Seeded Tree Alignment and Planar Tanglegram Layout

Antoni Lozano[1], Ron Y. Pinter[2], Oleg Rokhlenko[2], Gabriel Valiente[3], and Michal Ziv-Ukelson[4]

[1] Logic and Programming Research Group, Technical University of Catalonia, E-08034 Barcelona, Spain, `antoni@lsi.upc.edu`

[2] Department of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel. `pinter@cs.technion.ac.il`, `olegro@cs.technion.ac.il`

[3] Algorithms, Bioinformatics, Complexity and Formal Methods Research Group, Technical University of Catalonia, E-08034 Barcelona, Spain, `valiente@lsi.upc.edu`

[4] School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. `michaluz@post.tau.ac.il`

Abstract. The optimal transformation of one tree into another by means of elementary edit operations is an important algorithmic problem that has several interesting applications to computational biology. Here we introduce a constrained form of this problem in which a partial mapping of a set of nodes (the "seeds") in one tree to a corresponding set of nodes in the other tree is given, and present efficient algorithms for both ordered and unordered trees. Whereas ordered tree matching based on seeded nodes has applications in pattern matching of RNA structures, unordered tree matching based on seeded nodes has applications in co-speciation and phylogeny reconciliation. The latter involves the solution of the planar tanglegram layout problem, for which a polynomial-time algorithm is given here.

1 Introduction

Matching and aligning trees is a recurrent problem in computational biology. Two prominent applications are the comparison of phylogenetic trees [2, 3, 15, 18, 22, 24, 26] and the comparison of RNA structures [10, 11, 23, 28]. The specific problems defined and addressed in this paper are motivated by applications where densely seeded local tree alignments are sought.

In what follows, we first describe an example motivated by evolutionary studies of RNase P RNAs and their target tRNAs; it is interesting as it demonstrates the need for seeded tree alignments for both ordered and unordered trees. The basic formalism is given in Section 2. Section 3 describes a general framework and the corresponding analysis for seeded tree alignment. Finally, in Section 4, an algorithm is presented which computes a planar layout for two unordered seeded trees, if such exists.

1.1 Seeded trees based on RNase P structure comparison

Ribonuclease P is the endoribonuclease responsible for the 5' maturation of tRNA precursors [7]. RNase P is a ribonucleoprotein in all organisms, but is best understood in Bacteria, in which the RNA component of the enzyme is by itself catalytically proficient *in vitro* (it is a ribozyme). The structure of bacterial RNase P RNA has been studied in detail, primarily using comparative methods [16, 9, 6, 27]. Bacterial RNase P RNAs share a common core, and synthetic minimal RNase P RNAs consisting only of these core sequences and structures are catalytically proficient. Structural variation in RNase P RNA is predominated by variation in the presence or absence of helical elements and in variation of the size of the distal regions of these helices. However, there is additional variation in the form of small differences in the lengths of helices, loops and joining regions. In terms of RNA secondary structure tree alignment, this means that the operations applied in transforming one tree to another consist of subtree deletions and insertions as well as homeomorphic node insertions and deletions in ordered rooted trees (see Fig. 1).

Recently, sequences encoding RNase P RNAs of various genomes have been determined (see the RNase P database, http://www.mbio.ncsu.edu/RNaseP/). This broad sampling of RNase P RNAs allows some phylogenetic refinement of the secondary structure, and reveals patterns in the evolutionary variation of

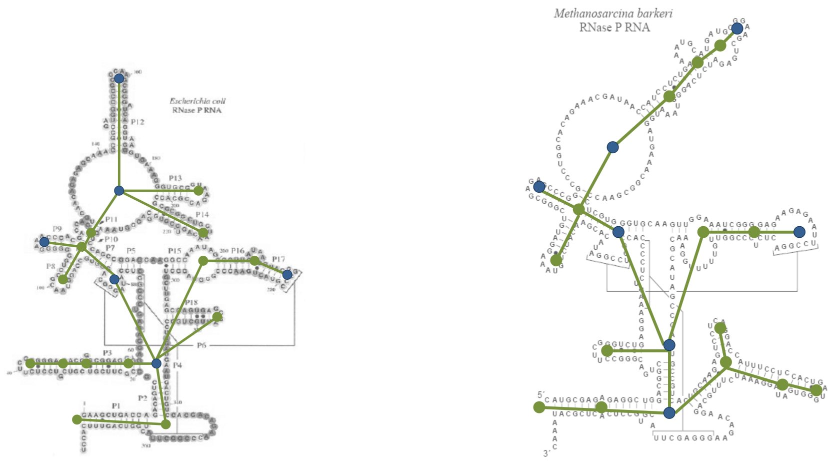


Fig. 1. The known secondary structures for two RNase P sequences and the corresponding coarse-grain trees. (left) *E. coli* RNase P, based on [7], shaded nts represent conserved loci. (right) *M. barkery* RNase P obtained from the RNase P database http://www.mbio.ncsu.edu/RNaseP/).

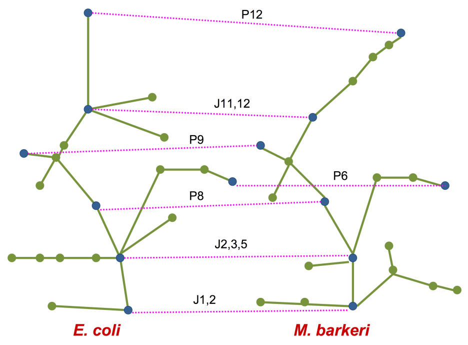
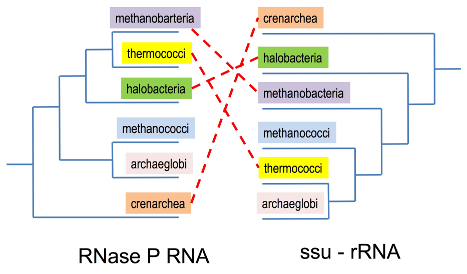


