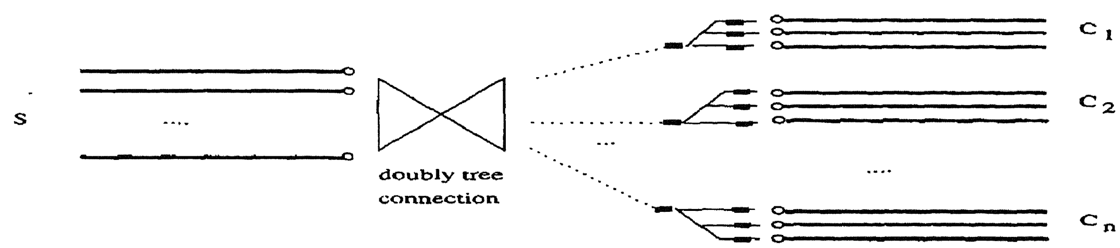

Figure 5: Structure of tree T_1

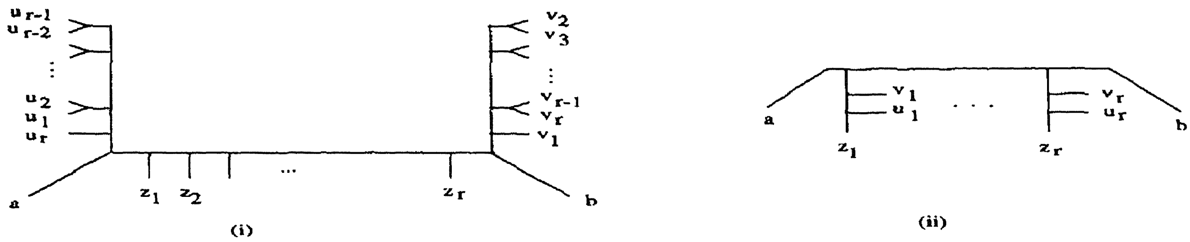

Figure 6: One-way circuit

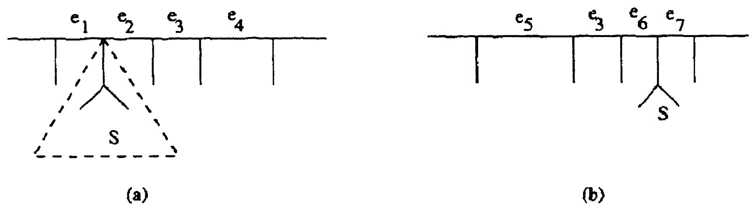

Figure 7: Subtree-transfer on weighted trees. Tree (b) is obtained from (a) with 1 subtree-transfer

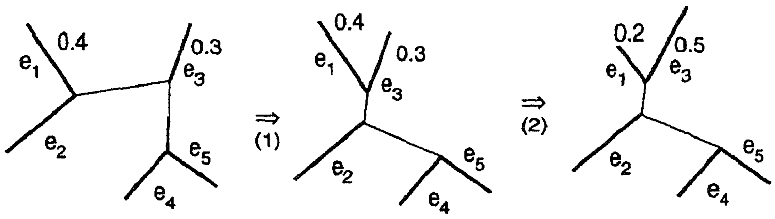

Figure 8: The operation $move(e_1, 0.2, e_3)$. (1) e_2, e_4, e_5 are assembled into a tree S; (2) S is moved along e_1 by a length of 0.2.

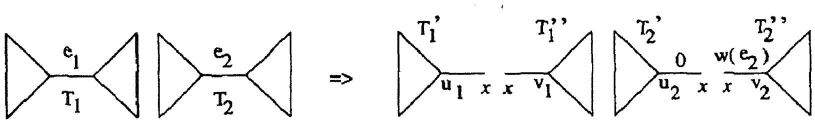

Figure 9: Cut each of T_1 and T_2 into two smaller trees.