On Computing the Nearest Neighbor Interchange Distance[*]

Bhaskar DasGupta[†]
Department of Computer Science
Rutgers University
Camden, NJ 08102, USA
E-mail: bhaskar@crab.rutgers.edu

Xin He[‡]
Department of Computer Science
SUNY-Buffalo
Buffalo, NY 14260, USA
E-mail: xinhe@cs.buffalo.edu

Tao Jiang[§]
Department of Computer Science
McMaster University
Hamilton, Ontario L8S 4K1, Canada
E-mail: jiang@maccs.mcmaster.ca

Ming Li[¶]
Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
Email: mli@math.uwaterloo.ca

John Tromp[||]
CWI
P.O. Box 94079
1090 GB Amsterdam, Netherlands
E-mail: tromp@cwi.nl

Louxin Zhang[**]
BioInformatics Center
Kent Ridge Digital Labs
Heng Mui Keng Terrace, Singapore 119597
Email: lxzhang@bic.nus.edu.sg

June 1, 1998

[*]The results reported here also form a subset of the results that appeared in *Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1997, pp. 427-436 [4]. The remaining results of the conference paper which do not appear in this paper will appear separately in a special issue in *Algorithmica* on computational biology.

[†]Supported by an CGAT (Canadian Genome Analysis and Technology) grant. Work done while the author was at University of Waterloo and McMaster University.

[‡]Supported in part by CGAT and NSF grant 9205982. Work done while the author was visiting at University of Waterloo.

[§]Supported in part by NSERC Operating Grant OGP0046613 and CGAT. Work done while the author was visiting at University of Washington.

[¶]Supported by NSERC Operating Grant OGP0046506 and CGAT.

[||]Supported by an NSERC International Fellowship and CGAT. Work done while the author was with University of Waterloo.

[**]Work done while the author was at University of Waterloo.

Abstract

In the practice of molecular evolution, different phylogenetic trees for the same group of species are often produced either by procedures that use diverse optimality criteria [24] or from different genes [15, 16, 17, 18, 14]. 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 [29, 28, 34, 3, 6, 2, 19, 20, 23, 33, 22, 21, 26] is a natural distance metric that has been extensively studied. Despite its many appealing aspects such as simplicity and sensitivity to tree topologies, computing this distance has remained very challenging, and many algorithmic and complexity issues about computing this distance have remained unresolved. This paper studies the complexity and efficient approximation algorithms for computing the nni distance and a natural extension of this distance on weighted phylogenies. The following results answer many open questions about the nni distance posed in the literature.

1. Computing the nni distance between two labeled trees is NP-complete. This solves a 25 year old open question appearing again and again in, for example, [29, 34, 3, 6, 2, 19, 20, 23, 22, 21, 26].
2. Computing the nni distance between two unlabeled trees is also NP-complete. This answers an open question in [3] for which an erroneous proof appeared in [23].
3. Biological applications motivate us to extend the nni distance to *weighted* phylogenies, where edge weights indicate the time-span of evolution along each edge. We present an $O(n^2)$ time approximation algorithm for computing the nni distance on weighted phylogenies with a performance ratio of $4 \log n + 4$, where n is the number of leaves in the phylogenies.

We also observe that the nni distance is in fact identical to the *linear-cost subtree-transfer* distance on unweighted phylogenies discussed in [4, 5]. Some consequences of this observation are also discussed.

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 internal nodes, is either *unrooted* or *rooted* (if the evolutionary origin is known), and usually all of whose internal nodes have degree 3. Over the past few decades, many different objective criteria and algorithms for reconstructing phylogenies have been developed, including (not exhaustively) parsimony [8, 11, 31], compatibility [25], distance [12, 30], and maximum likelihood [1, 8, 9]. 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 on the same set of species [24]. 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 [28] and [29].

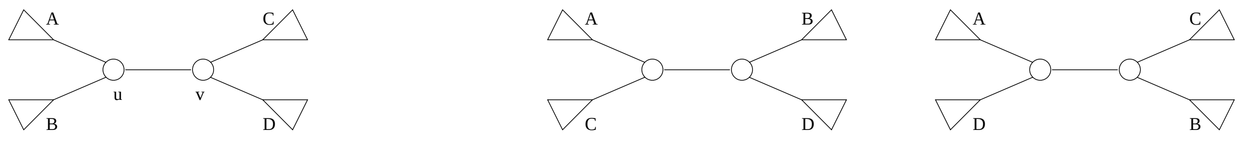

Figure 1: The two possible nni operations on an internal edge (u, v): exchange $B \leftrightarrow C$ or $B \leftrightarrow D$.

An nni operation swaps two subtrees that are separated by an internal edge (u, v), as shown in Figure 1. (An edge (u, v) is *internal* if neither u nor v is a leaf.) 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.

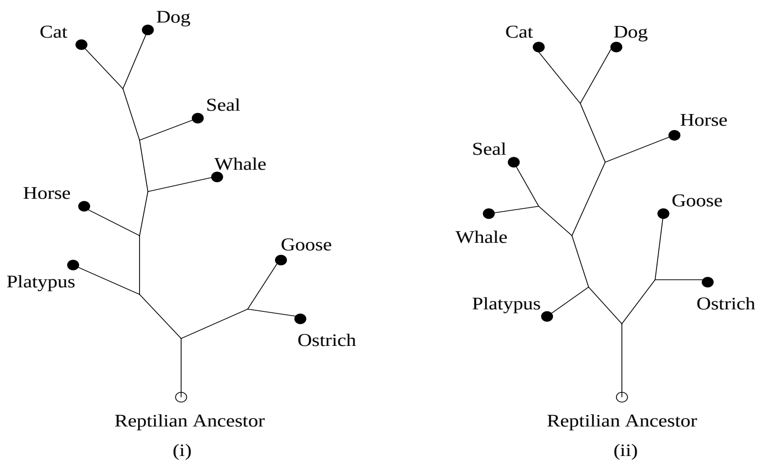

Figure 2: The nni distance between (i) and (ii) is 2.

The complexity of computing the nni distance has been open for more than 25 years (since [29]). The problem is surprisingly subtle as witnessed by the history of many erroneous results, disproved conjectures and a faulty NP-completeness proof [2, 19, 20, 23, 22, 26, 34]. The question is open even for the simpler case where the trees are unlabeled. The faulty NP-completeness proof [23] we mentioned above was for this case.

A phylogeny may also have *weights* on its edges, where an edge weight (more commonly 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 [24]. 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 [10]. The nni distance can be naturally extended to weighted phylogenies. An nni operation is simply charged a cost equal to the weight of the internal edge it operates on. Intuitively this extension of the nni distance is more sensitive to the tree topologies than the one in [24].

In this paper, we study the computational complexity and efficient approximation algorithms concerning the nni 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 (cf. § 2). The proof is quite involved and it uses the lower and upper bounds in [3, 33, 26] for sorting on a degree-3 tree by nni operations. The problem is also shown to be NP-complete for *unlabeled* trees, answering another open question in [3] (cf. § 3.) We will give an efficient approximation algorithm for computing the nni distance on weighted phylogenies with a performance ratio of $4 \log n + 4$, where n is the number of leaves (cf. § 4). Note that the approximation ratio does not depend on the weights. A special case of this result for unweighted phylogenies was recently reported in [26].

