Handbook of Combinatorial Optimization
D.-Z. Du and P.M. Pardalos (Eds.) pp. 35 - 76
©1998 Kluwer Academic Publishers

Computing Distances between Evolutionary Trees

Bhaskar DasGupta[1]
Rutgers University. E-mail: bhaskar@crab.rutgers.edu

Xin He[2]
SUNY at Buffalo. E-mail: xinhe@cs.buffalo.edu

Tao Jiang[3]
McMaster University. E-mail: jiang@maccs.mcmaster.ca

Ming Li[4]
City University of Hong Kong and University of Waterloo. E-mail: mli@math.uwaterloo.ca

John Tromp[5]
CWI. E-mail: tromp@cwi.nl

Lusheng Wang[6]
City University of Hong Kong. E-mail: lwang@cs.cityu.edu.hk

Louxin Zhang[7]
National University of Singapore. E-mail: lxzhang@iss.nus.sg

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

[2] Supported in part by NSF grant 9205982 and CGAT.

[3] Supported in part by NSERC Operating Grant OGP0046613 and CGAT.

[4] Supported in part by the NSERC Operating Grant OGP0046506 and CGAT.

[5] Supported in part by CGAT and NSERC Internation Fellowship.

[6] Supported in part by Hong Kong Research Council.

[7] Supported in part by CGAT.

Contents

1 **Introduction** **2**

2 **The Nni and Subtree-transfer Distances** **4**
2.1 The Case of Unweighted Trees 5
2.2 The Case of Weighted Trees 6

3 **Computing the Nni Distance** **7**
3.1 Unweighted trees: Computing nni distance exactly 7
3.2 Unweighted trees: Computing nni distance approximately 15
3.3 Weighted trees: generalizing the nni distance 18

4 **Computing the Subtree-Transfer Distance** **19**
4.1 The NP-hardness 20
4.2 An Approximation Algorithm of Ratio **3** 21

5 **Linear-Cost Subtree-Transfer Distance on Weighted Phylogenies** **23**
5.1 An NP-hardness Result 23
5.2 An Approximation Algorithm 25

6 **The Rotation Distance** **31**
6.1 Rotation and its equivalences 31
6.2 Upper and lower bounds for the rotation distance 34
6.3 Approximating the rotation and diagonal flip distances 35
6.4 Miscellaneous remarks 38

7 **Open Questions** **38**

References

1 Introduction

Comparing objects to find their *similarities* or, equivalently, *dissimilarities*, is a fundamental issue in many fields including pattern recognition, image analysis, drug design, the study of thermodynamic costs of computing, cognitive science, *etc.* Various models have been introduced to measure the degree of similarity or dissimilarity in the literature. In the latter case the degree of dissimilarity is also often referred to as the *distance*. While some distances are straightforward to compute, *e.g.* the Hamming distance for binary strings, the Euclidean distance for geometric objects; some others are formulated as combinatorial optimization problems and thus pose nontrivial challenging algorithmic problems, sometimes even uncomputable, such as the universal information distance between two objects [4].

Distances based on the notion of *economic transformation* usually fall in the latter category. In a nutshell, a transform based distance model assumes a set of transformation *operations* or *moves*, each associated with a fixed *cost*, which can be applied on the objects in the domain studied. The set of transformation operations should be *complete* in the sense that any object can be transformed into any other object by performing a sequence of such operations. The distance between two objects is then defined as the minimum cost of any sequence of operations transforming one object into the other. [8] The best known transform based distances are perhaps the *edit* distances for strings [40], labaled trees [43, 48] and graphs [49] using operations insertion, deletion, and replacement. The edit distances have applications in many fields including computational molecular biology and text processing, and have been studied extensively in both the literature and practical settings. For example, the UNIX command *diff* is essentially based on string edit distance. String edit distance is also a particularly suitable model for biological molecular sequence comparison because the edit operations often represent the most common form of evolutionary events.

In this chapter, we survey recent results on some transformation based distances for *evolutionary trees* (also called *phylogenies*). Such a tree is an *unordered* tree, it has uniquely labeled leaves and unlabeled interior nodes, can be *unrooted* or *rooted* if the evolutionary origin is known, can be *unweighted* or *weighted* if the evolutionary length on each edge is known, and usually has internal nodes of degree 3. Reconstructing the correct evolutionary tree for a set of species is one of the fundamental yet difficult problems in evolutionary genetics. Over the past few decades, many approaches for reconstructing evolutionary trees have been developed, including (not exhaustively) parsimony [12, 15, 39], compatibility [32], distance [16, 38], maximum likelihood [12, 13, 3]. 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 [28]. It is thus of interest to compare evolutionary trees produced by different methods, or by the same method on different data. Several distance models for evolutionary trees have been proposed in the literature. Among them,

[8] Usually the operations are reversible so we do not have to specify the direction of a transformation.

the best known is perhaps the *nearest neighbor interchange* (nni) distance introduced independently in [37] and [35]. We will focus on nni and a closely related distance called the *subtree-transfer* distance introduced in [19, 20] for dealing with evolutionary histories involving events like *recombinations* or *gene conversions*. Some variants of these distances will also be discussed. Since computing each such distance is NP-hard, our main interest is in the design of efficient approximation algorithms with guaranteed performance ratios.

The rest of the chapter is organized as follows. We first formally define the nni and subtree-transfer distances as well as a variant of subtree-transfer distance, called the *linear-cost subtree-transfer distance*, in Section 2. It is also demonstrated that the nni distance coincides with the linear-cost subtree distance on unweighted evolutionary trees. Section 3 presents results concerning the nni distance on both weighted and unweighted evolutionary trees. In particular, we give some tight upper and lower bounds on the nni distance, prove that computing the nni distance is NP-hard, which was a long-standing open problem, and give some logarithmic ratio approximation algorithms. Section 4 is concerned with the subtree-transfer distance on unweighted evolutionary trees. The main results include the NP-hardness of computing the subtree-transfer distance and an approximation algorithm with ratio 3. In Section 5, we consider the linear-cost subtree-transfer distance on weighted evolutionary trees and present a ratio 2 approximation algorithm. In Section 6, we discuss a variant of the nni distance for rooted, ordered trees, called the *rotation distance*, and present a nontrivial approximation algorithm. Some open problems are listed in Section 7.

We assume the reader has the basic knowledge of algorithms and computational complexity (such as NP and P). Consult, *e.g.*, [17] otherwise. 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*.

2 The Nni and Subtree-transfer Distances

In this section, we first define the nni, subtree-transfer, and linear-cost subtree-transfer distances for unweighted trees. Then we extend the nni and linear-cost subtree-transfer distances to weighted trees.

2.1 The Case of Unweighted Trees

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* 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. Although the distance has been studied extensively in the literature [37, 35, 47, 6, 10, 5, 25, 26, 29, 42, 30, 31, 33], the computational complexity of computing it has puzzled the research community for nearly 25 years until recently [7].

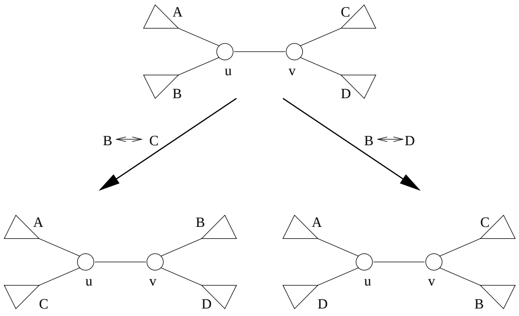

Figure 1: The two possible nni operations on an internal edge (u, v): exchange $B \leftrightarrow C$ or $B \leftrightarrow D$.

An nni operation can also be viewed as moving a subtree past a neighboring internal node. A more general operation is to transfer a subtree from one place to another arbitrary place. Figure 2 shows such a *subtree-transfer* operation. The subtree-transfer distance, $D_{st}(T_1, T_2)$, between two trees T_1 and T_2 is the minimum number of subtrees we need to move to transform T_1 into T_2 [19, 20, 22, 8, 7].

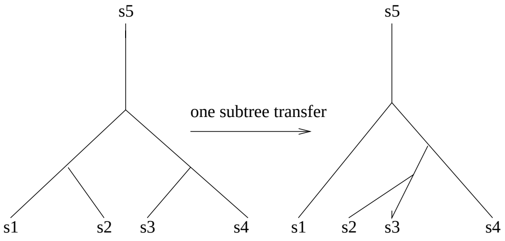

Figure 2: An example of subtree-transfer.