Fig. 2. Seeded tree alignment for the *E. coli* versus *M. barkery* RNase P secondary structures shown in Fig. 1. Dark vertices represent conserved loci, dotted lines represent alignment seeds.

sequences and secondary structures. In [7], the extent and patterns of evolutionary variation in RNase P RNA sequence and structure were studied, in both bacterial and archeal species, and it was shown that highly-conserved bases are scattered throughout the sequence and secondary structure, and are concentrated in the vicinity of the pre-tRNA binding surface of the tertiary structure. Furthermore, there are several helices, both in the core and periphery of the RNA, that are conserved in sequence at the base and terminal loop but are extremely variable in sequence along the length of the helix. The proximal ends of these helices are located within important conserved sequence and structure, and interact at their terminal loops in secondary or tertiary contacts elsewhere in the molecule. A detailed description of the conserved loci is given in [7] and shown in Fig. 1. In terms of RNA secondary structure tree comparison, this means that in a biologically correct alignment of two RNase P trees, the nodes corresponding to the conserved loci should be mapped to each other ("alignment seeds"), as shown in Fig. 2.

The need to align seeded tree-pairs also arises in applications where the bioinformatics data is represented in the form of unordered trees. To demonstrate this, consider the example in Fig. 3, which illustrates the reconciliation of a phylogenetic tree based on archeal RNase P structures with the phylogenetic tree based on archeal rRNA structures. This figure is based on a study by Harris et al. [8], where a detailed comparative analysis of archaeal RNase P RNA structures is reported, based on 37 sequences from a wide range of



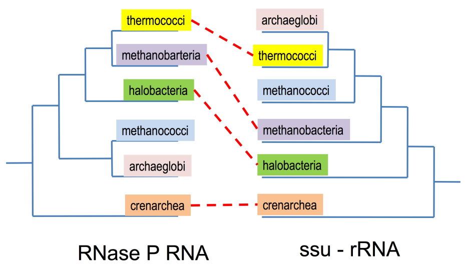


Fig. 3. Seeded phylogenetic unordered tree alignment, in the context of horizontal gene transfer prediction. (left) The tanglegram formed by connecting, via seed edges, the phylogenetic tree based on archeal RNase P structures [8] with another phylogenetic tree based on archeal RNA [29]. The seed edges for the two direct neighbors, according the RNase P RNA tree: Archeaglogi and Methanococci, which are putatively involved in RNase P RNA horizontal transfer [8], were omitted. (right) The planar layout of the tanglegram.

archaeal species. The RNase P RNA sequences were rigorously aligned in a comparative analysis of secondary structure, providing an opportunity to compare phylogenetic relations derived from RNase P RNA sequences with those derived from small subunit ribosomal RNA sequences from the same group of organisms [12].

Although the RNase P RNA sequences generally recreate trees similar to those based on rRNA, a significant exception is the placement of the sequence from *Archaeoglobus fulgidus*. In rRNA-based trees, this genus lies on a branch distinct from the other major euryarchaeal groups, separating from the other groups at approximately the bifurcation between methanobacteria and halobacteria/methanomicrobia [12]. The *A. fulgidus* RNase P RNA, however, is clearly related in structure and sequence to those of Methanococcus. Trees constructed using parsimony (DNAPARS) and maximum likelihood (DNAML) methods [4] agree on the placement of this sequence as a relative of *Methanococcus*, and this placement is robust. The most likely interpretations of the similarities between RNase P RNAs of *Methanococcus* and *A. fulgidus* are that either (1) the ribosomal RNA-based trees are for some reason misleading, and *A. fulgidus* is specifically related to the methanococcus, or (2) the gene encoding RNase P RNA has been transferred laterally from one group to another.

The above analysis could be formulated as a seeded unordered tree alignment, as follows (see Fig. 3). Connect each leaf from the RNase P RNA tree with the corresponding (same species) leaf from the ssu-rRNA tree, if such exists. Note that the layout of two unordered trees with additional edges forming a bijection among their leaves is called a *tanglegram* [17]. It is easy to see that the seeded unordered trees can be aligned if the input trees can be put in a non-crossing representation (in other words: if the tanglegram formed by the input trees together with the seed has a planar layout). Correspondingly, when formulating the problem raised by [8] as that of seeded unordered tree alignment: if the tanglegram formed by the two seeded RNA trees has a planar layout, and the two trees agree, then there is no basis for a lateral transfer hypothesis. In the above example, however, the tanglegram formed by the two RNA trees can be untangled, and the two trees can be aligned after removing the seed edges corresponding to the two new neighbors (by RNase P RNA homology) *Archeaglobi* and *Methanococci*. This supports the hypothesis of a lateral transfer of the gene encoding RNase P RNA from *Archeaglobi* to *Methanococci*, or vice versa.

2 Formalism

Consider the constrained form of the tree matching problem in which the mapping of a subset of the nodes in one tree to a corresponding subset of the nodes in the other tree is given in advance. The initial node mapping is called the *seed set* of the matching.

Definition 1 (mapping). *A* mapping M *of a tree* $T_1 = (V_1, E_1)$ *to a tree* $T_2 = (V_2, E_2)$ *is a bijection* $M \subseteq V_1 \times V_2$ *such that for all* $(v_1, v_2), (w_1, w_2) \in M$*, it holds that* v_1 *is an ancestor of* w_1 *in* T_1 *if and only* *if v_2 is an ancestor of w_2 in T_2. A seed set S is a bijection $S \subseteq M \subseteq V_1 \times V_2$ such that S itself also is a mapping of T_1 to T_2.*

Among all possible mappings, in this paper we deal with the commonly used least-common-ancestor (LCA) preserving ones.

Definition 2 (LCA-preserving mapping). *Let $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ be trees, and let $M \subseteq V_1 \times V_2$ be a mapping of T_1 to T_2. M is* LCA-preserving *if the following condition holds: if $(x_1, x_2) \in M$ and $(y_1, y_2) \in M$ then $(lca(x_1, y_1), lca(x_2, y_2)) \in M$.*

We next define a new tree alignment optimization problem over pairs of seeded trees, to be applied as a constrained form of a general, pre-selected tree alignment algorithm. Therefore, let `TAA`(T_1, T_2) denote a "black box" tree alignment algorithm, which applies a pre-selected tree alignment algorithm to an input consisting of two labeled trees T_1 and T_2. The class of tree alignment algorithms to which the seed constraint can actually be applied is discussed in the following section.