Unless otherwise mentioned, all the trees in this paper are trees with all *internal nodes* of degree 3 and with *unique labels* on leaves. We will mention it explicitly if a tree has nonuniquely labeled leaves or unlabeled leaves. Finally, two weighted trees are considered equal iff there is an isomorphism between them preserving topology, edge weights, if any, and leaf labels, if any.

1.1 The nni and Linear-Cost Subtree-Transfer Distances

The linear-cost subtree-transfer distance was introduced in [5]. This distance tries to address biological events such as *recombination* during the course of evolution of molecular sequences of organisms (because of which a single evolutionary tree is no longer sufficient to describe the evolutionary history of the sequences), and gives preferences to those recombinations which are more likely to occur than others. Somewhat surprisingly, although they are studied in parallel for very different reasons, it was demonstrated in [5] that the linear-cost subtree-transfer distance is in fact identical to the nni distance for unweighted phylogenies. As a result, all our results in this paper about the nni distance on unweighted phylogenies applies directly to the linear-cost subtree-transfer distance on unweighted phylogenies.

2 Computing the nni Distance is NP-complete

Theorem 1 *Let T_1 and T_2 be two trees with unique leaf labels and k be an integer. It is NP-complete to decide if $D_{nni}(T_1, T_2) \leq k$.*

Because the proof of Theorem 1 is quite lengthy and complicated, the reader is referred to the appendix for details of the proof. Here we will merely sketch the idea of the proof.

Theorem 1 is proved by a reduction from Exact Cover by 3-Sets (X3C), which is known to be NP-complete [13], to our problem. The X3C problem is defined as follows:

INSTANCE: A set $S = \{s_1, \ldots, s_m\}$, where $m = 3q$, and a collection of subsets $C_1, \ldots, C_n$, where $C_i = \{s_{i_1}, s_{i_2}, s_{i_3}\} \subset S$.

QUESTION: Is there an *exact cover* of S, that is, are there q disjoint subsets $C_{i_1}, \ldots, C_{i_q}$ such that $\cup_{j=1}^{q} C_{i_j} = S$?

Our reduction will construct two trees T_1 and T_2 with *unique* leaf labels, such that transforming T_1 to T_2 requires at most N (to be specified later) nni moves iff an exact cover of S exists. The following is an outline of our reduction.

Consider the linear tree shown in Figure 3, where $x_1, \ldots, x_k$ is a sequence of labels. For convenience, such a linear tree will be simply called a *sequence*. Sorting such a sequence means to transform it to another linear tree whose leaves are in a certain desirable order. This can be done by a sequence of nni operations. (During the sorting process, the tree may be nonlinear). The sequences we will construct later consist of small *coding regions* and larger *noncoding regions* that separate the coding regions. Sorting such a sequence will mean sorting each coding region to be in ascending order. The k^2-sized noncoding regions prevent the merging of adjacent blocks (of a coding region) in an optimal sorting procedure, *i.e.* it will not be beneficial for two blocks with the same corresponding subsequences to be merged and sorted together because it costs at most $ck \log k$ nni moves to sort a block and k^2 nni moves to bring a block across a noncoding region.

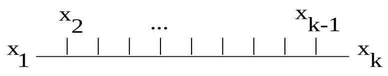

Figure 3: A linear tree with k leaves.

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" sequences for each pair $i \neq j$, in the sense that we cannot hope to significantly save nni moves by somehow combining the sorting of sequences S_i and S_j. Then for each subset $C_i = \{s_{i_1}, s_{i_2}, s_{i_3}\}$, we create three more sequences with the same ordering 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 and flipped (attached at the other end).

Here is the connection between an exact cover of 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, combine the sorting of the three pairs of sequences with identical ordering, and then 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 combined sorts involving merging and splitting, plus 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.

3 Computing the nni Distance Between Unlabeled Trees is NP-Complete

In this section, we prove the NP-completeness of the problem of computing the nni distance between two trees with *unlabeled* or *non-uniquely-labeled* leaves (for the unlabeled case, two trees are identical if and only if they have the same topology). First, we consider the non-uniquely labeled case.

Theorem 2 *For two given trees T_1 and T_2 with non-uniquely-labeled leaves and an integer k, it is NP-complete to decide if $D_{nni}(T_1, T_2) \leq k$.*

A flawed proof of Theorem 2 was published in [23].[1] Although Theorem 2 can be proved by using Theorem 1, here we give a direct and much simpler reduction from the X3C problem (cf. § 2) to this problem.

Assume that we are given the following instance of the X3C problem: A set $S = \{s_1, s_2, \ldots, s_m\}$ with $m = 3q$ and subsets $C_1, C_2, \ldots, C_n$ where $C_i = \{s_{i_1}, s_{i_2}, s_{i_3}\} \subset S$. If $n < q$, S clearly has no exact cover. So we assume $n \geq q$. Let $K > 0$ be an integer to be specified later. We construct two non-uniquely-labeled trees as in Figure 4. There are n *long arms* of length K in T_1, $n - q$ long arms

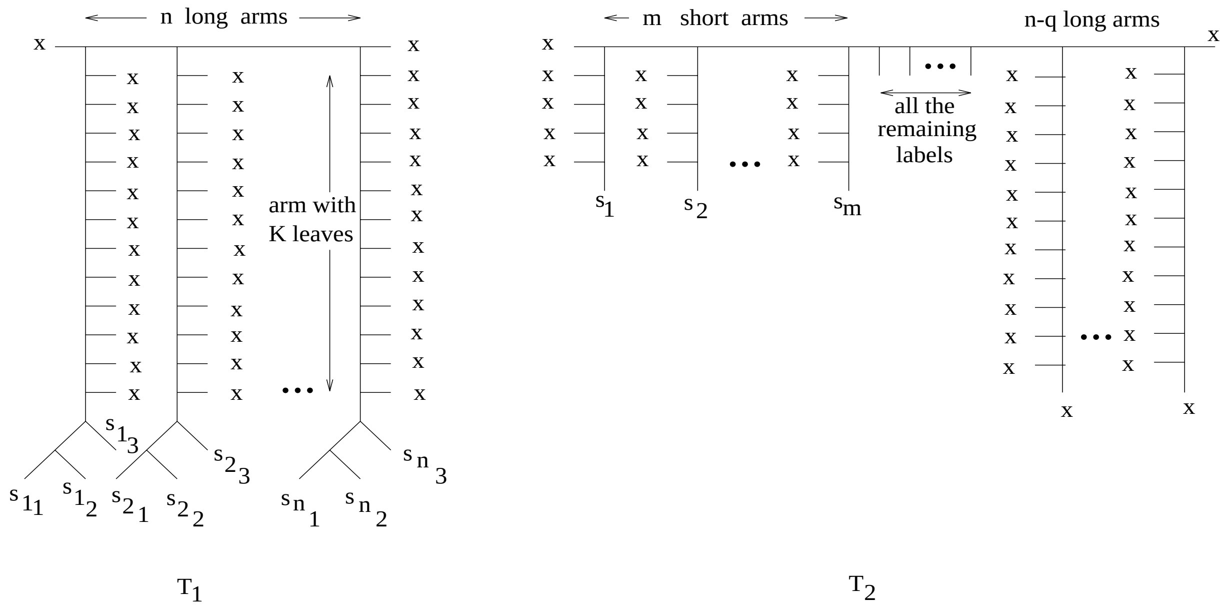

Figure 4: The trees T_1 and T_2 used in the reduction from X3C ($K = 12$).

of length K and m *short arms* of length $K/3$ in T_2. Each of these long (short, respectively) arms is a linear chain with K ($K/3$, respectively) leaves connected to it. These leaves are all identically labeled with a label $x \notin S$. At the end of the i-th long arm in T_1, we attach three leaves as shown in Figure 4 labeled with the three elements $s_{i_1}, s_{i_2}, s_{i_3}$ in C_i, respectively. At the end of the j-th short arm in T_2, we attach a leaf labeled with s_j. The extra $3n - m$ labels (in the multiset $\cup_{i=1}^{n} C_i - S$) not attached to the short arms in T_2 are placed between the short and long arms of T_2. At the bottom of each long arm in T_2, there is no additional labeled leaf.