It is sometimes appropriate in practice to discriminate among subtree-transfer operations as they occur with different frequencies. In this case, we can charge each subtree-transfer operation a cost equal to the distance (the number of nodes passed) that the subtree has moved in the current tree. The *linear-cost* subtree-transfer distance, $D_{lcst}(T_1, T_2)$, between two trees T_1 and T_2 is then the minimum total cost required to transform T_1 into T_2 by subtree-transfer operations [8, 7]. Clearly, both subtree-transfer and linear-cost subtree-transfer models can also be used as alternative measures for comparing evolutionary trees generated by different tree reconstruction methods.

It is easy to demonstrate that the linear-cost subtree-transfer and nni distances in fact coincide. As mentioned before, an nni move is just a restricted subtree-transfer where a subtree is only moved across a single node. (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. However, it will soon become clear that the two models are different on weighted trees.

2.2 The Case of Weighted Trees

An evolutionary 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 evolutionary tree reconstruction methods, including the distance and maximum likelihood methods, actually produce weighted evolutionary trees. Comparison of weighted evolutionary trees has recently been studied in [28]. 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 [14]. Both the linear-cost subtree-transfer and nni models can be naturally extended to weighted trees. The extension for nni is straightforward: An nni is simply charged a cost equal to the weight of the edge it operates on. In the case of linear-cost subtree-transfer, although the idea is immediate, *i.e.* a moving subtree should be charged for the weighted distance it travels, the formal definition needs some care and is given below.

Consider (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 2.2 gives an example. The edge-weights of the given tree are normalized so that their total sum is 1. 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 $w(e_2) + w(e_3) + w(e_6)$.

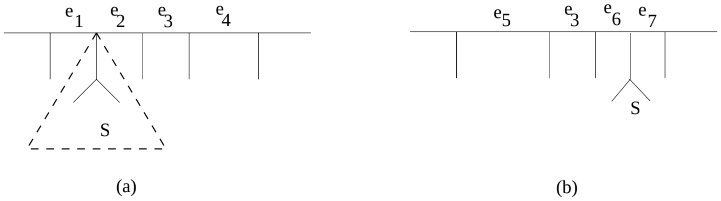

Figure 3: Subtree-transfer on weighted phylogenies. Tree (b) is obtained from tree (a) with one subtree-transfer.

Note that for weighted trees, 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. Intuitively both these measures, especially the nni distance, are more sensitive to the tree topologies than the one in [28].

3 Computing the Nni Distance

In this section, we discuss the complexity of computing the nni distance between labeled or unlabeled trees, either exactly or approximately. We first discuss the case of unweighted trees, and then consider the more general case of weighted trees.

3.1 Unweighted trees: Computing nni distance exactly

The *nearest neighbor interchange* (nni) distance was introduced independently in [37] and [35]. The complexity of computing the nni distance has been open for 25 years (since [37]). The problem is surprisingly subtle given the history of many erroneous results, disproved conjectures, and a faulty NP-completeness proof [47, 5, 25, 26, 29, 30, 33][9]

K. Culik II and D. Wood [6] (improved later by [33]) proved that $n \log n + O(n)$ nni moves are sufficient to transform a tree of n leaves to any other tree with the same set of leaves. D. Sleator, R. Tarjan, and W. Thurston [42] proved an $\Omega(n \log n)$ lower bound for most pair of trees. A restricted version of the nni operation, known as the tree rotation operation (discussed in Section 6), was considered in [41] and a trivial approximation algorithm with approximation ratio of 2 was given. But given two individual pair of trees, computing the nni distance between them (either for labeled or unlabeled trees) has been a long standing open question until recently when this problem was settled (for both labeled and unlabeled trees) in [7, 9].

Theorem 1 *Computing the nni distance (between two labeled or unlabeled trees) is NP-complete.*

We provide a rough sketch of the proof of Theorem 1 for labeled trees (which is the more difficult case). The proof is by a reduction from Exact Cover by 3-Sets (X3C), which is known to be NP-complete [17], 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.

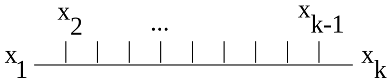

Figure 4: A linear tree with k leaves.

[9] In [29], the author reduced the Partition problem to nni by constructing a tree of i nodes for a number i, in an attempt to prove the NP-hardness of computing nni distance between unlabeled trees.

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 [6, 33] and the lower bound in [42] (see [33]) 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 *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 [6, 33, 42], 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 $\frac{1}{4} - o(1) \leq c_k \leq 1 + o(1)$ there must be infinitely many k satisfy $c_{2k} \geq c_k - \frac{\epsilon}{2}$. 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 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.

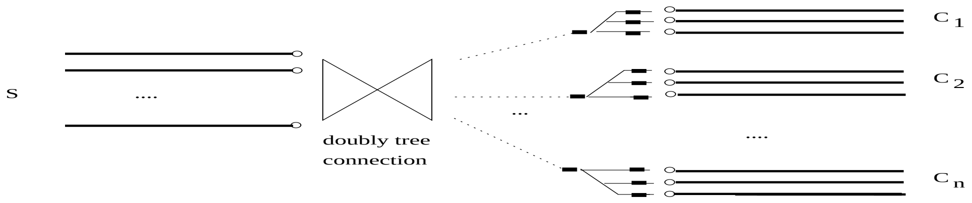

Figure 5: Structure of tree T_1

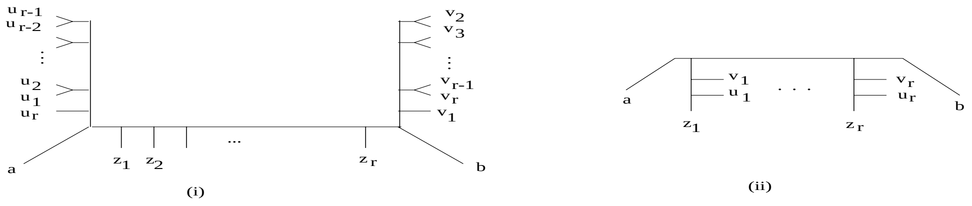

Figure 6: One-way circuit

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 1.

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

We provide an informal sketch of the proof of Lemma 3; the reader is referred to [7, 9] for more formal proofs. Assume that we have an exact cover for S. First, we show that the one-way circuit in Figure 6 behaves as was claimed. This can be seen as follows. The counterpart of the one-way circuit in T_2 is as shown in Figure 6(ii). Consider any optimal transformation of circuit (i) to (ii). 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,assuming r is odd, in total we need $2(\frac{r-3}{2} + 6\frac{r-1}{2} + 1) = 7r - 7$ 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'. 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.

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*. 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 6(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, $j = i_1, \ldots, i_q$, in the cover, we send the three sequences $S_{j_1}, S_{j_2}, S_{j_3}$ to their counterparts $S_{j,j_1}, S_{j,j_2}, S_{j,j_3}$, merge each pair and sort them together, 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 sequences $S_{i,j}$ 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 to the right.

1. The overhead for these q groups to cross the tree connection network: $q(\log m + \log n) + O(1)$ nni moves.

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 6(ii) costs $7r - 7$ nni's. Since we select $r = m^4$ and there are in all $4n$ one-way circuits, the total cost is $28nm^4 - 28n$.

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. The total cost is therefore $O(q)$.

5. Let M be the cost for sorting a sequence $S_{i,j}$ optimally. M can be computed easily, given optimal ways to sort an x block and an y block. Observe that the k^2 delimiter sequences inserted in front of each x/y block prevent the *folding* of any sequence $S_{i,j}$ in an optimal sorting procedure, *i.e.* it will not be beneficial for two blocks on the same sequence 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 delimiter sequence. Similarly, shrinking any sequence $S_{i,j}$ does not help either. So totally we need $3nM$ nni moves to sort the $3n$ sequences defined for $C_1, \ldots C_n$.

6. The extra cost of merging each sequence S_i with its counterpart $S_{j,i}$ while sorting the latter, and splitting it out when the sorting is done. The process is as follows. We sort $S_{j,i}$ block by block from head to tail. Before processing each block, we first merge this block with the corresponding block of S_i. After sorting this pair of blocks, we split out

the sorted block of S_i, and move down to the next block of S_i, passing a delimiter path of length k^2 length. So the extra cost to sorting $S_{j,i}$ is $(k^2 + 6k)m^3 \log m$. Observe that the above process automatically reverses $S_{j,i}$ and S_i.

Conversely, suppose that S has no exact cover. Then to transform T_1 into T_2, either we have to send $q + 1$ groups 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 2 and the choice $\epsilon = 0.5$, this is not much better than sorting the two segments separately and costs at least $0.5cm^3k \log k - m^3k$ more nni moves than sorting one such segment, which is larger than m^2 for sufficiently large k and m. The second case introduces an extra cost of $(0.5cm^3k \log k \log m) - m^3k \log m - m^2$ by Lemma 2, 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 2,3,5,6 are all optimal. The bounds in items 1 and 4 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. This completes the sketch of proof of Lemma 3.