Definition 3 (seeded tree alignment problem). *Given two trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$, a set of seeds $S \subseteq V_1 \times V_2$, and a predefined tree similarity measure MAP, such that $MAP(M)$ denotes a similarity score computed based on the pairs of nodes $(v_1, v_2) \in M$. The* seeded tree alignment problem *`STA`$(T_1, T_2, S,$`TAA`$)$ is to find a mapping $M \subseteq V_1 \times V_2$ such that $S \in M$ and the alignment score $MAP(M)$ is maximal under this constraint.*

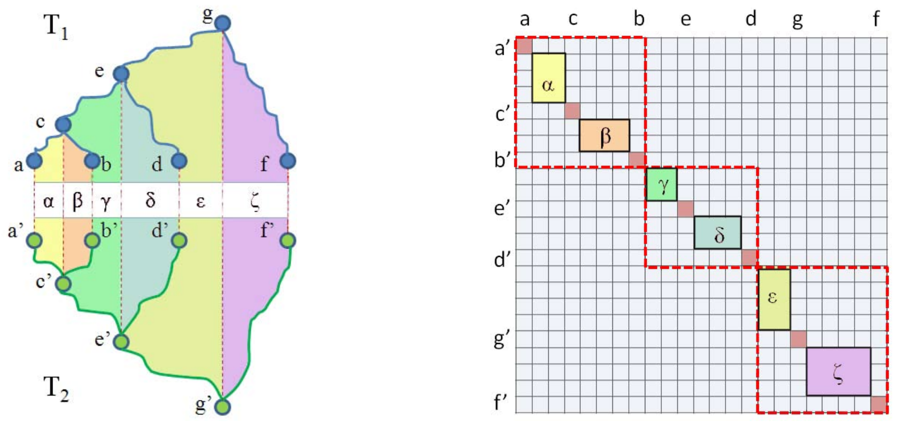


Fig. 4. An illustration of the dynamic programming table computed during the seeded matching algorithm. (left) The matched trees with a partitioning induced by seeds. (right) The corresponding DP table, divided into independent rectangles to be computed by an appropriate LCA-preserving mapping algorithm. The colored areas illustrate which parts of the DP need to be computed. The lowest right-most corner of each colored rectangle holds the value for the roots of the corresponding compared subtrees. Each dashed rectangle corresponds to a secondary seed. Within each dashed rectangle, the single-cell components correspond to seeds, where the primary seeds are at the bottom-right and top-left (only for the subtrees framed by two primary seeds) corners, and the secondary seed is located between the two rectangles, each corresponding to one of the subtrees rooted at this secondary seed.

3 Tree Alignment based on Seeded Nodes

In this section, we show how to efficiently apply seeded alignment on top of existing tree alignment algorithms. We note that our results apply to LCA-preserving mappings (see Def. 2). This class of algorithms includes subtree isomorphism [13, 14, 21], subtree homeomorphism [1, 19, 20], and maximum common subtree (MCS) [25] finding algorithms. For the sake of clarity of presentation, we note that all these algorithms employ a dynamic programming table where entry $[i, j]$ denotes the similarity score of subtree i of tree T_1 versus subtree j of tree T_2. Moreover, the time complexity of each of the above algorithms is computed by summing up the work invested in matching a subtree $t_u \in T_1$ with a subtree $t_v \in T_2$:

$$O(\sum_{u=1}^{n}\sum_{v=1}^{m}(c(u)^x c(v)^y f(c(u), c(v))) \tag{1}$$

where $|T_1| = n$, $|T_2| = m$, $c(u)$ denotes the out-degree of t_u, $c(v)$ denotes the out-degree of t_v, and $f(c(u), c(v))$ is a concave function that differs from one algorithm to another along with coefficients x and y. For example, unordered subtree homeomorphism can be computed in $O(nm\sqrt{m})$ time using the top-down algorithm of [1] and the corresponding concave function is $\sqrt{m}$ (see Exmp. 1 for a complete analysis).

In the discussion to follow, let the seeds contained in the initial seeds set S be denoted *primary* seeds. Since we restrict our analysis to LCA-preserving mappings, the LCAs of the primary seeds also function as seeds, to be denoted *secondary* seeds (see Fig. 4 (left)). For the sake of simplicity of presentation we will describe the seeded tree alignment algorithm for binary trees. Extensions to non-binary trees are straightforward. Note that, given an LCA-preserving tree alignment algorithm, and given as input a planar layout tanglegram of a pair of seeded trees that are to be aligned, the corresponding seeded tree alignment could immediately be derived by extending the applied node label similarity table as follows: For each seed $s = (u, v) \in S$ such that $u \in T_1$ and $v \in T_2$, relabel the seeded nodes to u' and v' respectively and add two new rows and two new columns to the label similarity table – one for node u' and the other one for node v'. Then, the similarity score for entries $[u', v']$ and $[v', u']$ is set to infinity while all the remaining entries in these two rows and columns are set to zero. This way we ensure that the above LCA-preserving tree alignment algorithms will match seeds as required.

