

# Comparing Trees Via Crossing Minimization

Henning Fernau<sup>1,2</sup>, Michael Kaufmann<sup>1</sup>, and Mathias Poths<sup>1</sup>

<sup>1</sup> Univ. Tübingen, WSI für Informatik, Sand 13, 72076 Tübingen, Germany  
 {fernau/mk/poths}@informatik.uni-tuebingen.de

<sup>2</sup> Univ. Hertfordshire, Comp. Sci., College Lane, Hatfield, Herts AL10 9AB, UK

**Abstract.** Two trees with the same number of leaves have to be embedded in two layers in the plane such that the leaves are aligned in two adjacent layers. Additional matching edges between the leaves give a one-to-one correspondence between pairs of leaves of the different trees. Do there exist two planar embeddings of the two trees that minimize the crossings of the matching edges? This problem has important applications in the construction and evaluation of phylogenetic trees.

## 1 Introduction

The comparison of trees has applications in various areas like text data bases, bioinformatics, compiler construction etc. Various algorithms for different models for comparisons have been proposed. Keywords are tree matching, tree alignment, tree editing [12,13,14,15,16], a nice overview being [3]. Motivated by discussions with researchers from bioinformatics like D. Huson and D. Bryant, we propose to compare two tree structures by finding two most similar left-to-right orderings of their leaves. The ordering of the leaves can be made best visible by drawing one tree downward from the root and aligning all the leaves on a horizontal line. The leaves of the other tree are also aligned parallel to the first line and the corresponding tree structure is drawn from the leaves upside downward to the root. Corresponding leaves from the two trees are connected by matching edges.

Let  $L(T)$  denote the leaves of a tree  $T$ . A linear order  $<$  on  $L(T)$  is called *suitable* if  $T$  can be embedded into the plane such that  $L(T)$  is mapped onto a straight line (*layer*) in the order given by  $<$ . A *two-tree*  $(T_1, T_2; M)$  is given by a pair of rooted binary trees  $(T_1, T_2)$  with perfect matching  $M \subseteq L(T_1) \times L(T_2)$ , where the matching is given by a bijective labeling  $\lambda_i : L(T_i) \rightarrow \Lambda$  with  $(\ell_1, \ell_2) \in M$  iff  $\lambda_1(\ell_1) = \lambda_2(\ell_2)$ . A *drawing* of  $(T_1, T_2; M)$  is given by two suitable linear orders  $<_1$  and  $<_2$  on  $L(T_1)$  and  $L(T_2)$ , respectively. We assume that a drawing is *realized* by embedding  $L(T_1)$  and  $L(T_2)$  into two parallel lines  $L_1$  and  $L_2$ , so that all nodes of  $T_i$  lie within one of the half-planes described by  $L_{3-i}$ . Hence, only matching edges may cross. The number of crossings is independent of the chosen realization. Let  $cr(T_1, T_2, M, <_1, <_2)$  denote the number of crossings in the drawing of  $(T_1, T_2; M)$  given by  $<_1$  and  $<_2$ . A two-tree  $(T_1, T_2; M)$  is called *drawable* if  $cr(T_1, T_2, M, <_1, <_2) = 0$  for some linear orders  $<_1$  and  $<_2$ .

Very simple non-drawable two-trees are shown in Fig. 2. These are the smallest examples of non-drawable two-trees: all two-trees with three leaves are drawable.