[1]In [23], the author reduced the Partition problem to nni by constructing a tree of i nodes for a number i.

Lemma 3 *For any internal edge e within the long arms in T_1, the partition of (the multiset of) leaf labels induced by e in T_1 is different from the partition of the leaf labels induced by any edge e' in T_2.*

Proof: Every internal edge within a long arm in T_1 partitions the leaves of T_1 such that one partition contains 3 leaves labeled by the elements of S and an additional of p leaves $(0 \leq p \leq K)$ labeled by x. The only internal edges of T_2 that partition the leaves with 3 labels from the elements of S has at least $K+1$ leaves labeled by x in that same partition. ■

By Lemma 3 we need at least nK nni operations to create all the short arms of T_2 from the long arms of T_1. We choose K to be a sufficiently large integer (yet polynomial in m and n), for example, $K = n^2$. Then the proof of Theorem 2 is completed by proving the following lemma.

Lemma 4 *For all sufficiently large m and n, $D_{nni}(T_1, T_2) \leq nK + o(n^2)$ if and only if there is an exact cover of S.*

Proof: First, assume that there is an exact cover of S, say $C_{i_1}, \ldots, C_{i_q}$. Then, T_1 can be transformed to T_2 by using $nK + o(n^2)$ nni moves in the following manner.

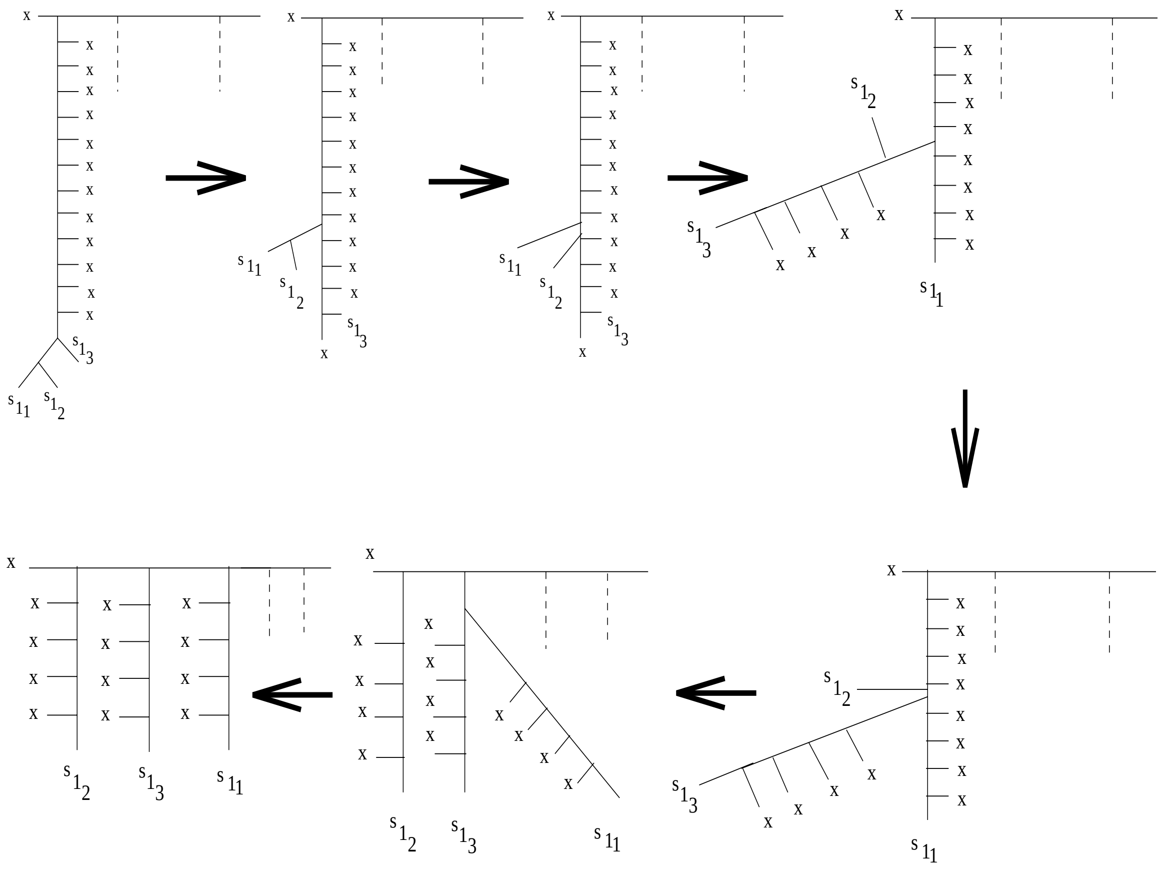

Figure 5: Transforming a long arm into three short arms.

- We transform each of the q long arms in T_1 corresponding to $C_{i_1}, \ldots, C_{i_q}$ into three short arms as follows. Consider the long arm L corresponding to $C_{i_j} = \{s_{l_1}, s_{l_2}, s_{l_3}\}$. (See Figure 5 for an illustration). Leave the leaf node with label s_{l_3} at the bottom of L, and move the two leaf nodes with labels s_{l_1} and s_{l_2} up L by a distance $K/3$. Then leave the leaf node with label s_{l_1} there, and move the remaining leaf node with label s_{l_2} together with the linear subtree of L of the last $K/3+1$ nodes up by a distance $K/3$. Now, leave the leaf node with label s_{l_2} there, and move two linear subtrees up. This in total needs $K+4$ nni moves for each long arm.

- For each long arm corresponding to a subset C_l not in the exact cover, we simply move the three leaf nodes at the bottom up (see Figure 6). This needs $K+3$ nni moves.