Having said that, in this section we show how to exploit the seeds to more efficiently apply the tree alignment, and avoid redundant work by restricting the computations to limited areas in the dynamic programming (DP) matrix. This "constrained-by-seeds" dynamic programming can be intuitively explained by following the example in Fig. 4. A regular, unconstrained application of the LCA-preserving algorithms mentioned above to the two trees in Fig. 4 (left) would require the computation of each and every entry in the DP table of Fig. 4 (right). The algorithm described below, however, will only compute the shaded rectangles along the diagonal of the table. Note that each primary seed corresponds to a single entry in the DP table whose score can be computed in an initialization step. Furthermore, each pair of consecutive seeds in S (according to a planar layout) defines a rectangle in the DP matrix with a side of size at most k, where k denotes the maximum gap between two consecutive seeds (in a planar layout), that can be filled independently of other rectangles. This is true for all entries except for the single entry in the rectangle which corresponds to a secondary seed, and whose computation depends on the availability of entries external to the rectangle. This availability, however, can be taken care of if the rectangles are processed by postorder traversal of the corresponding secondary seeds. The number of rectangles is bounded by $n/2k$ and thus, there is an immediate $O(nk)$ bound on the number of entries that need to be computed in the table (in comparison to $O(n^2)$ entries in the unconstrained tree alignment case). Furthermore, each application of `TAA` is given as input two subtrees with no more than k nodes. The time complexity of seeded LCA-preserving tree alignment algorithms is formally analyzed in Obs. 2 and demonstrated in Exmp. 1.

We refer the reader to Fig. 4 (left) for the following discussion. Consider the subtree obtained during a postorder traversal of T_1, from node c to node d: note that all nodes located in the left part are colored green and all nodes located in the right part are colored blue. Correspondingly, in the subtree obtained during a postorder traversal of T_2, from node c' to node d', all nodes located in the right part are colored green and all nodes located in the left part are colored blue. This correspondence of colors is explained by Obs. 1; before we state it we need the following definition.

Definition 4. *For any tree T and nodes $x, y \in T$, let t_{x-y} denote the subtree consisting of all nodes found in a postorder traversal of T, starting from node x and ending in node y. Also, let $left_{x-y}$ and $right_{x-y}$ denote the left and the right subtrees of t_{x-y}, respectively.*

Note that both left$_{x-y}$ and right$_{x-y}$ are rooted at $lca(x, y)$.

Observation 1. *Let $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ be trees to be aligned and $(x_1 \in V_1, x_2 \in V_2)$ and $(y_1 \in V_1, y_2 \in V_2)$ be a pair of seeds such that $x_1 < y_1$ and $x_2 < y_2$ in the postorder traversal of T_1 and T_2, respectively. In an LCA-preserving mapping of T_1 to T_2, all nodes in left$_{x_1-y_1}$ are mapped to nodes in right$_{x_2-y_2}$. Symmetrically, all nodes in right$_{x_1-y_1}$ are mapped to nodes in left$_{x_2-y_2}$.*

Proof. By recursive invocation of Def. 2. □

The seeded tree alignment algorithm starts by extending the seeds set S to include the secondary seeds. Next, it orders S such that all the seeds obey a planar layout, that is, there is no crossing between seeds. An algorithm to compute this layout, if such layout exists, is given in Sect. 4. The resulting order partitions the target trees, according to Obs. 1, into exclusive subtree-pair intervals (see Fig. 4 (right)). The suggested algorithm processes these subtree pairs in postorder traversal of their roots (which are paired as secondary seeds). For each such interval, it retrieves the corresponding subtrees and feeds them as input to `TAA`.

The pseudocode for the algorithm is given below; we refer the reader to the Appendix for some special cases that are handled by the algorithm.

Algorithm 1. Given two trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ and a set of primary seeds $S \subseteq V_1 \times V_2$, find a best possible mapping M of T_1 to T_2 such that $S \subseteq M \subseteq V_1 \times V_2$.

```
1: procedure Seeded Matching($T_1, T_2, S$)
2:     $S' \leftarrow$ a set of secondary seeds based on $S$
3:     $S \leftarrow S \cup S'$
4:     $S \leftarrow$ layout order($S$)
5:     for all primary seeds $(x, x')$ in $S$ do
6:         $DP[x, x'] \leftarrow$ TAA($t_x, t_{x'}$)
7:     $(x, x') \leftarrow$ the first secondary seed in a postorder on $T_1$
8:     while $(x, x') \neq$ the last secondary seed in a postorder on $T_1$ do
9:         $(y, y') \leftarrow$ the left child seed of $(x, x')$
10:        $(z, z') \leftarrow$ the right child seed of $(x, x')$
11:        $DP[y, y' \ldots x, x'] \leftarrow$ TAA($left_{x-y}, right_{x'-y'}$)
12:        $DP[x, x' \ldots z, z'] \leftarrow$ TAA($right_{y-z}, left_{y'-z'}$)
13:        $DP[x, x'] \leftarrow$ TAA($t_x, t_{x'}$)
14:        $(x, x') \leftarrow$ the next secondary seed in a postorder on $T_1$
15:    return $DP[r_1, r_2]$
```

Lemma 1. *Let $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ be two trees to be aligned, and let $S \subseteq V_1 \times V_2$ be a primary seeds set. Given an LCA-preserving tree alignment algorithm* `TAA` *and the corresponding score function MAP, Alg. 1 computes* `STA`*($T_1, T_2, S,$* `TAA`*).*

Proof. The condition $S \in M$ is kept by lines 5,6 of Alg. 1. LCA-preservation is kept by the definition of secondary seeds, by adding secondary seeds to S, and by the fact that, in line 13 of Alg. 1, for any seed cell $DP[i, j]$, the values all other entries in line i and all other entries in line j remain set to null. In lines 11, 12 of Alg. 1, the pre-selected tree alignment algorithm `TAA` computes the optimal score to each one of the subtree-pairs confined by the seeds, according to the LCA-preservation constraint enforced by Obs. 1. The candidate LCA-preserving algorithms mentioned above compute the values of DP in bottom-up node order and thus the postorder processing of the subtrees corresponding to secondary seeds ensures that the necessary node-values are available when needed. Therefore, by Def. 3, the resulting alignment is the best scoring one under the seed constraints. □

Restricting the computations to limited areas in the DP matrix results in a speedup of the applied, predefined tree comparison algorithms, as analyzed below.

Lemma 2. *The above framework for computing* STA(T_1, T_2, S, TAA) *yields a speedup of* $\Omega((n/k)^{x+y-1} f(n,n)/f(k,k))$ *over the time complexity of the corresponding, unseeded, tree alignment algorithm* TAA(T_1, T_2).