In practice, however, 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. It turns out that by using the results in [42, 33], we could improve this asymptotically to $O(n^2 \log n + n \cdot 2^{23d/2})$ time. To be precise, the following result was proved in [7, 8].

Theorem 4 *Suppose that the nni distance between T_1 and T_2 is at most d. Then, an optimal sequence of nni operations transforming T_1 into T_2 can be computed in $O(n^2 \log n + n \cdot 2^{23d/2})$ time.*

A sketch of proof of Theorem 4 is as follows. Let T_1 and T_2 be the two trees being compared. An edge $e_1 \in T_1$ is *good* if there is another edge $e_2 \in T_2$ such that e_1 and e_2 partition the leaf labels of T_1 and T_2 identically; e_1 is *bad* otherwise. It is easy to see that T_1 contains at least 1 and at most d bad edges. Moreover, assume that these bad edges form t connected components $B_1, \ldots, B_t$ $(1 \le t \le d)$. As observed in [33], for an optimal nni transformation, sometimes one or more nni operations are needed across a good internal edge of T_1. Consider the set of at most $d-1$ good edges in T_1 across which at least one nni operation is performed in an optimal nni sequence. This set of good edges form at most $d-1$ connected components in T_1. Consider any one such connected component S. Since good edges in T_1 and T_2 partition the trees in similar manner, it is very easy to see that there must be at least one connected component B_i sharing a vertex with S.

Using these observations, one can devise the algorithm shown in the next page. Figure 7 illustrates how the algorithm works. Figure 7(a) shows two bad edges α, β in T_1 (shown by thick lines) forming two connected components $(t = 2)$. In Figure 7(b) we show one choice of two subtrees containing k_1 and k_2 edges, and including the edges α and β, respectively. For each subtree, algorithm NNI-d computes all possible nni sequences such that at most 3 nni are performed across edges of each subtree.

How fast does the algorithm run? There are at most $\binom{d+t-1}{d} < 2^{5d/2}$ choices for the integers $k_1, \ldots, k_t$ (using the fact that $\binom{n}{j} \le (2.8n/j)^j$). Note that any subtree of k edges including a fixed edge can be represented by a rooted binary tree on $k+2$ nodes (the root corresponding to the middle of the fixed edge), hence there are at most $C_{k+2} = \frac{1}{k+3}\binom{2k+4}{k+2} \le 2^{2k}$ such trees. It follows that the total number of choices for the subtrees $A_1, \ldots, A_t$ (for any particular value of $k_1, \ldots, k_t$) is at most $2^{\sum_{i=1}^{t}((2k_i+1))} \le 2^{3d}$. For each tree A_i, the number of sequences of k_i nni operations to consider is at most $3^{k_i-1}2^{4k_i} < 2^{6k_i}$ by Lemma 1 of [33]. Combining everything, the number of trees we have to examine is at most $2^{5d/2} \cdot 2^{3d} \cdot 2^{6d} < 2^{23d/2}$. The set of all good edges of T_1 can be found in $O(n^2 \log n)$ time and this time bound is also sufficient to find the connected components of good edges. Using the adjacency-list representation of trees, updating a tree during a single nni operation can easily be done in $O(1)$ time, and whether two trees are isomorphic can be easily checked in $O(n)$ time. Hence, this algorithm finds an optimal nni sequence in $O(n^2 \log n + n \cdot 2^{23d/2})$ time.

3.2 Unweighted trees: Computing nni distance approximately

Since computing the nni distance is NP-hard, the next obvious question is: *can we get a good approximation of the distance?* The following result appeared in [33].

```
For every choice of integers $k_1, \ldots, k_t \geq 1$, $\sum_{i=1}^{t} k_i \leq d$ do
    For every choice of subtrees $A_1, \ldots, A_t$ of $T_1$ such that
    $A_i$ has at most $k_i$ edges and contains the component $B_i$ do
        Examine all sequence of nni transformations across edges
        of all $A_i$'s such that no more than $k_i$ nni operations
        are performed across the edges of $A_i$
Among all sequences examined, select the one of shortest length that
transforms $T_1$ to $T_2$
```

Algorithm NNI-d for the case when nni distance is bounded

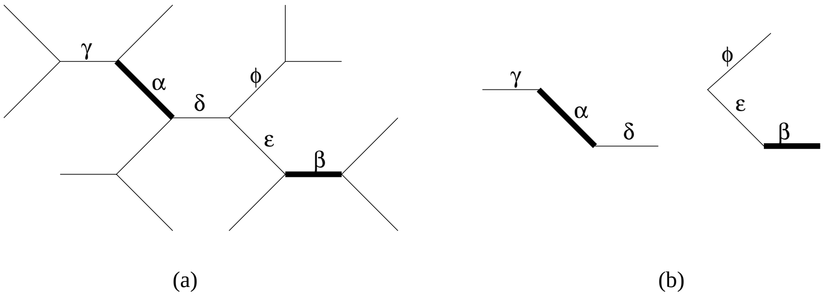

Figure 7: Illustration of how Algorithm NNI-d works ($d = 6$, $k_1 = k_2 = 3$, $t = 2$).

Theorem 5 *The nni distance can be polynomial time approximated within a factor of* $\log n + O(1)$.

Proof. Given two trees, T_0, T_1, we first identify the bad edges in T_0 with respect to T_1. These edges induce a subgraph of T_0 consisting of one or more components, each of which is a subtree of T_0. Each bad-edge component links up the same set of neighboring shared-edge-components in T_0 and T_1, but it does so in different ways.

The algorithm transforms T_0 into T_1 by transforming each non-shared edge component separately. Consider a component consisting of k non-shared edges in T_0. This links up $k + 3$ shared-edge-components, which we can consider as leaves for the purpose of linking them up differently. So we want to transform C_0 into C_1, where C_i is the $(k + 3)$-tree corresponding to the component in T_i. By the 'compression'-method of [6], the distance between C_0 and C_1 is at most $4(k + 3)\log(k + 3) + (4 - \log 3)(k + 3) - 12$.

On the other hand, it is clear that any transformation from T_0 into T_1 must use at least one nni operation on every non-shared edge.

The approximation factor of this algorithm is at most

$$\frac{\sum 4(k+3)\log(k+3)+(4-\log 3)(k+3)-12}{\sum k} \leq \frac{4n\log n+O(n)}{n-3},$$

since $\sum k$ is at most the number of internal edges, which is $n-3$. □

As is apparent from the previous two sections, the question of the computability of the nni distance measure, which we will denote by d, has generated a lot of interest. Of course, a brute force method can be employed which searches all (or a significant fraction of) trees in *exponential time and space* ([33] implemented a C program that uses $O(n)$ space to find the distance of any tree to a given one using a brute-force approach and could run it for trees up to size 11). In an attempt to improve efficiency, Waterman and Smith in [47] propose another distance measure, "closest partition" which they conjecture is actually equal to d. The closest partition distance $c(T,S)$ for trees sharing a partition is defined recursively as the sum of the two distances between the corresponding smaller parts resulting from splitting each tree into two. For trees T, S not sharing a partition it is defined as $k + c(R,S)$, where k is the minimum number of nni operations required to transform a tree T into a tree R that shares a partition with tree S. Note that the nondeterminism in choosing R makes this measure somewhat ill-defined. They base their conjecture on what [10] aptly calls a *decomposability property* (DP) of nni. Informally, DP says that if two trees can each be split at some internal edge into identical subsets of leaves, then an optimal transformation of one into the other can be found in which no nni operation affects that internal edge. This claim appears in [47] as Theorem 4. It's proof however appeals to their Theorem 3, which was shown invalid in [26] with a 6-node counterexample. Consequently, [26] concludes that the status of Theorem 4 is unresolved, and observes that Theorem 5 of [47] is a single step version of the Waterman and Smith's conjecture that $c = d$. This conjecture was shown to fail in [26] and [5] in a weak sense (for some choices that c allows), and shortly thereafter in [25] in a strong sense (for all choices in defining c). These papers also point out that computation of c appears to require exponential time as well, since there is no obvious bound on k in the definition of c. The work in [30] shows a logarithmic gap between measures c and d. Their example is a pair of trees, each on $n = 2^k$ nodes equidistant from the central internal edge. In one tree, the leaves can be drawn in normal order, while in the other, the leaves can be drawn in bit-reverse order (e.g. 0,4,2,6,1,5,2,7). For this pair of trees one can show $d = \Theta(n)$, whereas $c = \Theta(n \log n)$ (in the weak sense at least). Finally, the following result was proved in [33], serving as a counterexample to all three theorems 3, 4, and 5 of [47].