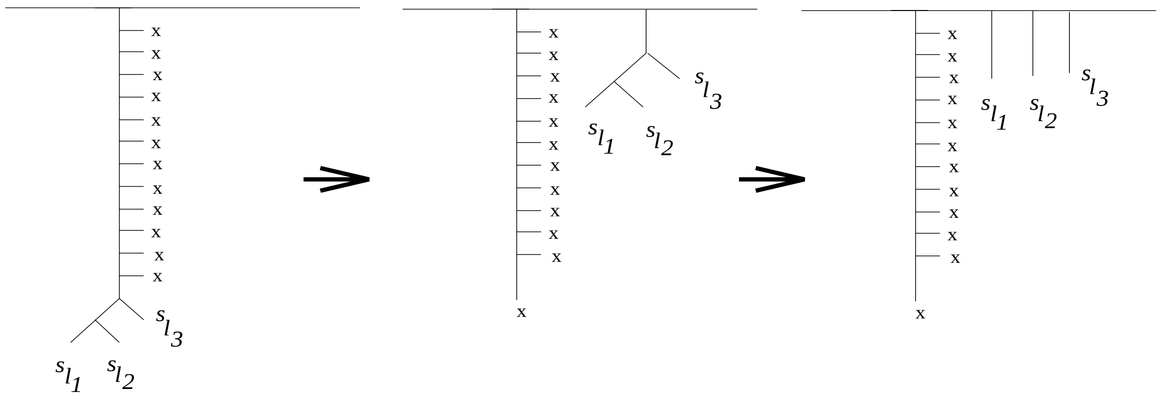

Figure 6: Transforming a long arm corresponding to a subset C_l not in the exact cover of S.

- Now, we have created the short and long arms of T_2, but not necessarily in the order they appear in T_2. Since $n \geq q = \frac{m}{3}$, it is trivial to use at most $o(n^2)$ nni operations[2] to shuffle the short and the long arms to finally obtain T_2.

The total number of nni moves we use is:

$$(K+4)\frac{m}{3} + (K+3)(n-\frac{m}{3}) + o(n^2) \leq nK + o(n^2),$$

as claimed.

Conversely, assume that there is no exact cover of S. Then, to convert T_1 to T_2, by Lemma 3, the nK nni moves are still necessary to move all leaves with labels in S up and create the short and long arms in T_2. However, since we do not have an exact cover, either one leaf with label in S must move down at least a short arm, or a short arm of length $K/3$ must merge with another short arm. Both costs at least $K/3$ extra nni moves. Thus in total, we must use at least $nK+K/3 > nK+o(n^2)$ nni moves (by our choice of K for sufficiently large values of n). This completes the proof. ■

Next, we consider the case when the leaves are unlabeled.

Theorem 5 *For any two given trees T_1 and T_2 with unlabeled leaves and an integer k, it is NP-complete to decide if $D_{nni}(T_1, T_2) \leq k$.*

[2]Obviously, this is a trivial upper bound which can be further improved. However, such improvements are not necessary for the correctness of this proof.

Proof: Again, we reduce the X3C problem to this problem. In Figure 4, the trees T_1 and T_2 use $m+1$ different labels: $s_1, \ldots, s_m$ and x. We denote x by s_{m+1}. For each leaf node with label s_i $(1 \leq i \leq m+1)$ in T_1 and T_2, we replace the node by a linear tree L_i of length iK^2. Let T_1' and T_2' be the resulting two trees (with unlabeled leaves). Since these linear trees L_i's are non-isomorphic and have distance of at least K^2 to each other, in order to transform T_1' to T_2', they have to be moved entirely to satisfy the bounds in Lemma 4 if there is an exact cover. Thus, the claim in Lemma 4 still holds, which implies Theorem 5. ■

4 Approximating the nni Distance on Weighted Phylogenies

In this section, we present an $O(n^2)$ time approximation algorithm with performance ratio $4\log n+4$ for computing the nni distance on weighted phylogenies. As mentioned in § 1, many phylogeny reconstruction methods produce weighted phylogenies. Hence the weighted nni distance problem is also very important in computational molecular biology. This algorithm generalizes the approximation algorithm in [26] for unweighted phylogenies, but is considerably more complicated.

Obviously, the nni operations can be performed only on internal edges and they do not change the weight of any edge. Given two weighted trees T_1 and T_2, for feasibility of transformation between T_1 and T_2 by using nni moves, we require that the following *feasibility conditions* are satisfied:

(i) For each leaf label a, the weight of the edge in T_1 incident to a is the same as the weight of the edge in T_2 incident to a.

(ii) The multisets of the weights of internal edges of T_1 and T_2 are identical.

If T_1 and T_2 do not satisfy the above two conditions, clearly T_1 cannot be transformed to T_2 by using nni operations.

Definition 1 Let T_1 and T_2 be two weighted trees with (possibly non-uniquely) labeled leaves. An internal edge e_1 of T_1 and an internal edge e_2 of T_2 are a *good edge pair* iff the following hold:

1. $w(e_1) = w(e_2)$.
2. The partition (of the multiset) of edge weights induced by e_1 in T_1 is the same as the partition of edge weights induced by e_2 in T_2.
3. The partition (of the multiset) of leaf labels induced by e_1 in T_1 is the same as the partition of leaf labels induced by e_2 in T_2.

Intuitively, if e_1 and e_2 form a good edge pair, then in order to transform T_1 to T_2, it is not necessary to perform an nni operation on e_1. An edge $e_1 \in T_1$ is *bad* if there is no edge e_2 in T_2 such that e_1 and e_2 form a good edge pair. Definition 1 applies to unweighted trees by letting all edges to have weight 1.

Theorem 6 *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 $4(1+\log n)$ in $O(n^2)$ time.*

In the rest of this section, we prove Theorem 6. The basic idea of the algorithm is as follows. We first identify "bad" components in the tree that need a lot of nni moves in the 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 the edges with heavy weights. 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.

Note that given an adjacency-list representation of a tree, it takes $O(1)$ time to update the tree after a single nni operation. Since the multisets of internal edge weights of T_1 and T_2 are the same, for simplicity, we use $\{e_1, e_2, \ldots, e_{n-3}\}$ to denote the set of internal edges of both T_1 and T_2. We renumber the edges of both T_1 and T_2, if necessary, in $O(n \log n)$ time such that $w(e_1) \le w(e_2) \le \ldots \le w(e_{n-3})$. Let $W = \sum_{i=1}^{n-3} w(e_i)$.

The following lemma provides a lower bound on $D_{nni}(T_1, T_2)$, which is needed to establish the performance ratio of our approximation algorithm. It holds for either unweighted or weighted trees.

Lemma 7 *If T_1 and T_2 have no good edge pairs, then $D_{nni}(T_1, T_2) \ge W$.*

Proof: For each internal edge $e_i \in T_1$, the partition of either the leaf labels or the edge weights induced by e_i is different from that induced by any edge in T_2. Hence, in order to transform T_1 into T_2, at least one nni operation must be performed on e_i with cost $w(e_i)$. So the total cost of transforming T_1 into T_2 is at least W. ■

We are ready to present our algorithm. First we consider the special case where T_1 and T_2 have no good edge pairs. By Lemma 7 it suffices to describe how to transform T_1 to T_2 with total cost at most $(4 \log n + 4)W$. The algorithm for this case consists of three steps.

Step 1: Pick an arbitrary leaf node r and transform T_1 into a balanced binary tree T_1' of height $\lceil \log n \rceil$ as shown in Figure 7, where the internal edges e_i $(1 \le i \le n-3)$ are positioned as follows: at the ith level $(i \ge 1)$, $e_{2^i - 1 + j}$ $(0 \le j < 2^i)$ is the jth edge from the left. It is easy to see that, on any path from r to a leaf, the weight of the internal edges are non-decreasing. This fact will be needed in Step 3 of our algorithm. Step 1 is carried out in three phases.