Proof. Let $f(c(u), c(v))$ denote the concave function quantifying the work that a given (LCA-preserving, DP subtree-to-subtree based) tree-comparison algorithm TAA applies per alignment of a subtree $t_u \in T_1$ with a subtree $t_v \in T_2$, where $c(u)$ denotes the out-degree of t_u and $c(v)$ denotes the out-degree of t_v.

Observation 2. $\sum_{u=1}^{k} c(u) = k$ *and* $\sum_{v=1}^{k} c(v) = k$.

Summing up the work over all node pairs, applying Obs. 2 to Eq. 1 we get:

$$O(\frac{n}{k}\sum_{u=1}^{k}\sum_{v=1}^{k}(c(u)^x c(v)^y f(c(u), c(v))) =$$
$$= O(\frac{n}{k}k^x \sum_{v=1}^{n}(c(v)^y f(c(u), k) = O(\frac{n}{k}k^x k^y f(k,k)) = O(nk^{x+y-1}(f(k,k))\,.$$

This yields a speedup of $\Omega((n/k)^{x+y-1} f(n,n)/f(k,k))$ over the time complexity obtained by applying the corresponding unseeded version of the tree comparison algorithm. Below we give an example of one such seeded tree alignment algorithm. Time complexities of the seeded versions of additional currently known LCA-preserving tree comparison algorithms will be given in the full version of this paper. ☐

Example 1 (Top-down unordered subtree homeomorphism [1]). The algorithm for top-down unordered subtree isomorphism between trees T_1 and T_2 with $|T_1| = n_1$ and $|T_2| = n_2$ runs in $O(n_1 n_2 \sqrt{n_2})$ time, since

$$O(\sum_{u=1}^{n_1}\sum_{v=1}^{n_2}(c(u)c(v)\sqrt{c(v)}) = O(n_1 n_2 \sqrt{n_2})$$

When applied over a seeded tree matching, we get

$$O((n_1/k)\sum_{u=1}^{k}\sum_{v=1}^{k}(c(u)c(v)\sqrt{c(v)}) = O((n_1/k)k\sum_{u=1}^{k}(c(v)\sqrt{c(v)})) =$$
$$= O((n_1/k)k^2\sqrt{k}) = O(n_1 k\sqrt{k})$$

Thus, if the compared trees are heavily seeded and $k = O(1)$ then the algorithm runs in $O(n_1)$ time and the speedup factor is $O(n_2\sqrt{n_2})$.

4 Planar Tanglegram Layout

A layout of two unordered trees with additional edges forming a bijection among their leaves, is called a *tanglegram* [17]. These diagrams arise in host-parasite cospeciation studies and in the reconciliation of gene and species phylogenies.

Definition 5 (Tanglegram). *A* tanglegram *is a triple* (T_1, T_2, S) *where* $T_1 = (V_1, E_1)$ *and* $T_2 = (V_2, E_2)$ *are unordered trees, and* $S \subseteq V_1 \times V_2$ *is a seed, that is, a partial mapping of* T_1 *to* T_2. *A tanglegram is binary if both* T_1 *and* T_2 *are binary trees.*

Given a tanglegram (T_1, T_2, S), we will be interested in finding a way to represent the two trees in such a way that the seed does not create any crossings among the edges corresponding to seeds in that representation. We call such a representation a *planar layout* of the tanglegram. To define it formally, we first introduce the notion of an extension of a set (and a pair of sets) of nodes.

Definition 6 (Extension). *Let T_1 be an unordered tree, let X be an ordered set of nodes in T_1, and let $u \in X$ be a non-leaf of T_1. Denote by X' the ordered set X where u has been replaced by its children in some particular ordering. Then, we call X' a* one-step extension *of X. We say that Z is an* extension *of X if there is a sequence of zero or more one-step extensions from X to Z.*

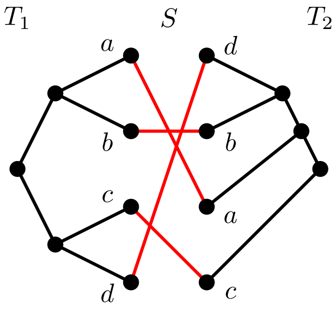
Let Y be an ordered set of nodes in an unordered tree T_2. Then, we also say that (X', Y') is an extension of (X, Y) if X' is an extension of X and Y' is an extension of Y.

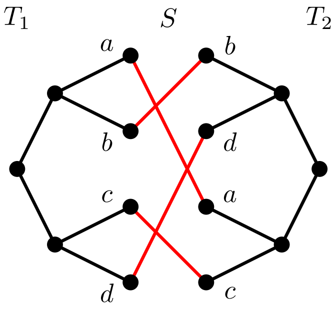
We are interested in extending the pair formed by the roots of the two trees of a tanglegram until there is no point in extending it further. The extensions are performed until no seed with seeded descendants can be found (for instance, seeded leaves satisfy this condition). In the following, we will call these nodes *terminals*.

Definition 7 (Planar layout). *Let T_1 and T_2 be unordered trees with roots r_1 and r_2, respectively. A* planar layout *of a tanglegram (T_1, T_2, S) is a pair (x, y) with $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$, such that:*

- *(x, y) is an extension of $((r_1), (r_2))$,*
- *the nodes in x and y are terminals, and*
- *$(x_i, y_i) \in S$ for every i with $1 \leqslant i \leqslant n$.*

Example 2. The tanglegram to the left has a planar layout, namely: $((a, b, d, c), (a, b, d, c))$, while the one to the right does not.





We next describe an algorithm for finding a planar layout of a binary tanglegram. The procedure *Untangle* computes the layout of a binary tanglegram by successive refinements of two lists, the ordered sets X and Y, which initially contain the roots of the trees. At each iteration of the loop, a node of one of the lists is "refined," which means that a correct ordering of its children is found and fixed for the rest of the algorithm. The loop stops when all the elements of the lists X and Y are terminal nodes of the trees; at this point, the planar layout (if it exists) is completed.