Lemma 6 *There are trees T_0, T_1 sharing a partition which is not shared by any intermediate tree on a shortest path from T_0 to T_1.*

3.3 Weighted trees: generalizing the nni distance

In this section we discuss how to generalize the nni distance between two trees T_1 and T_2 when both T_1 and T_2 are weighted. The cost of an nni operation is now the weight of the edge across which two subtrees are swapped. As mentioned before, 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 3.1) implies the NP-completeness of the weighted nni distance problem also.

The authors in [7, 9] present a polynomial time algorithm with logarithmic approximation ratio for computing the nni distance on weighted phylogenies, generalizing the logarithmic ratio approximation algorithm in [33] (discussed in Section 3.2). 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 7 *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.

4 Computing the Subtree-Transfer Distance

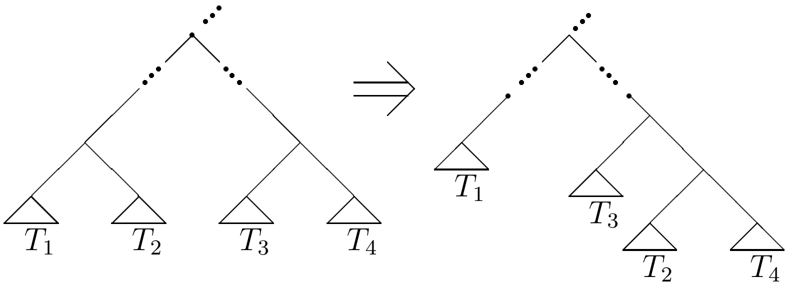

Figure 8: The operations.

In this section, we show that computing the subtree-transfer distance between two evolutionary trees is NP-hard and give an approximation algorithm with performance ratio 3. Before we prove the results, it is again convenient to reformulate the problem. Let T_1 and T_2 be two evolutionary trees on set S. An *agreement forest* of T_1 and T_2 is any forest which can be obtained from both T_1 and T_2 by cutting k edges (in each tree) for some k and applying forced contractions in each resulting component trees. Define the size of a forest as the number of components it contains. Then the *maximum agreement forest* (MAF) problem is to find an agreement forest with the *smallest* size. The following lemma shows that MAF is really equivalent to computing the subtree-transfer distance.

Lemma 8 *The size of a MAF of T_1 and T_2 is one more than their subtree-transfer distance.*

The lemma can be proven by a simple induction on the number of leaves. Intuitively, the lemma says that the transfer operations can be broken down

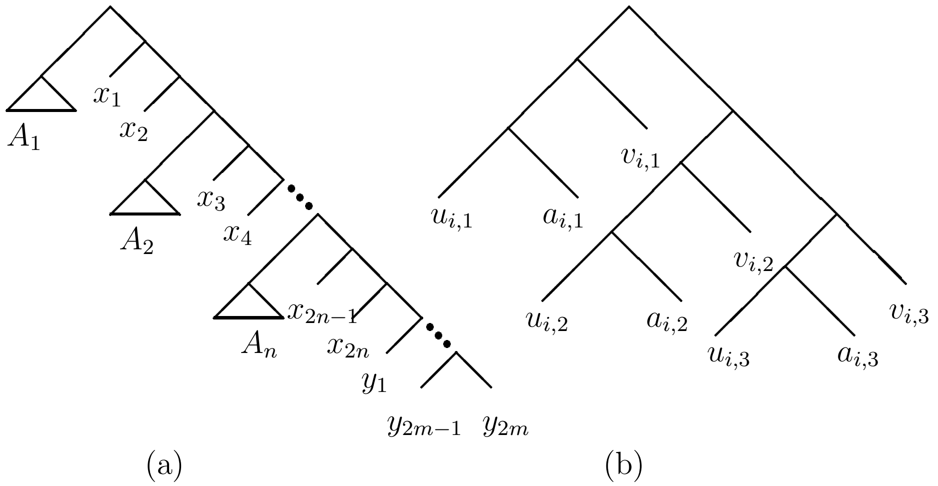

Figure 9: (a) The tree T_1. (b) The subtree A_i.

into two stages: first we cut off the subtrees to be transferred from the rest in T_1 (not worrying where to put them), then we assemble them appropriately to obtain T_2. This separation will simplify the proofs.

4.1 The NP-hardness

Theorem 9 *It is NP-hard to compute the subtree-transfer distance between two binary trees.*

Proof. (Sketch) The reduction is from Exact Cover by 3-Sets. Let $S = \{s_1, s_2, \ldots s_m\}$ be a set and $C_1, \ldots, C_n$ be an instance of this problem. Assume $m = 3q$.

The tree T_1 is formed by inserting n subtrees $A_1, \ldots, A_n$ into a chain containing $2n + 2m$ leaves $x_1, \ldots, x_{2n}, y_1, \ldots, y_{2m}$ uniformly. (See Figure 9(a).) Each A_i corresponds to $C_i = \{c_{i,1}, c_{i,2}, c_{i,3}\}$, and has 9 leaves as shown in Figure 9(b). Suppose that $c_{j,j'}$, $c_{k,k'}$ and $c_{l,l'}$ are the three occurrences of an $s_i \in S$ in C. Then in T_2, we have a subtree B_i as shown in Figure 10(a). For each C_i, we also have a subtree D_i in T_2 as shown in Figure 10(b). The subtrees are arranged as a linear chain as shown in Figure 10(c).

Note that, each adjacent pair of subtrees A_i and A_{i+1} in T_1 is separated by a chain of length 2 which also appears in T_2. Thus, to form a

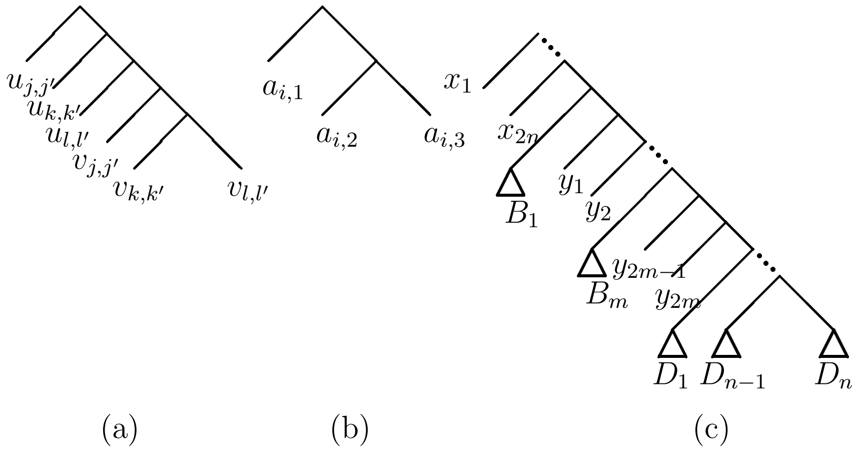

Figure 10: (a) The subtree B_i. (b) The subtree D_i. (c) The tree T_2.

MAF of T_1 and T_2, our best strategy is clearly to cut off $A_1, A_2, \ldots, A_n$ in T_1 and similarly cut off $B_1, B_2, \ldots, B_m$ in T_2. This then forces us to cut off $D_1, D_2, \ldots, D_n$ in T_2. Now in each A_i, we can either cut off the leaves $u_{i,1}, v_{i,1}, u_{i,2}, v_{i,2}, u_{i,3}, v_{i,3}$ to form a subtree containing three leaves $a_{i,1}, a_{i,2}, a_{i,3}$ (yielding $6+1=7$ components totally), or we can cut off $a_{i,1}$, $a_{i,2}$, and $a_{i,3}$. In the second case, we will be forced to also cut links between the three subtrees containing leaves $\{u_{i,1}, v_{i,1}\}$, $\{u_{i,2}, v_{i,2}\}$ and $\{u_{i,3}, v_{i,3}\}$ respectively, as the B_i's are already separated. Hence in this case the best we can hope for is $3+3=6$ components (if we can keep all three 2-leaf subtrees in the agreement forest).