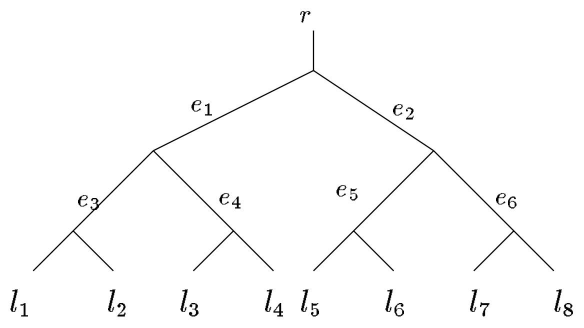

Figure 7: The balanced tree T_1' when $n = 9$.

Phase 1.1: Transform T_1 to a linear tree L as shown in Figure 8, where the internal edges $e_1, \ldots, e_{n-3}$ appear in some arbitrary order form left to right as in Figure 8. This can be done as follows: Treat T_1 as an rooted and ordered tree with root r. Each edge of T_1 will be either a *left* or a *right* edge. The edge from r to its child is drawn as a left edge. Consider any internal node u with children v_1 and v_2. If both v_1 and v_2 are internal nodes, or both are leaf nodes, we may chose either the edge (u, v_1) or the edge (u, v_2) as the left edge. If only one child (say v_1) is an internal node, we chose (u, v_1) as the left edge. The *left path* of T_1 is the path P from r to a leaf node using only left edges. If P contains all internal edges, then T_1 is already a linear tree and we are done. Otherwise, let e be an right internal edge with one end node on P. Perform an nni operation on e. Re-arranging the left and the right edges according to the above description, we obtain a new tree whose left path contains one more edge than the left path P of T_1. Repeat this process until T_1 is transformed into a linear tree. Clearly, at most one nni operation is performed on each internal edge of T_1 during the transformation process. Thus Phase 1.1 costs at most W and can be completed in $O(n)$ time.

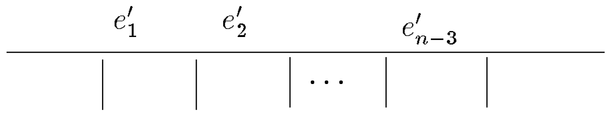

Figure 8: The linear tree L.

Phase 1.2: Similar to Phase 1.1, we transform T'_1 to some linear tree L' in $O(n)$ time with total cost at most W. Let the internal edges appear in L' from left to right as

$$e''_1, e''_2, \cdots, e''_{n-3}. \tag{1}$$

Phase 1.3: We use an analogue of merge sort to transform the linear tree L to the linear tree L'. The transformation is the same as in [26], but focuses on the internal edges instead. This transformation costs $W \log n$ and can be performed in $O(n \log n)$ time.

To transform T_1 to T'_1, we perform the nni operations in Phase 1.1, followed by the nni operations in Phase 1.3, followed by the inverse of the nni operations in Phase 1.2. Thus Step 1 can be completed in $O(n \log n)$ time with total cost at most $(2 + \log n)W$.

Step 2: Transform T_2 to a balanced binary tree T'_2. The position of the internal edges of T'_2 are identical to that of T'_1. Similar to Step 1, this can be done in $O(n \log n)$ time with total cost at most $(2 + \log n)W$.

Step 3: Transform T'_1 to T'_2. Since both trees have identical internal structure, we only need to move the leaves of T'_1 to their corresponding positions in T'_2. Let $r = l_0, l_1. \ldots, l_{n-1}$ denote the leaf nodes in T'_1 in counter-clockwise order starting at the root leaf r. Let $l_{i'}$ be the destination in of the leaf l_i $(1 \leq i \leq n-1)$ in T'_2. Denote the permutation mapping i to i' by π.

We move the leaves in T'_1 according to π. Write π as a product of disjoint cycles, and process each cycle $C = (i_1, i_2, \ldots, i_k)$ in turn as follows. The lowest internal edge above the leaf l_{i_j} will be called the *internal edge adjacent to* l_{i_j}. Without loss of generality, assume l_{i_1} have the lightest adjacent internal edge weight among all l_{i_j} $(1 \leq j \leq k)$. Now move the leaf l_{i_1} to the leaf $l_{\pi(i_1)} = l_{i_2}$ by a sequence of nni operations. Then, move l_{i_2} to l_{i_3} and so on. The last nni operation that swaps l_{i_1} into place also starts l_{i_2} on its way up and we continue moving it to l_{i_3}. Finally, leaf l_{i_k} is moved to take the place vacated by l_{i_1}. This completes the processing of the cycle C. It is easy to see that

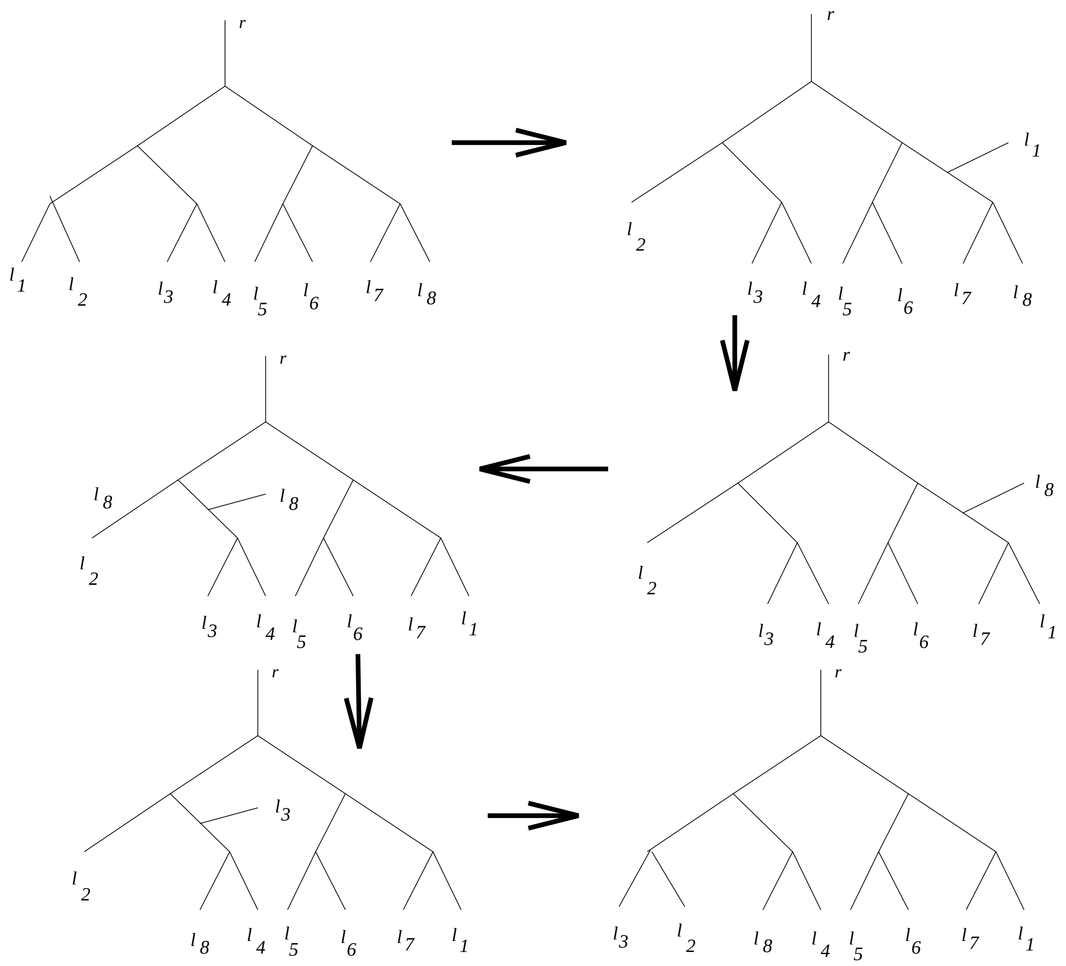

Figure 9: The nni operations for processing the cycle $(1, 8, 3)$.

this process restores the original topology of internal edges. Figure 9 illustrates this process for a cycle $(1, 8, 3)$.