Before starting the main loop, the procedure *Paths* computes a table P of Boolean values which can be understood as an extension of the bijection S to all the nodes of the trees. In particular, for any node u in T_1 and any node v in T_2, $P[u, v]$ is true if and only if the subtree of T_1 rooted at u has a descendant u', the subtree of T_2 rooted at v has a descendant v', and $(u', v') \in S$.

The computation made by *Paths* can follow a dynamic programming approach. In the first place, the entries corresponding to the leaves are given by s. Then, it proceeds by computing the entries (u, v) where u and v are inner nodes in the trees. The value of $P[u, v]$ is set to *true* if and only if an entry $P[u_i, v_j]$ has the value *true* for some child u_i of u and some child v_j of v. The cost of computing all the entries is, therefore, $O(n^2)$.

Now we return to the main procedure.

Algorithm 2. Given a tanglegram (T_1, T_2, S), obtain a planar layout (X, Y) for (T_1, T_2, S). Let r_1, r_2 be the roots of T_1, T_2, respectively.

```
procedure Untangle($T_1, T_2, S$)
    $X, Y \leftarrow (r_1), (r_2)$
    $E \leftarrow \{\{r_1, r_2\}\}$
    $P \leftarrow$ *Paths*$(T_1, T_2, S)$
    **while** $X \cup Y$ contain some non-terminal node **do**
        $u \leftarrow$ a non-terminal node of highest degree in $(X \cup Y, E)$
        **if** $u$ is in $X$ **then**
            *Refine*$(u, X, Y, E, P)$
        **else**
            *Refine*$(u, Y, X, E, P)$
    **return** $(X, Y)$
```

In the refinement step, a node u in the graph $(X \cup Y, E)$ is substituted by its children u_1, u_2 in such a way that no edge crossing is introduced.

Algorithm 3. Given a partial planar layout $(A \cup B, E)$ and a node u, refine the planar layout by substituting u by its children and return A and E modified according to the refinement.

```
  **procedure** *Refine*$(u, A, B, E, P)$
    $u_1, u_2 \leftarrow$ children of $u$
    **for every** node $v \in B$ such that $\{u, v\} \in E$ **do**
        **if** $P[u_1, v]$ **then**
            add edge $\{u_1, v\}$ to $E$
        **if** $P[u_2, v]$ **then**
            add edge $\{u_2, v\}$ to $E$
        delete $\{u, v\}$ from $E$
    **if** $u_1$ is an isolated node in $(\{u_1\} \cup B, E)$ **then**
        replace $u$ by $u_2$ in $A$
    **else if** $u_2$ is an isolated node in $(\{u_2\} \cup B, E)$ **then**
        replace $u$ by $u_1$ in $A$
    **else if not** *Crossings*$(u_1, u_2, B, E)$ **then**
        replace $u$ by the ordered set $(u_1, u_2)$ in $A$
    **else if not** *Crossings*$(u_2, u_1, B, E)$ **then**
        replace $u$ by the ordered set $(u_2, u_1)$ in $A$ and flip clade $u$
    **else**
        **reject**
```

The above procedure selects an ordering of the nodes $U = \{u_1, u_2\}$ such that, replacing u by U in A, the graph $(A \cup B, E)$ does not create any edge crossings. Formally, we say that $(A \cup B, E)$ has an *edge crossing* if there are two nodes a_1, a_2 in A and two more nodes b_1, b_2 in B, appearing in this order in A and B, such that E contains the edges (a_1, b_2) and (a_2, b_1). Assuming $(A \cup B, E)$ does not already have any edge crossings before replacing u by U in A, this property is checked in the procedure *Crossings* with cost $O(n)$ by just checking if any edge adjacent with node u_2 crosses the last (in the order given by B) edge adjacent with node u_1.

Note that the whole algorithm can be thought of as the computation of an extension of $((r_1), (r_2))$ (where r_1 and r_2 are the roots of the initial trees), which becomes a planar layout at the end. In order to prove the correctness of *Untangle*, we introduce the following concept.

Definition 8 (Promising partial layout). *Let $T = (T_1, T_2, S)$ be a tanglegram, let X be an ordered set of nodes in T_1, and let Y be an ordered set of nodes in T_2. Then, we say that (X, Y) is* promising *for T if it extends to a planar layout of T.*

In the following, $T = (T_1, T_2, S)$ will denote the binary tanglegram which is given as input to *Untangle*. Additionally, X and Y will represent, as above, two ordered sets of nodes of the trees T_1 and T_2, respectively, and E will be the set of pairs of nodes kept by the algorithm. The following lemma provides an invariant for the loop in *Untangle*.

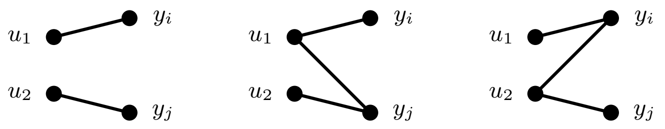


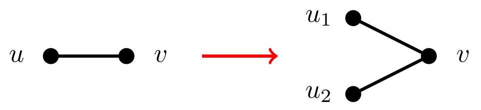
Fig. 5. Possible connections for u_1, u_2 without crossings (and degree at least 1). The exchange of u_1 and u_2 would create an edge crossing in all cases.

Lemma 3. *If* (X, Y) *is promising for* T *at the beginning of an iteration of the while loop in* Untangle*, then it is promising for* T *at the end.*

Proof. Suppose that (X, Y) is promising at the beginning of the loop, and let u be a nonterminal of highest degree. Without loss of generality, we can suppose that $u \in X$. Note that u must have degree at least one in the graph $G = (X \cup Y, E)$, since a promising pair cannot have isolated nodes. Let u_1 and u_2 be the children of u in the corresponding tree. Now, in case that u_1 (u_2) is isolated in G, the algorithm just deletes it and replaces u by u_2 (u_1). Since an isolated node cannot contribute to a planar layout, the new pair (X, Y) must be promising, and we are done.