It can be shown that C has an exact cover of S if and only if T_1 and T_2 have an agreement forest of size $1+6q+7(n-q) = 7n-q+1$. □

4.2 An Approximation Algorithm of Ratio 3

Our basic idea is to deal with a pair of sibling leaves a, b in the first tree T_1 at a time. If the pair a and b are siblings in the second tree T_2, we replace this pair with a new leaf labeled by (a, b) in both trees. Otherwise, we will cut T_2 until a and b become siblings or separated. Eventually both trees will be cut into the same forest. Five cases need be considered. Figure 11 illustrate the first four cases. The last case (Case (v)) is that a and b are

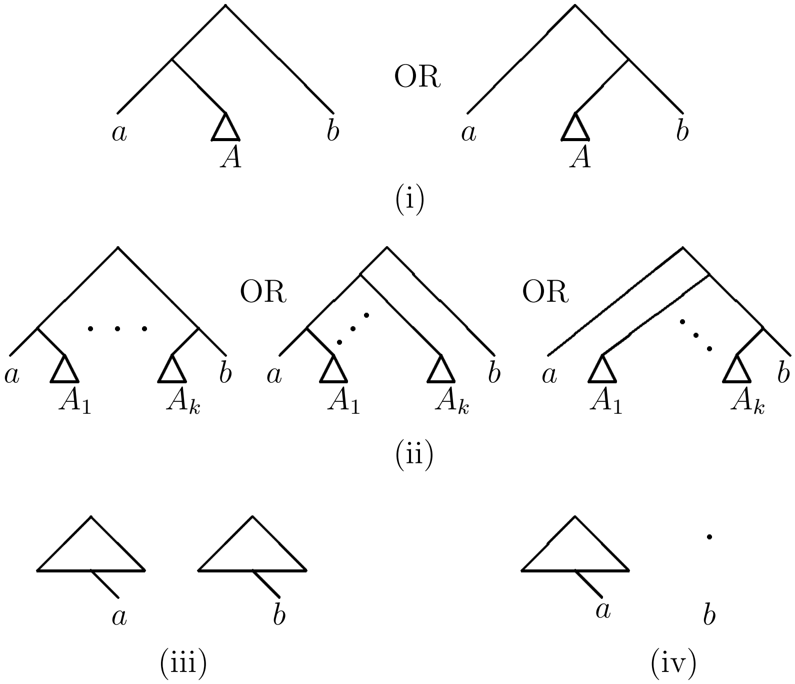

Figure 11: The first four cases of a and b in T_2.

also siblings in T_2.

The approximation algorithm is given in Figure 12. The variable N records the number of components (or the number of cuts plus 1).

Theorem 10 *The approximation ratio of the algorithm in Figure 12 is* 3*, i.e., it always produces an agreement forest of size at most three times the size of a MAF for T_1 and T_2.*

The NP-hardness proof can be easily strengthened to work for MAX SNP-hardness. Thus there is no hope for a polynomial-time approximation scheme for this problem. Moreover, the small distance exact algorithm described in section 3 for nni also works here.

```
Input: $T_1$ and $T_2$.
0. $N := 1$;
1. For a pair of sibling leaves $a, b$ in $T_1$,
  consider how they appear in $T_2$ and cut the trees:
  *Case (i):* Cut off the middle subtree $A$ in $T_2$; $N := N + 1$;
  *Case (ii):* Cut off $a$ and $b$ in both $T_1$ and $T_2$; $N := N + 2$;
  *Case (iii):* Cut off $a$ and $b$ in both $T_1$ and $T_2$; $N := N + 2$;
  *Case (iv):* Cut off $b$ in $T_1$;
  *Case (v):* Replace this pair with a new leaf labeled $(a, b)$ in both $T_1$ and $T_2$;
2. If some component in the forest for $T_1$ has size larger than 1, repeat Step 1.
Output: The forest and $N$.
```

Figure 12: The approximation algorithm of ratio 3.

5 Linear-Cost Subtree-Transfer Distance on Weighted Phylogenies

In this section we investigate the linear-cost subtree-transfer model on weighted phylogenies.

5.1 An NP-hardness Result

It is open whether the linear-cost subtree-transfer problem is NP-hard for weighted phylogenies. However, we can show that the problem is NP-hard for weighted trees with non-uniquely labeled leaves.

Theorem 11 *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-hard.*

Proof. Our proof is by a reduction from the following *Exact Cover by 3-Sets* (X3C) problem.

INSTANCE: $S = \{s_1, \ldots, s_m\}$, where $m = 3q$, and $C_1, \ldots, C_n$, where $C_i = \{s_{i_1}, s_{i_2}, s_{i_3}\} \subseteq S$.

QUESTION: Are there q disjoint sets $C_{i_1}, \ldots, C_{i_q}$ such that $\cup_{j=1}^{q} C_{i_j} = S$?

X3C is known to be NP-complete [17]. Given an instance of the X3C problem, we will construct two trees T_1 and T_2 with leaf labels (not necessarily unique) as shown in Figure 5.1, such that transforming from T_1 into T_2 requires subtree-transfers of total cost exactly 1 iff an exact cover of S exists.

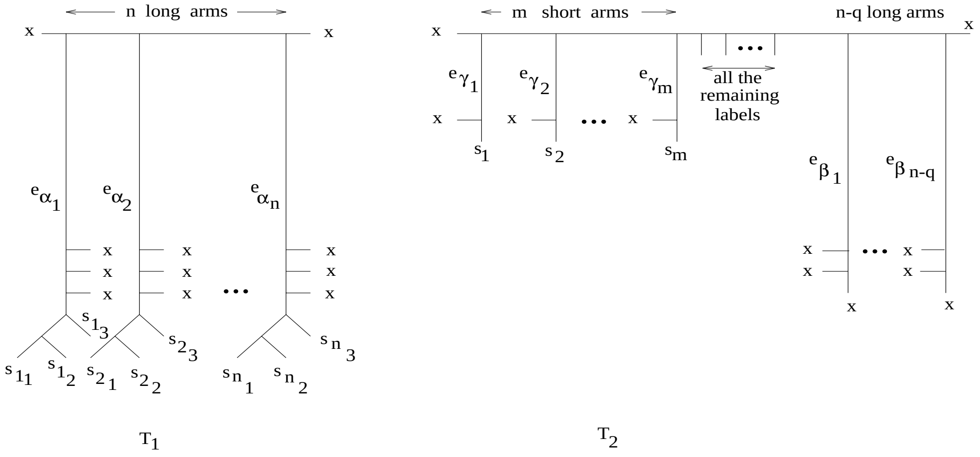

Figure 13: Trees T_1 and T_2 used in the proof of Theorem 11. The leaf labels are shown beside the corresponding leaves. The notations for some of the internal edges are shown beside the corresponding edges. The edge weights are as follows: $w(e_{\alpha_1}) = w(e_{\alpha_2}) = \cdots = w(e_{\alpha_n}) = w(e_{\beta_1}) = w(e_{\beta_2}) = \cdots = w(e_{\beta_{n-q}}) = \frac{1}{n}$, $w(e_{\gamma_1}) = w(e_{\gamma_2}) = \cdots = w(e_{\gamma_m}) = \frac{1}{3n}$, and all other edges have zero weights.

T_1 has n long arms, $\alpha_1, \ldots, \alpha_n$. T_2 has $n-q$ long arms, $\beta_1, \ldots, \beta_{n-q}$, and m short arms, $\gamma_1, \ldots, \gamma_m$. Each long (resp. short) arm consists of an edge of weight $\frac{1}{n}$ (resp. $\frac{1}{3n}$), with three leaves (resp. one leaf) labeled by the same label x ($x \notin S$), connected to it as shown in Figure 5.1. For notational convenience, let e_{α_i} (resp. e_{β_i}, e_{γ_i}) denote the edge of non-zero weight in the long arm α_i (resp. in the long arm β_i, in the short arm γ_i). In T_1, at the bottom of the i^{th} long arm α_i, we attach a subtree t_i consisting of three leaves, as shown in Figure 5.1, labeled by the three elements s_{i_1}, s_{i_2} and s_{i_3} of C_i. At the bottom of each long arm of T_2, there are no additional subtrees attached. The labeling of the remaining leaves of T_2 is as follows:

- At the bottom of the i^{th} short arm γ_i, we attach a leaf labeled by s_i.
- The remaining $3n - m$ leaf labels (each leaf label is an element of S) are associated (in any order) with the $3n - m$ leaves in the middle of T_2 between the long and the short arms.

Note that the trees T_1 and T_2 are not uniquely labeled. The following claim proves the correctness of the NP-hardness reduction.

$$D_{st}(T_1, T_2) = 1 \text{ iff there is a solution of the X3C problem.}$$

A proof of the above claim can be found in [8]. □

5.2 An Approximation Algorithm

In this section, we present an approximation algorithm for computing the linear-cost subtree-transfer distance on weighted trees. First, we introduce some notations and a lower bound on the subtree-transfer distance which will be useful in subsequent proofs. 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 $W_{ext}(T) = w(E_{ext}(T))$. Partition $E_{ext}(T_1)$ into three subsets as follows:

$$\begin{aligned}
E_{ext,T_1>T_2}(T_1) &= \{e_{T_1}(a) \mid w(e_{T_1}(a)) > w(e_{T_2}(a))\} \\
E_{ext,T_1=T_2}(T_1) &= \{e_{T_1}(a) \mid w(e_{T_1}(a)) = w(e_{T_2}(a))\} \\
E_{ext,T_1<T_2}(T_1) &= \{e_{T_1}(a) \mid w(e_{T_1}(a)) < w(e_{T_2}(a))\} \\
W_{ext,T_1>T_2}(T_1) &= \sum_{e_{T_1}(a) \in E_{ext,T_1>T_2}(T_1)} w(e_{T_1}(a)) - w(e_{T_2}(a))
\end{aligned}$$

Similarly, $E_{ext}(T_2)$ can be partitioned into: $E_{ext,T_1>T_2}(T_2)$, $E_{ext,T_1=T_2}(T_2)$, and $E_{ext,T_1<T_2}(T_2)$. $W_{ext,T_1<T_2}(T_2)$ is defined analogously. The following lemma is easy to prove.