Since the height of T_1' is $\lceil \log n \rceil$, it is easy to see that Step 3 takes at most $O(n \log n)$ time. Next, we analyze the total cost of Step 3. For simplicity, we assume that $n = 2^m + 1$ (see Figure 7). Partition the set of the internal edges of T_1' into $E_1 \cup \cdots \cup E_{m-1}$, where E_i consists of the internal edges at level i. Let W_i be the total weight of the edges in E_i. For any $e \in E_i$, there are 2^{m-i} leaves below e. Thus there are at most 2^{m-i} nni operations that move a leaf "down" e and at most 2^{m-i} nni operations that move a leaf "up" e. Furthermore, consider the processing of a cycle $C = (i_1, i_2, \ldots, i_k)$. Let e_{i_j} $(1 \leq j \leq k)$ be the internal edge in E_{m-1} that is above the leaf l_{i_j}. Let $w(C)$ denote the total weight of the edges $e_{i_1}, \ldots, e_{i_k}$. During the processing of C, the up and down nni operations coincide for each e_{i_j} $(2 \leq j \leq k)$. Thus the cost contribution of the edges in E_{m-1} for processing C is at most: $2w(e_{i_1}) + \sum_{j=2}^{k} w(e_{i_j})$ which, by our choice of cycle start i_1, is at most $1.5w(C)$. Summing up over all cycles C in π, the total cost D for Step 3 is at most:

$$D \leq 3W_{m-1} + \sum_{i=1}^{m-2} 2^{m-i+1} W_i = \sum_{i=1}^{m-1} 2^{m-i+1} W_i - W_{m-1}.$$

In the linear combination $\omega = \sum_{i=1}^{m-1} 2^{m-i+1} W_i$, the coefficient of W_i is twice that of W_{i+1}. By our choice of the ordering of the internal edges, we have $W_i \leq W_{i+1}/2$. It follows that ω is maximized when $W_i = W_{i+1}/2$, i.e. all edges have uniform weight. Together with the only other constraint $\Sigma_{i=1}^{m-1} W_i = W$ and noting that there are $2^m - 2$ internal edges in all, this gives $W_i = 2^i * W/(2^m - 2)$ for the worst case cost. Assuming for convenience that $m \geq 5$, so that $2^{m-1}/2 \geq 2(m-1)$, we have:

$$D \leq \sum_{i=1}^{m-1} 2^{m-i+1} W_i - W_{m-1} = \frac{(m-1)2^{m+1} - 2^{m-1}}{2^m - 2} W \leq 2(m-1)W \tag{2}$$

Thus the total cost of Step 3 is at most $2(\log n)W$.

To transform T_1 to T_2, we first perform the nni operations in Step 1 (transforming T_1 to T_1'), followed by the nni operations in Step 3 (transforming T_1' to T_2'), followed by the inverse of the nni operations in Step 2 (transforming T_2' to T_2). The complete algorithm can be done in $O(n \log n)$ time with total cost at most $4(1 + \log n)W$.

Next we consider the general case when T_1 and T_2 may have some good edge pairs. We first need to identify the set E'' of edges in T_1 that form good edge pairs with edges in T_2. The proof of the following lemma can be found in [7].

Lemma 8 *[7] Let T_1 and T_2 be two trees, each with n leaves. Then, the set of edges of T_1 which partition the leaf labels similarly as some edge of T_2 can be found in $O(n)$ time.*

Using the above lemma, it is trivial to find the set E'' in $O(n^2)$ time: First, we find the set of all edges $E''' \supseteq E''$ of T_1 that partition the leaf labels similarly as some edge of T_2 in $O(n)$ time using Lemma 8. It is trivial to identify the edges in E'' that also satisfy the other conditions of Definition 1, in a total of $O(n^2)$ time.

Let E' be the set of internal edges of T_1 not in E''. Similar to Lemma 7, we can show that at least one nni operation on each internal edge $e \in E'$ is needed to transform T_1 to T_2. Thus:

$$D_{nni}(T_1, T_2) \geq W' = \sum_{e \in E'} w(e) \tag{3}$$

The edges in E' induce in T_1 a subgraph consisting of one or more connected components each of which is a subtree of T_1. These connected components can easily be found in $O(n)$ time. To transform T_1 to T_2, we perform the approximation algorithm described above on each such component. The total cost is bounded by $4(1 + \log n)W'$. The algorithm takes $O(n^2)$ time. This completes the proof of Theorem 6.

5 Conclusion and Open Problems

The results reported in this paper have been obtained as a part of our project of building a software package for comparing phylogenetic trees. Several open questions still remain:

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

- The nni operation is similar to and slightly more powerful than the *rotation operation* discussed in [3, 32]. Is it NP-complete to compute the rotation distance? Can we approximate the rotation distance better than the trivial ratio 2? This question turns out to be subtler than it appears to be. Partial results in this direction can be found in [27].

Acknowledgments: We wish to thank J. Felsenstein for explaining to us the biological motivation for comparing weighted phylogenies.

References

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

[2] 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.

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

[4] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp and L. Zhang, On distances between phylogenetic trees, in *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms*, 1997, pp. 427-436.

[5] B. DasGupta, X. He, T. Jiang, M. Li, and J. Tromp, On the linear-cost subtree-transfer distance, to appear in the special issue in *Algorithmica* on computational biology, 1998.

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

[7] W. H. E. Day, Optimal algorithms for comparing trees with labeled leaves, *J. Classification*, 2(1985), 7-28.

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

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

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

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

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

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

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

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

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

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

[18] J. Hein, T. Jiang, L. Wang, and K. Zhang, On the complexity of comparing evolutionary trees, *Discrete Applied Mathematics* 71, 153-169, 1996.

[19] 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.

[20] 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.

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

[22] 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.

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

[24] 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.

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

[26] M. Li, J. Tromp, and L. Zhang, Some notes on the nearest neighbor interchance distance, *Journal of Theoretical Biology*, 182(1996), 463-467.

[27] M. Li and L. Zhang, Better approximation of diagonal-flip transformation and rotation transformation. *COCOON98*, Aug. 12-14, 1998, Taipei, Taiwan.

[28] 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.

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

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

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

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

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

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

APPENDIX

Proof of Theorem 1: Using the discussion and notations in § 2, we now continue to give the detailed proof of Theorem 1. 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 ensure that sorting one such sequence will never help to sort others? How can we ensure that it is most beneficial to bring the sequences $S_{j_1}, S_{j_2}, S_{j_3}$ corresponding to elements in a subset C_j to their counterparts to get sorted, and not the other way around?