Suppose now that none of u_1 or u_2 is isolated. Let X_1 be X with u replaced by (u_1, u_2); symmetrically, let X_2 be X with u replaced by (u_2, u_1). We now differentiate between degree one and degree larger than one for u:

Degree $= 1$. Since u has highest degree, its neighbor v must have degree 1 too and, then, $\{u, v\}$ is an edge in G that is not incident upon any other edge, so it cannot be crossed over by any other edge in the one-step extensions of (X, Y).



The fact that (X, Y) is promising means that it must extend to a planar layout, say (x, y), which is obtained either from the pair (X_1, Y) or from (X_2, Y). In the first case, (X_1, Y) must be promising. In the second case, (X_2, Y) is promising but since the subgraph of G induced by $\{u_1, u_2, v\}$ is a connected component of G, exchanging u_1 and u_2 and reversing children's order in any later refinement of them leads to a planar layout. Therefore, (X_1, Y) must be promising, too.

Degree > 1. Observe that we have now the following interesting situation:

Claim. If $(X_1 \cup Y, E)$ does not have any edge crossing, then $(X_2 \cup Y, E)$ has some edge crossing.

The reason for the above claim is the following. Let $N(u_1)$ be the set of neighbors of u_1, and $N(u_2)$ the set of neighbors of u_2. Then, there must be at least a node in the symmetric difference of $N(u_1)$ and $N(u_2)$; otherwise, if $|N(u_1)| = |N(u_2)| = 1$, u would have degree one (which is not the present case), and if $|N(u_1)| = |N(u_2)| > 1$, there would be an edge crossing. Suppose that there is a node w in the symmetric difference of $N(u_1)$ and $N(u_2)$ that actually belongs to $N(u_1)$ (the other case being similar). Then, since we are assuming that $(X_1 \cup Y, E)$ has no edge crossings, w must appear in Y before all the neighbors of u_2. Then, if $Y = (y_1, \ldots, y_m)$, we have the edge $\{u_1, w\}$ in E with $w = y_i$, for some $i \leq m$, and some other edge (from $N(u_2)$) $\{u_2, y_j\}$ for $j > i$. Now, if we exchange the order of u_1 and u_2, we get an edge crossing, as we wanted to show. (See Fig. 5.)

It is still left to show that the pair (X', Y) is promising, for X' being the new value of X at the end of the iteration (that is, $X' = X_1$ or $X' = X_2$). But now, it is straightforward since we are supposing that (X, Y) is promising. If no edge crossings are found with the ordering (u_1, u_2) then, by the Claim, the graph $(X_2 \cup Y, E)$ has some edge crossing and, therefore, (X_2, Y) cannot be promising. But then, (X_1, Y) must be promising. On the contrary, if some edge crossing is found for (u_1, u_2), then $(X_2 \cup Y, E)$ must be promising since (X, Y) is. In any case, the new (X', Y) must be promising. □

Remark 1. Note that the choice for u made in Alg. 2 (as a non-terminal node of highest degree) is used in the *Degree 1* case of the previous proof. The only possibility that must be avoided, however, is choosing a node u of degree 1 adjacent to a node of degree > 1, since it would not be clear – at this stage – what the right ordering of the children of u is. So, the "highest degree" condition ensures that if u has degree 1, its (only) neighbor must have degree 1 too and then, both possible orderings of the children of u give rise to a promising pair, as it is argued in the proof of Lemma 3.

Theorem 1. *The procedure Untangle*(T_1, T_2, S) *computes a planar layout for* (T_1, T_2, S) *if there is one.*

Proof. Supose there is a planar layout for $T = (T_1, T_2, S)$. Then, if r_1 is the root of T_1 and r_2 is the root of T_2, it is clear that $((r_1), (r_2))$ must be promising for T. By Lemma 3, the pair (X, Y) is kept as a promising pair until the main loop is exited. At this point, X and Y only contain terminals, (X, Y) is certainly an extension of $((r_1), (r_2))$, and there are no crossings. Therefore, (X, Y) is a planar layout. □

Lemma 4. *Algorithm 2 runs in* $O(n^2)$ *time and space.*

Proof. Let T_1 and T_2 be unordered trees with $|T_1| = n_1$ and $|T_2| = n_2$, and let $n = n_1 + n_2$. The cost of Alg. 2 is dominated by the computation of the path matrix P, which takes $O(n^2)$ time and uses $O(n^2)$ additional space. Once P is available, the *Refine* procedure is called exactly once for each non-terminal node of the trees, and in each call the neighbors of the node in the graph $(A \cup B, E)$ are updated in $O(\max(n_1, n_2)) = O(n)$ time; the *Crossing* procedure also takes $O(n)$ time. Therefore, the *Untangle* procedure runs in $O(n^2)$ time. □

Note that, in practical applications, the local, or "all subtree versus subtree" version of seeded tree alignment is actually sought, in which case we iteratively run the described framework over all subtree pairs of T_1 and T_2. In this case, P is only constructed once, as a preprocessing step, in $O(n^2)$, and then, for each local seeded subtree pair to be aligned, the processing work consists of untangling the corresponding tanglegram in $O(n)$, using the table P which was already computed in the preprocessing stage, and then applying the seeded tree alignment algorithm. Since there are $O(n^2)$ subtree pairs to be processed, the bottleneck in practice is actually dictated by the time complexity of the seeded tree alignment, according to the density of the given seeds set and the pre-selected tree alignment algorithm `TAA` to be applied.

The extension of the *Untangle* procedure to compute the planar layout of an arbitrary – not necessarily binary – tanglegram, is an interesting open problem. While the top-down approach of the binary case is maintained, the refinement of the nodes, in the general case, implies replacing a node by an arbitrary number of new nodes. In order to sort them correctly, so that the whole graph is kept planar, a characterization in terms of caterpillars [5] can be used, together with a technique for grouping the nodes which cannot be correctly ordered at some particular step.

The optimization problem of finding the smallest number of seeds to be removed from a tanglegram in order to obtain a planar layout, is another interesting line of future research.