Lemma 12 $W_{int}(T_1) + W_{ext,T_1>T_2}(T_1) = W_{int}(T_2) + W_{ext,T_1<T_2}(T_2)$.

We next define the notion of good edge pairs as follows:

Definition 13 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. One of the following two conditions holds:

   (a) $w(E(T_1')) \leq w(E(T_2')) < w(E(T_1')) + w(e_1)$; or

   (b) $w(E(T_2')) \leq w(E(T_1')) < w(E(T_2')) + w(e_2)$.

Lemma 14 *If T_1 and T_2 share no good edge pairs, then:*
(1) $D_{st}(T_1, T_2) \geq W_{int}(T_1) + W_{ext,T_1>T_2}(T_1)$;
(2) $D_{st}(T_1, T_2) \geq W_{int}(T_2) + W_{ext,T_1<T_2}(T_2)$.

Proof. We only prove (1). The proof of (2) follows from (1) and Lemma 12. For each edge $e \in E(T_1)$, we determine the minimum portion of e over which some subtrees of T_1 must be transferred in order to transform T_1 to T_2. First, consider an edge $e_1 \in E_{int}(T_1)$. By the assumption of the lemma, there is no edge e_2 in T_2 such that e_1 and e_2 are a good pair. There are two cases:

Case 1. The partition of $L(T_1)$ induced by e_1 is different from the partition of $L(T_2)$ induced by any edge in T_2. Then, in order to transform T_1 to T_2, some leaf nodes of T_1 must be transferred across the entire length of e_1.

Case 2. The partition of $L(T_1)$ induced by e_1 is the same as the partition of $L(T_2)$ induced by an edge e_2 in 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, where $L(T_1') = L(T_2')$ and $L(T_1'') = L(T_2'')$.

Case 2.1. $w(E(T_2')) \geq w(E(T_1')) + w(e_1)$. In this case, in order to transform T_1' to T_2', some subtree in T_1' must be transferred across entire length of e_1.

Case 2.2. $w(E(T_1')) \geq w(E(T_2')) + w(e_2)$. This implies: $w(E(T_1'')) + w(e_1) \leq w(E(T_2''))$. In order to transform T_1'' to T_2'', some subtree in T_1'' must be transferred across the entire length of e_1.

In either case, some subtree of T_1 must be transferred across the entire length of e_1 with cost $w(e_1)$.

Next consider an edge $e_{T_1}(a) \in E_{ext,T_1>T_2}(T_1)$. In order to transform $e_{T_1}(a)$ to $e_{T_2}(a)$, a subtree of T_1 must be transferred across a portion of $e_{T_1}(a)$ of length $w(e_{T_1}(a)) - w(e_{T_2}(a))$. Thus:

$$\begin{aligned} D_{st}(T_1, T_2) &\geq \textstyle\sum_{e \in E_{int}(T_1)} w(e) + \sum_{e \in E_{ext,T_1>T_2}(T_1)} [w(e_{T_1}(a)) - w(e_{T_2}(a))] \\ &= W_{int}(T_1) + W_{ext,T_1>T_2}(T_1) \end{aligned}$$

□

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 re-connect 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 our subsequent descriptions.

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).

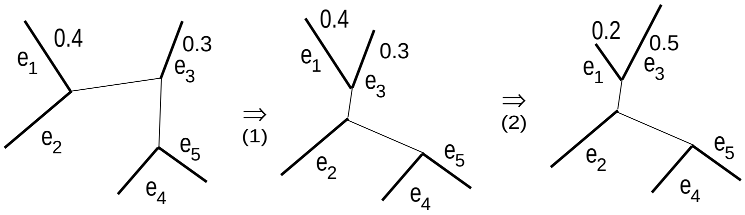

Figure 14: The operation $move(e_1, 0.2, e_3)$. (1) e_2, e_4, e_5 are assembled into a tree S; (2) S is moved along e_1 by a length of 0.2.

Figure 5.2 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.

In the rest of this section, we prove the following theorem.

Theorem 15 *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.*

First, we describe the algorithm DST which approximates $D_{st}(T_1, T_2)$ to within a factor of 2 for the special case when T_1 and T_2 do not have any good edge pairs. Then we will show how to apply the algorithm to the general case.

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). T_1' (resp. T_2') is initialized to be 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.

Lemma 16 *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.*

Proof. We analyze the cost and running time of each step of the algorithm. We use the adjacency-list representation of a tree. Steps 0 and 4 incur no costs and can easily be implemented in $O(n)$ time. During Steps 1, 2 and 3.1, we only transfer subtrees across internal edges of T_1 and T_2. Over any portion of such an edge e, at most one subtree-transfer operation occurs. So the total cost of these steps is bounded above by $W_{int}(T_1) + W_{int}(T_2)$. Moreover, it is easy to see that at most $O(n)$ moves are performed during Steps 1,2, and 3.1, and since each move operation can be implemented in $O(n)$ time, the total time for all these steps is at most $O(n^2)$.

Next, consider Step 3.2. Before the repeat loop is entered, for any $c \in C$, we have:

- $w(e_{T'_1}(c)) = w(e_{T_1}(c))$. (This is because no additional weight is moved to the edge $e_{T'_1}(c)$ during Steps 1 and 2).
- $w(e_{T'_2}(c)) \geq w(e_{T_2}(c))$.

During Step 3.2, we only transfer subtrees across the edges $e_{T'_1}(c)$ for $c \in C$. Fix such an edge. Note that any portion of $e_{T'_1}(c)$ is traversed at most once during Step 3.2. Once the length of $e_{T'_1}(c)$ is reduced to $w(e_{T'_2}(c))$, c is removed from C. So the portion of $e_{T'_1}(c)$ traversed during Step 3.2 is $w(e_{T'_1}(c)) - w(e_{T'_2}(c)) = w(e_{T_1}(c)) - w(e_{T'_2}(c)) \leq w(e_{T_1}(c)) - w(e_{T_2}(c))$. So the total cost of Step 3.2 is at most $\sum_{c \in C}[w(e_{T'_1}(c)) - w(e_{T'_2}(c))] \leq \sum_{c \in C}[w(e_{T_1}(c)) - w(e_{T_2}(c))] \leq W_{ext,T_1>T_2}(T_1)$. Also, we perform at most $O(n)$ move operations during Step 3.2, and hence this step can also be implemented in $O(n^2)$ time.

Thus the total cost of the algorithm is bounded above by $W_{int}(T_1) + W_{int}(T_2) + W_{ext,T_1>T_2}(T_1)$, which is at most $2D_{st}(T_1, T_2)$ by Lemma 14. □

Next, we show how to apply algorithm DST to achieve an approximation ratio of 2 when T_1 and T_2 may share some good edge pairs. We concentrate on the algorithm and omit implementation details. 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 16. 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 15). Note that the weights of all new edges are non-negative.

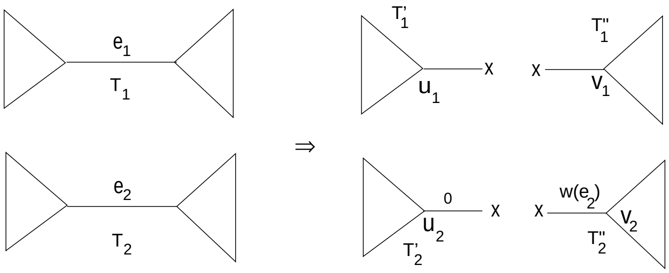

Figure 15: Cut each of T_1 and T_2 into two smaller trees.

Now we have: $L(T_1') = L(T_2')$ 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 15.

6 The Rotation Distance

6.1 Rotation and its equivalences

A *rotation* is an operation that changes one rooted binary tree into another with the same size. Figure 16 shows the general rotation rule. Note that the rotation is an invertible operation. If a tree T is changed into T' by a rotation, then T' can be changed back into T by another rotation. In a rooted binary tree of size n, there are $n-1$ possible rotations, each corresponding a non-root node.