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 3. 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 because the upper bound in [3, 26] and the lower bound in [33] (see [26] for the calculation) do not match, this simple idea does not work. The reason is that it is impossible to guarantee that sorting one of these pieces will not help to sort the other pieces (by merging, sorting together, and then splitting). So we first have to find sequences that are hard to sort and do not help each other in sorting.

Lemma 9 *For any constant $\epsilon > 0$, there exists a constant $c > \epsilon$ and infinitely many k for which there are 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, and (ii) each of them takes at most $ck\log k$ nni moves to sort, and (iii) it takes at least $(c-\epsilon)(2k)\log(2k)$ nni moves to sort both of them together, i.e. the sequence xy.*

Proof: From the results in [3, 26, 33], 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 moves to sort any sequence of length k, divided by $k\log k$. Since $\frac{1}{4} - o(1) \le c_k \le 1 + o(1)$, infinitely many k satisfy

$$c_{2k} \ge c_k - \frac{\epsilon}{3}. \tag{4}$$

Since any length $2k$ sequence can be sorted by first sorting both length k halves and then merging them using less than $2k$ nni moves, we have $c_{2k}2k\log(2k) < 2c_k k\log k + 2k$ hence $c_{2k} \le c_k + o(1)$. Taking xy to be a hardest sequence of length $2k$ for large enough k satisfying the inequality (4), and taking $c = c_k$, necessarily satisfies conditions (ii) and (iii). Without loss of generality, let $w_x \le w_y$ be the costs of sorting x and y. Then the above sorting method for xy shows that $c_{2k}2k\log(2k) \le w_x + w_y + 2k$ hence, together with the fact that $w_y \le ck\log k$, we have:

$$w_y \ge w_x \ge (c - \frac{\epsilon}{3})2k(1+\log k) - 2k - ck\log k \ge (c - \frac{2\epsilon}{3})k\log k - 2k(1 + \frac{\epsilon}{3} - c) \ge (c-\epsilon)k\log k,$$

proving (i). Furthermore, infinitely many of these c_k must be arbitrarily close to each other, giving a single c to satisfy the lemma. ■

Let $\epsilon = 1/8$, k a sufficiently large integer satisfying Lemma 9 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 binary strings $\alpha_1, \ldots, \alpha_m$ in $\{0,1\}^{\lceil \log m \rceil}$. The sequence S_i $(1 \le i \le m)$ is obtained from the string α_i by performing the following: 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 between every pair of adjacent x/y blocks a *delimiter* sequence of length k^2 with unique labels.

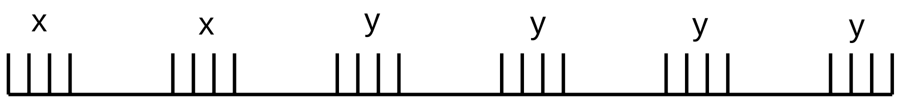

Figure 10: The structure of the string S_i corresponding to binary string 011.

This results in sequences $S_1, \ldots, S_m$, all with distinct labels. Each pair of them differ by at least m^3 x or y blocks at some position. The m sequences will have specific orientations in the tree; let's refer to one end as *head* and the other end as *tail.*

Figure 10 shows the structure of a single sequence using binary string 011. To make the picture clear, we pretend m^3 is only 2. Assume both x and y blocks have length $k = 4$. The gaps between blocks represent the noncoding regions whose leaves are not shown explicitly. Recall that sorting this sequence entails sorting all the blocks plus moving the point of attachment (connecting point to the tree) from the left to the right of the sequence. The latter requires $k^2(m^3 \log m - 1)$ moves to cross the noncoding regions. To this we need to add for each block the cost of sorting it and having the attachment move through it. Let q_i denote the sum of the cost of sorting and passing through each block in the sequence.

We are now ready to do the reduction. From the set $S = \{s_1, \ldots, s_m\}$ and the subsets $C_1, C_2, \ldots, C_n$, we construct the tree T_1 as follows. (The construction of T_2 will be described later). For each element $s_i \in S$, T_1 has a sequence S_i as defined above. For each subset $C_i = \{s_{i_1}, s_{i_2}, s_{i_3}\}$, we create three sequences $S_{i,i_1}, S_{i,i_2}, S_{i,i_3}$, with the same ordering as $S_{i_1}, S_{i_2}, S_{i_3}$, respectively, but with different and unique labels. Notice that we are not allowed to repeat labels.

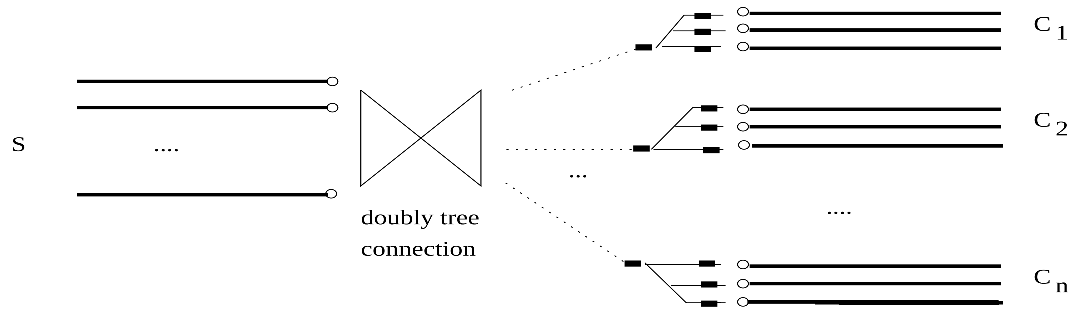

Figure 11: The structure of the tree T_1

Figure 11 outlines the structure of the 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; and a small black rectangle represents a **one-way circuit** as illustrated in Figure 13(i). The heads of the m sequences at the left of Figure 11 are connected by a (double) tree-type connection 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 subset C_i.

As mentioned before, a sequence S_i and a counterpart S_i' with identical ordering will be brought together to be sorted. We now consider the cost of combining the sorting process of S_i and S_i'. At the beginning of this process, the attachments of S_i and S_i' (i.e. the edge linking it to the main structure of T_1) are at the left end of S_i and S_i', respectively (see Figure 12). During the sorting process, the attachments move through S_i and S_i' from left to right and their blocks are sorted on the way. At the end of the process, the attachments are at the right end of S_i and S'_i, with all their blocks sorted. In Figure 12, the top (bottom, respectively) long horizontal line represents the sequence S_i (S'_i, respectively). The bridge between the two sequences represents the attachments to the main structure (which is not shown). Consider the processing of a block B in S_i and its corresponding block B' in S'_i. The attachments move from left to right. Each leaf in B' moves four steps across the attachments to join its partner in B, followed by one move to advance the attachments. (In Figure 12, one leaf of B' has joined its partner in B and the attachment has passed one leaf of B). After $5k$ moves, all leaves of B' have joined their partners in B and the attachments are at the right of B. We spend another k moves to get attachment of S_i back to the left of B. Now we can proceed as in the single sequence case to sort the combined blocks B and B'. (This will move the attachments to the right of the block B again). Finally, we move the attachment of S from right to left again in k moves (to get ready for splitting process), and spend another $5k$ moves to split B and B'. At the end, both B and B' are sorted and the attachments end up on the right of both blocks. The total cost for processing B and B' is thus $12k$ plus the cost of sorting a single block B.