Acknowledgements

The research described in this paper has been partially supported by the Spanish CICYT projects TIN 2004-07925-C03-01 GRAMMARS, TIN2004-04343 iDEAS, TIN2005-08832-C03-03 MOISES-BAR, MTM2006-07773 COMGRIO, and by EU project INTAS IT 04-77-7178.

References

1. Chung, M.J.: $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. J. Algorithms **8** (1987) 106–112
2. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Zhang, L.: On distances between phylogenetic trees. In: Proc. 8th Annual ACM-SIAM Symp. Discrete Algorithms, ACM Press (1997) 427–436

3. Dufayard, J.F., Duret, L., Penel, S., Gouy, M., Rechenmann, F., Perrière, G.: Tree pattern matching in phylogenetic trees: Automatic search for orthologs or paralogs in homologous gene sequence databases. Bioinformatics **21** (2005) 2596–2603
4. Felsenstein, J.: Phylip—phylogeny inference package (version 3.2). Cladistics **5** (1989) 164–166
5. Fößmeier, U., Kaufmann, M.: Nice drawings for planar bipartite graphs. In Bongiovanni, G., Bovet, D.P., Battista, G.D., eds.: Proc. 3rd Italian Conf. Algorithms and Complexity. Volume 3393 of Lecture Notes in Computer Science. Springer-Verlag (1997) 122–134
6. Gardiner, K.J., Marsh, T.L., Pace, N.R.: Ion dependence of the bacillus subtilis rnase p reaction. J. Biol. Chem. **260** (1985) 5415–5419
7. Haas, E.S., Brown, J.W.: Evolutionary variation in bacterial RNase P RNAs. Nucleic Acids Res. **26** (1998) 4093–4099
8. Harris, J.K., Haas, E.S., Williams, D., Frank, D.N., Brown, J.W.: New insight into RNase P RNA structure from comparative analysis of the archaeal RNA. RNA **7** (2001) 220–232
9. James, B.D., Olsen, G.J., Liu, J., Pace, N.R.: The secondary structure of ribonuclease P RNA, the catalytic element of a ribonucleoprotein enzyme. Cell **52** (1988) 19–26
10. Jansson, J., Hieu, N.T., Sung, W.K.: Local gapped subforest alignment and its application in fnding RNA structural motifs. J. Comput. Biol. **13** (2006) 702–718
11. Le, S.Y., Nussinov, R., Maizel, J.V.: Tree graphs of RNA secondary structures and their comparisons. Comput. Biomed. Res. **22** (1989) 461–473
12. Maidak, B.L., Cole, J.R., Lilburn, T.G., Parker, C.T., Saxman, P.R., Stredwick, J.M., Garrity, G.M., Li, B., Olsen, G.J., Pramanik, S., Schmidt, T.M., Tiedje, J.M.: The RDP (Ribosomal Database Project) continues. Nucleic Acids Res. **28** (2000) 173–174
13. Matula, D.W.: An algorithm for subtree identification. SIAM Rev. **10** (1968) 273–274
14. Matula, D.: Subtree isomorphism in $o(n^{5/2})$. Ann. Discrete Math. **2** (1978) 91–106
15. Nye, T.M., Lio, P., Gilks, W.R.: A novel algorithm and web-based tool for comparing two alternative phylogenetic trees. Bioinformatics **22** (2006) 117–119
16. Pace, N.R., Brown, J.W.: Evolutionary perspective on the structure and function of ribonuclease P, a ribozyme. J. Bacteriol. **177** (1995) 1919–1928
17. Page, R.D.M., ed.: Tangled Trees: Phylogeny, Cospeciation, and Coevolution. The University of Chicago Press (2002)
18. Page, R.D.M., Valiente, G.: An edit script for taxonomic classifications. BMC Bioinformatics **6** (2005) 208
19. Pinter, R.Y., Rokhlenko, O., Tsur, D., Ziv-Ukelson, M.: Approximate labelled subtree homeomorphism. In: Proc. 15th Ann. Symp. Combinatorial Pattern Matching. Volume 3109 of Lecture Notes in Computer Science., Springer-Verlag (2004) 59–73
20. Reyner, S.W.: An analysis of a good algorithm for the subtree problem. SIAM J. Comput. **6** (1977) 730–732
21. Shamir, R., Tsur, D.: Faster subtree isomorphism. J. Algorithms **33** (1999) 267–280
22. Shan, H., Herbert, K.G., Piel, W.H., Shasha, D., Wang, J.T.L.: A structure-based search engine for phylogenetic databases. In: Proc. 14th Int. Conf. Scientific and Statistical Database Management, IEEE Computer Society Press (2002) 7–10
23. Shapiro, B.A., Zhang, K.: Comparing multiple RNA secondary structures using tree comparisons. Comput. Appl. Biosci. **6** (1990) 309–318
24. Shasha, D., Wang, J.T.L., Giugno, R.: Algorithmics and applications of tree and graph searching. In: Proc. 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. Lecture Notes in Computer Science, Springer-Verlag (2002) 39–52
25. Valiente, G.: Algorithms on Trees and Graphs. Springer-Verlag (2002)
26. Valiente, G.: A fast algorithmic technique for comparing large phylogenetic trees. In: Proc. 12th Int. Symp. String Processing and Information Retrieval. Volume 3772 of Lecture Notes in Computer Science., Springer-Verlag (2005) 371–376
27. Woese, C.R., Pace, N.R.: Probing RNA structure, function, and history by comparative analysis. In Gesteland, R.F., Atkins, J.F., eds.: The RNA World. Cold Spring Harbor Laboratory Press (1993) 91–117
28. Zhang, K., Wang, L., Ma, B.: Computing similarity between RNA structures. In: Proc. 10th Ann. Symp. Combinatorial Pattern Matching. Volume 1645 of Lecture Notes in Computer Science., Springer-Verlag (1999) 281–293
29. Hugenholtz, P.: Exploring prokaryotic diversity in the genomic era. Genome Biol. **3** (2002) reviews0003.1–reviews0003.8