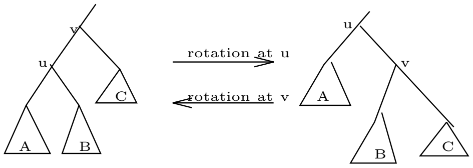

Figure 16: The definition of rotation.

A *symmetric order* traversal of a rooted tree visits all of the nodes exactly once. The order can be described recursively as follows: for a node in the tree, traverse its left subtree(if there is), visit the node itself, and then traverse its right subtree(if there is). A rotation maintains the symmetric order of the nodes.

The rotation on binary trees can be formulated with respect to different systems of combinatorial objects and their transformations. The diagonal-flip operation in triangulations is perhaps more intuitive and so supplies more insight.

Consider the standard convex $(n+2)$-gon. We choose an edge of the polygon as a distinguished edge, called “root edge”, and label its ends as 0 and $n+1$. We also label the other n vertices from 1 to n counterclockwise. Any triangulation of the $(n+2)$-gon has n triangles and $n-1$ diagonals. ¿From a triangulation of the $(n+2)$-gon, we derive a binary tree of size n by assigning a node for each triangle and connecting two nodes if the corresponding triangles sharing a common diagonal. The root of the tree corresponds to the triangle containing the root edge. It is not difficult to see that the ith node of the binary tree in symmetric order corresponds to the triangle with vertices i, j and k such that $j < i < k$. In this way, we obtain a 1-1 correspondence between n-node binary trees and triangulations of the $(n+2)$-gon as illustrated in Figure 17.

A *diagonal-flip* is an operation that transforms one triangulation of a convex polygon into another as showed in Figure 18. A diagonal inside the polygon is removed, creating a quadrilateral. Then the opposite diagonal of this quadrilateral is inserted in place of the one removed, restoring a

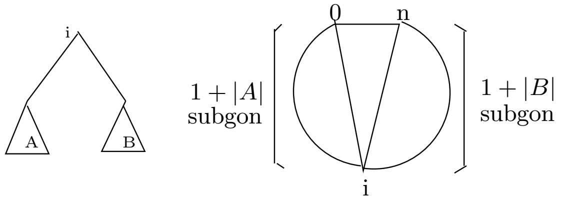

Figure 17: A triangulation and its corresponding n-node rooted binary tree.

triangulation of the polygon. It is not difficult to see that diagonal-flips in a triangulation correspond one-to-one to rotations in the corresponding binary tree.

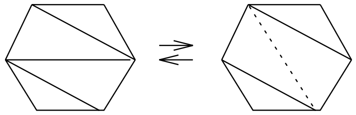

Figure 18: A diagonal flip in a triangulation of the hexagon.

Given a triangulation π of a polygon, we define the *internal degree* of a vertex v as the number of diagonals adjacent to v, denoted by $id(v)$. Now let us see how $id(v)$ is reflected in the corresponding binary tree. In a rooted binary tree T, the *left (resp. right) path* is a maximal sequence of nodes that form a path starting at the root all of whose edges go in left (resp. right) direction. For a node $v \in T$, the left and right subtree rooted at v are denoted by LT_v and RT_v respectively. Recall that all non-leaf nodes are *internal nodes* in a tree. The following result is of interest itself and has not appeared in literature to the best of the authors knowledge.

Theorem 17 *([34]) Suppose that the $(n+2)$-gon P is oriented by labeling its vertices from 0 to $n+1$ and $(0, n+1)$ is the root edge. Let π be a triangulation of P and T be the corresponding n-node rooted binary tree. Then*

(1) The internal degree $id(0)$ of vertex 0 in P equals the number of internal nodes on the left path of T;

(2) The internal degree $id(n+1)$ of vertex $n+1$ in P equals the number of internal nodes on the right path of T;

(3) The internal degree $id(i)$ of vertex $i \in P$ $(0 < i < n+1)$ equals the number of subtrees at node i, which is at most 2, plus the number of internal nodes on the right path of the left subtree LT_i and the number of internal nodes on the left path of the right subtree RT_i.

Other interesting relationship between a triangulation of a convex polygon and its corresponding rooted binary tree can be found in a nice survey article [1].

6.2 Upper and lower bounds for the rotation distance

Any rooted binary tree of size n can be converted into any other with the same size by performing an appropriate sequence of rotations. Therefore, we can define the *rotation distance* between two trees as the minimum number of rotations required to convert one tree into the other. Let $rt(T_1, T_2)$ to denote the rotation distance between two trees T_1 and T_2. Define

$$rt(n) = \max\{rt(T_1, T_2) \mid |T_1| = |T_2| = n\}.$$

Similarly, we can define the *diagonal flip* distance between two triangulations of the n-gon and denote the maximum distance between any pair of such triangulations by $fd(n)$. Obviously, $rd(n) = fd(n+2)$.

Culik and Wood showed that $rd(n) \leq 2n - 2$([6]). Sleator, Tarjan and Thurston improved this bound to $2n-6$ and showed that the bound is tight for all sufficiently large n using hyperbolic geometry.

Theorem 18 *([41]) $rd(n) = fd(n+2) \leq 2n - 6$ for all $n > 10$. Furthermore, the equality holds for all sufficiently large n.*

The exact values of $rd(n)$ for $n \leq 16$ are listed below[41].

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
rd(n)	0	1	2	4	5	7	9	11	12	15	16	18	20	22	24	26

However, little is known about the lower bound for the rotation distance $rd(T_1, T_2)$ of two given trees T_1 and T_2. The following two theorems are the only known lower bounds, which is presented in term of the diagonal flip distance for simplicity. The first one is a variant of lemma 3 in [41].

Theorem 19 *Let π_1 and π_2 be two triangulations of the n-gon. If π_1 and π_2 have k different diagonals, then $fd(\pi_1, \pi_2) \geq k$.*

Let π_1 and π_2 be two triangulations of the n-gon. Consider a sequence Π of diagonal-flips that transforms π_1 into π_2. A diagonal-flip $(ab, cd) \in \Pi$ is *auxiliary* if $cd \notin \pi_2$. We also say that the flip (ab, cd) *touches* the vertices a, b, c, d. Let $A(\Pi)$ denote the set of all auxiliary diagonal-flips in S. Let $|\Pi|$ denote the number of flips in Π. Then

$$|\Pi| \geq |A(\Pi)| + n - 3. \tag{1}$$

Finally, a triangle of a triangulation is said to be *internal* if it contains three diagonals of the triangulation.

Theorem 20 *([34]) Let π_1 and π_2 be two triangulations of a convex polygon and let v be a vertex of the polygon. Suppose that the following conditions are satisfied:*

(a) v is an end of at least two diagonals in π_2,

(b) v is not a vertex of any internal triangles in π_1 or π_2,

(c) v is not connected by a π_2-diagonal to any vertices of internal triangles in π_2, and

(d) flipping any π_1-diagonal adjacent to v does not create a π_2-diagonal.

Then, there is at least one auxiliary diagonal touching v in any sequence Π of diagonal-flips that converts π_1 into π_2.

As showed in the next subsection, Theorem 20 is useful for estimating the diagonal flip distance between two triangulations.

6.3 Approximating the rotation and diagonal flip distances

Since the rotation and diagonal distance are equivalent, we just state results in term of the diagonal flip distance.

Given two triangulations π_1 and π_2 with k different diagonals. Since every different π_1-diagonal has to be flipped, any diagonal-flip transformation from one to another contains at least k flips. On the other hand, by Theorem 18, $2k$ flips are enough to transform π_1 into π_2. This implies an approximation with ratio 2.

Theorem 21 *The diagonal flip distance can be approximated with ratio 2 in polynomial time.*

However, it is very hard to develop a polynomial approximation algorithm with constant ratio < 2 for the distance. In what follows, we prove a slightly better approximation.

Theorem 22 *([34]) There is a polynomial approximation algorithm that, on the input of two triangulations π_1 and π_2 of the n-gon, output a diagonal flip transformation of length at most $(2 - \frac{2}{4(d-1)(d+6)+1}) fd(\pi_1, \pi_2)$, where d is the maximum number of diagonals adjacent a vertex in one of the given triangulations.*

Proof. Let e be a diagonal in π_1 or π_2. The diagonal e is said to be *isolated* if there is only one diagonal (in the other triangulation) crossing e. Given two triangulations of the n-gon. They may have some diagonals in common in general. All the common diagonals divide the rest of diagonals into disjoint subclasses. Each disjoint subclass together with common diagonals around it is called a *cell.* A desired algorithm is presented in Table 1. Obviously, the algorithm takes polynomial time. We analyze its approximation ratio as follows.