Compared to sorting S_i and S'_i separately, because we saved q_i for sorting (and passing through) all coding blocks of a sequence, but spent extra $12k$ moves on each coding block, we thus save

$$q_i - 12km^3 \log m \tag{5}$$

nni moves.

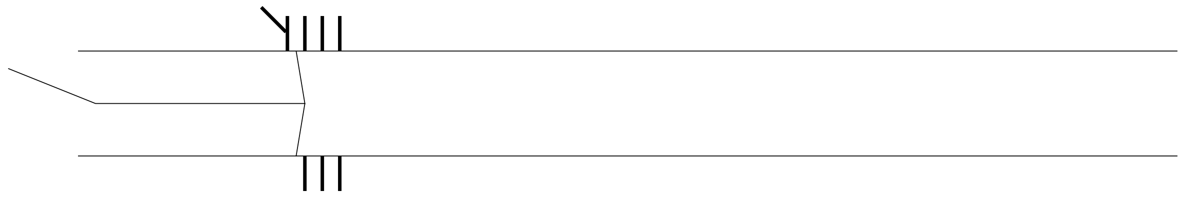

Figure 12: Merging two sequences.

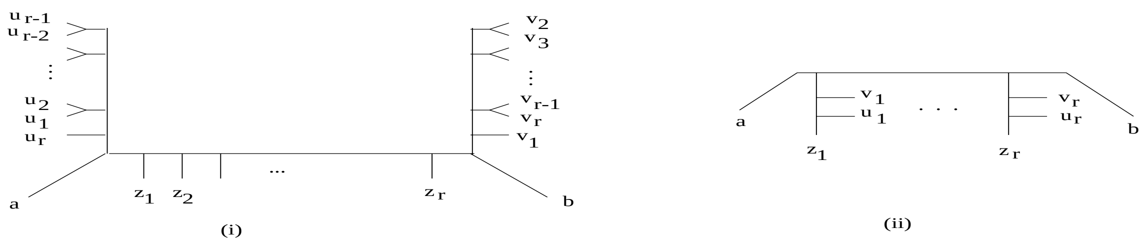

Figure 13: One-way circuit.

A one-way circuit is shown in Figure 13 (i) (where r is an integer to be determined later). The leaf labeled by a (b, respectively) is the *entrance* (*exit*, respectively) of the one-way circuit. The counterpart of the one-way circuit in T_2 is as shown in Figure 13 (ii). It is designed for the purpose of giving "free rides" to a subtree moving first from the entrance to the exit and then later from the exit back to the entrance, while transforming the one-way circuit (i) into its counterpart (ii). On the other hand, it imposes a large extra cost for subtrees first moving from the exit to the entrance and then back to the exit. This can be seen as follows.

In any optimal transformation of circuit (i) to (ii), the leaves marked by u's are paired up with the leaves marked by z's first and then the leaves marked by 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.

A precise breakdown of the cost is as follows: $(r-3)/2$ steps to move u_r up, then $\frac{r-1}{2}$ times 6 steps to move each u pair down between the proper z's and pair them up, and one final step to pair u_r. The exact same number of steps is needed for the symmetric pairing of v's. Hence in total we need (assuming r is odd)

$$2(\frac{r-3}{2} + 6\frac{r-1}{2} + 1) = r - 3 + 6(r-1) + 2 = 7r - 7$$

nni moves. Note that a subtree situated at 'a' can initially pair up with u_r in 2 steps and move together with it, spending 3 more steps to pop off just before u_r pairs with z_r, to end up at 'b'. It can later spend another 5 steps to move together with v_1 ending up back at 'a'. A subtree going first from 'b' to 'a' and then back to 'b' could only be done 'for free' by pairing with v_1 first and with u_r later, since these are the only leaves to move away from 'b' and 'a' respectively in an optimal transformation. But for v_1 to reach 'a' with minimum cost requires collapsing all the v's which imposes an extra cost on pairing u's with z's later. The least penalty for moving from 'b' to 'a' back to 'b' is thus for v_1 not to take the shortcut which costs an extra $\frac{r}{2}$ steps.

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.

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 13 (i) is changed to (ii).

In order to transform T_1 into T_2, we need to sort the sequences S_i and $S_{i,j}$ and convert each one-way circuit to the structure shown in Figure 13 (ii). If the set S has an exact cover $C_{i_1}, \ldots, C_{i_q}$, we can do the transformation efficiently as follows. For each $C_j = \{s_{j_1}, s_{j_2}, s_{j_2}\}$, $(j = i_1, \ldots, i_q)$, in the cover, we send the three sequences $S_{j_1}, S_{j_2}, S_{j_3}$ on the left of T_1 to their counterparts $S_{j,j_1}, S_{j,j_2}, S_{j,j_3}$ on the right of T_1, combine the sorting of each pair as explained in Figure 12, then move the sorted $S_{j_1}, S_{j_2}, S_{j_3}$ sequences back. During this process we also get each one-way circuit involved into the correct shape. We then sort the other un-sorted sequences $S_{i,j}$ on the right of T_1 and get their leading one-way circuits into the correct shape.

The total cost N for this process is calculated as follows. Recall that we send precisely q groups of sequences from the left to the right of T_1.

1. The overhead for these q groups to cross the tree connection network: $q(\log m + \log n) + O(1)$ nni moves. This is done by grouping three sequences in each group first, then moving them as one unit.

2. The cost of crossing the q toll sequences of length m^2 before the first batch of one-way circuits: qm^2 nni moves.

3. Converting each one-way circuit to the structure in Figure 13 (ii) costs $7r - 7$ nni moves.

4. Moving a group of sequences across a one-way circuit and back costs $O(1)$ extra nni moves, for each of the q groups.

5. The cost of sorting $3q$ pairs of sequences (by combining).

6. the cost of sorting the remaining $3n - 3q$ sequences individually.

The following lemma completes the reduction and thus the proof of Theorem 1.

Lemma 10 *If the set S has no exact cover, then $D_{nni}(T_1, T_2) \geq N + m^2/2$.*

Proof: Suppose that S has no exact cover. Then to transform T_1 into T_2, we either have to send more than q groups (some groups with less than 3 sequences) to the right crossing the one-way circuits, or some sequence S_i is sorted separately from $S_{j,i}$'s or some sequence S_i is sorted together with a "wrong" sequence $S_{j,h}$, where $h \neq i$. In the first case, the cost will be increased by m^2 nni moves, which is the cost of moving an extra group past a delimiter sequence of length m^2. In the last case, at least one segment of m^3 x's is sorted together with a segment of y's. By Lemma 9 and the choice $\epsilon = \frac{1}{8}$, this is not much better than sorting the two segments separately and costs at least $(c - \frac{1}{8})m^3 k \log k - m^3 k$ more nni moves than sorting one such segment, which is larger than m^2 for sufficiently large k and m, since $c > \frac{1}{4}$. The second case introduces an extra cost of $q_i - 12km^3 \log m - m^2 = \Omega((m^3 \log m) k \log k)$ by equation (5) and Lemma 9, which is again larger than m^2 for sufficiently large k and m.

Notice that in the above definition of N, the bounds in items 2,3,6 are all optimal. The bounds in items 1 and 5 are the worst case overheads and may not be optimal. But these two items only account for $O(m(\log m + \log n))$ nni moves, which is not sufficient to compensate for the extra cost m^2 given above. ■