Without loss of generality, we assume that π_1 and π_2 do not have common diagonals. Suppose that the Do loop runs m_1 times for isolated diagonals. Then, after the Do loop, π_1 and π_2 have been transformed into triangulations π_1' and π_2' which have m diagonals in common. Without loss of generality, we may assume that different diagonals in π_1' and π_2' forms two triangulations of a convex $(n-m)$-gon. Note that $fd(\pi_1, \pi_2) = m + fd(\pi_1', \pi_2')$.

A vertex $v \in P$ is *pure* with respect to π_i', if it is only an end of π_i'-diagonals. Let V_1 and V_2 denote the sets of pure vertices with respect to π_1' and π_2' respectively. We first prove that

$$fd(\pi_1', \pi_2') \geq (n-m) - 3 + |V_1|/4.$$

Consider a shortest sequence S of diagonal flips that transforms π_1' into π_2'. Since there are no isolated edges in both π_1' and π_2', each vertex in V_1 is an end of at least two π_1-diagonals. By Theorem 20, for each node in V_1, there is at least one auxiliary flip touching it. Since each auxiliary flip can touch at most 4 vertices, there are at most $|V_1|/4$ auxiliary flips in S. Hence, by Inequality (1), $fd(\pi_1', \pi_2') \geq (n-m) - 3 + |V_1|/4$.

```
Input: Two triangulations $\pi_1$ and $\pi_2$;

Do until the following 'if' conditions fails
    if there are isolated diagonals then
        pick such an edge $e$;
        let $e'$ be the unique diagonal that intersects $e$;
        if $e' \in \pi_1$ then $\pi_1 := \pi_1 + e - e'$ else $\pi_2 := \pi_2 + e - e'$;
Enddo

Let the resulting polygon triangulations have $k$ cells $P_i (i \leq k)$, and let
$\pi_j|_{P_i}$ denote the restriction of $\pi_j$ on $P_i$ for $j = 1, 2$ and $i \leq k$; assume $P_i$
has $n_i$ vertices.

For each cell $P_i$
        pick a node $v$;
        transform $\pi_1|_{P_i}$ into the unique triangulation $\pi$ all of whose
            diagonals have one end at $v$ using at most $n_i$ steps;
        transform $\pi$ into $\pi_2|_{P_i}$ reversely.
Endfor
```

Table 1: *Transformation algorithm.*

Similarly, by considering pure vertices with respect to π_2, we are able to prove that $fd(\pi_1', \pi_2') \geq (n - m) - 3 + |V_2|/4$. Combining these two bounds together, we obtain that

$$fd(\pi_1, \pi_2) \geq (n - m) - 3 + \frac{|V_1| + |V_2|}{8}. \qquad (2)$$

On the other hand, one can prove that there are at least $\frac{n-m}{d-1} - 3|V_1| - (d+2)|V_2|$ vertices satisfying the conditions in Theorem 20. Thus, by Inequality (1), $fd(\pi_1', \pi_2') \geq (n - m) - 3 + \frac{n-m}{4(d-1)} - \frac{3|V_1|}{4} - \frac{(d+2)|V_2|}{4}$. Similarly, we have $fd(\pi_1', \pi_2') \geq (n - m) - 3 + \frac{n-m}{4(d-1)} - \frac{3|V_2|}{4} - \frac{(d+2)|V_1|}{4}$. Combining these two inequalities together, we have that

$$fd(\pi_1', \pi_2') \geq (n - m) - 3 + \frac{n - m}{4(d - 1)} - \frac{(d + 5)(|V_1| + |V_2|)}{8}. \qquad (3)$$

By (2) and (3), $fd(\pi_1, \pi_2) = m + fd(\pi_1', \pi_2') \geq m + (n - m)(1 + \frac{1}{4(d-1)(d+6)}) - 3$.

The algorithm transforms π'_1 to π'_2 using at most $M = 2(n-m)+m = 2n-m$ flips, which is less than $(2 - \frac{2}{4(d-1)(d+6)+1})fd(\pi_1, \pi_2)$. This finishes the proof. □

Furthermore, [34] also presented a polynomial approximation algorithm with ratio 1.97 for the diagonal flip transformation between two triangulations without internal triangle. Such a triangulation corresponds to a rooted binary tree without degree 3 internal nodes.

6.4 Miscellaneous remarks

The diagonal flip operation was early studied by Wagner [45] in the context of arbitrary triangulated planar graphs and by Dewdney [11] in the case of graphs of genus one. They showed that any such graph can be transformed to any other by diagonal flips. However, they did not try to accurately estimate how many flips are necessary. After [41] was published, the rotation and diagonal flip operations have been studied in several aspects. Pallo [36] proposed a heuristic search algorithm to compute the rotation distance between two given trees. Hurtado, Noy and Urrutia [24] studied diagonal flips in arbitrary polygons. Guibas and Hershberger [18] abstracted polygon morphing as a sequence of rotations on weighted binary trees and Hershberger and Suri [23] proved that a weighted rooted binary tree can be converted into any other in at most $O(n \log n)$ rotations.

7 Open Questions

In this section, we list some open questions concerning the nni and subtree-transfer distances.

1. Give a constant ratio approximation algorithm for the nni distance on unweighted evolutionary trees or prove that the ratio $O(\log n)$ is the best possible.

2. Is the linear-cost subtree-transfer distance NP-hard to compute on weighted evolutionary trees?

3. Can we improve the approximation ratios for subtree-transfer distance on unweighted or weighted evolutionary trees?

4. Are there simple approximation algorithms for the rotation distance with nontrivial ratios?

References

[1] D. Aldous, Triangulating the circle, at random. *Amer Math. Monthly*, 89, pp. 223-234, 1994.

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

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

[4] C.H. Bennett, P. Gács, M. Li, P. Vitányi, and W. Zurek, Information Distance, to appear in *IEEE Trans. Inform. Theory.*

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

[6] K. Culik II and D. Wood, A note on some tree similarity measures, *Inform. Proc. Let.*, 15, pp. 39-42, 1982.

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

[8] B. DasGupta, X. He, T. Jiang, M. Li, and J. Tromp, On the linear-cost subtree-transfer distance, *Algorithmica*, submitted, 1997.

[9] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang, On computing the nearest neighbor interchange distance, *Preprint*, 1997.

[10] W. H. E. Day, Properties of the nearest neighbor interchange metric for trees of small size, *Journal of Theoretical Biology*, 101, pp. 275-288, 1983.

[11] A. K. Dewdney, Wagner's theorem for torus graphs, *Discrete Math.*, 4, pp. 139-149, 1973.

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

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

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

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

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

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

[18] L. Guibas and J. Hershberger, Morphing simple polygons, *Proceeding of the ACM 10th Annual Sym. of Comput. Geometry,* pp. 267-276, 1994.

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

[20] J. Hein, A heuristic method to reconstruct the history of sequences subject to recombination, *J. Mol. Evol.*, 36, pp. 396-405, 1993.

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

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

[23] J. Hershberger and S. Suri, Morphing binary trees. *Proceeding of the ACM-SIAM 6th Annual Symposium of Discrete Algorithms*, pp. 396-404, 1995.

[24] F. Hurtado, M. Noy, and J. Urrutia, Flipping edges in triangulations, *Proc. of the ACM 12th Annual Sym. of Comput. Geometry,* pp. 214-223, 1996.

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

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

[27] J. Kececioglu and D. Gusfield, Reconstructing a history of recombinations from a set of sequences, *Proc. 5th Annual ACM-SIAM Symp. Discrete Algorithms,* 1994.

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

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

[30] 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.

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

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

[33] M. Li, J. Tromp, and L.X. Zhang, On the nearest neighbor interchange distance between evolutionary trees, *Journal of Theoretical Biology*, 182, pp. 463-467, 1996.

[34] M. Li and L. Zhang, Better Approximation of Diagonal-Flip Transformation and Rotation Transformation, Manuscript, 1997.

[35] 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, pp. 423-457, 1973.

[36] J. Pallo, On rotation distance in the lattice of binary trees, *Infor. Proc. Letters*, 25, pp. 369-373, 1987.

[37] D. F. Robinson, Comparison of labeled trees with valency three, *Journal of Combinatorial Theory, Series B,* 11, pp. 105-119, 1971.

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

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

[40] D. Sankoff and J. Kruskal (Eds), *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison,* Addison Wesley, Reading Mass., 1983.

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

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

[43] K.C. Tai, The tree-to-tree correction problem, *J. ACM*, 26, pp. 422-433, 1979.

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

[45] K. Wagner, Bemerkungen zum vierfarbenproblem, *J. Deutschen Math.-Verin.*, 46, pp. 26-32, 1936.

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

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

[48] K. Zhang and D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, *SIAM J. Comput.* 18, pp. 1245-1262, 1989.

[49] K. Zhang, J. Wang and D. Sasha, On the editing distance between undirected acyclic graphs, International J. of Foundations of Computer Science 7 (13), March 1996.