Let  $cr(T_1, T_2, M, \langle \cdot, \cdot \rangle)$  denote the minimum value of  $cr(T_1, T_2, M, \langle \cdot, \cdot \rangle)$  for all suitable orders  $\langle \cdot, \cdot \rangle$ , and  $cr(T_1, T_2, M, \cdot, \cdot)$  denote the minimum value of  $cr(T_1, T_2, M, \langle \cdot, \cdot \rangle)$  for all suitable orders  $\langle \cdot, \cdot \rangle$ . An instance of TWO-TREE CROSSING MINIMIZATION (TTCM) is given by a two-tree  $(T_1, T_2; M)$ , and the parameter, a positive integer  $k$ . Question: Is  $cr(T_1, T_2, M, \cdot, \cdot) \leq k$ ? An instance of ONE-TREE CROSSING MINIMIZATION (OTCM) is given by a two-tree  $(T_1, T_2; M)$ , a suitable fixed order  $\langle \cdot \rangle$  on  $L(T_1)$ , and the parameter, a positive integer  $k$ . Question: Is  $cr(T_1, T_2, M, \langle \cdot, \cdot \rangle) \leq k$ ?

A possible application from bioinformatics for the variant ONE-TREE CROSSING MINIMIZATION is that for a known species tree different gene trees should be compared to the species tree. The more general variant TWO-TREE CROSSING MINIMIZATION supports tasks like: Compare different construction methods for phylogenetic trees for some data set or compare multiple gene trees [11].

A related important problem from graph drawing [4] is the TWO-SIDED CROSSING MINIMIZATION problem (TSCM) for bipartite graphs, where the vertices within each layer are connected only to vertices of the other layer. The main differences are that the vertices might have more than one incident edge and that no trees restrict the possible orderings. TWO-SIDED CROSSING MINIMIZATION is  $\mathcal{NP}$ -complete, and the problem remains  $\mathcal{NP}$ -complete even if the order of one of the layers is fixed [8] (ONE-SIDED CROSSING MINIMIZATION(OSCM)). Both problems are fixed-parameter tractable. In the case of a binary tree with  $n$  leaves, there are exactly  $2^{n-1}$  different leaf orders implied by different orderings of the subtrees. This is in contrast to the  $n!$  permutations which are possible in OSCM.

Similarly related is the problem of finding an embedding of a graph in the plane that minimizes the number of crossings; this problem remains  $\mathcal{NP}$ -complete even if the degree of the graph is bounded by three. Notice that a two-tree (plus matching edges) obeys this degree bound. Moreover, in that case, the crossing minimization problem is known to be fixed-parameter tractable [9].

Since we like to talk about left and right subtrees and these notions usually depend on the parent's position, let us fix the following convention: We assume that all our trees are drawn either downwards (the *upper tree*) or upwards (the *bottom tree*); then, "left" and "right" refers to how an observer would name these relative directions when viewing such a drawing. More specifically, in a two-tree  $(T_1, T_2)$ ,  $T_1$  is the upper and  $T_2$  is the bottom tree. In a tree  $T = (V, E)$  with root  $r$ , the notion of a *least common ancestor*  $lca(X)$  of a non-empty set  $X \subseteq V$  is well-defined. Given  $X \subseteq V$ , let the *ancestral tree*  $T \langle X \rangle = (V', E')$  be the given by  $V' = X \cup lca(X)$  and  $xy \in E'$  iff there is a path  $P$  from  $x$  to  $y$  in  $T$  and no other vertex  $z \in V'$  is on  $P$ . Conversely, for  $x \in V$ , the *descendants tree*  $T[x]$  is the graph induced by all vertices  $y$  of  $T$  such that the path from  $y$  to the root  $r$  contains  $x$ . Furthermore, we may talk about the *left* and the *right child* of an inner node  $x$ , written  $l_x$  and  $r_x$ , respectively.

**Results.** (1) We improve on the dynamic programming approach exhibited in [6] to solve OTCM in time  $\mathcal{O}(n \log^2 n)$ . (2) We give an linear-time algorithm for

the drawability test. (3) We prove  $\mathcal{NP}$ -completeness for TTCM. (4) We show in the main part that TTCM is fixed-parameter tractable.

## 2 The OTCM Problem

Let  $(T_1, T_2; M)$  be a two-tree with a fixed suitable order  $<_1$  on  $L(T_1)$ . The task is to find a suitable order  $<_2$  on  $L(T_2)$  that minimizes the number of crossings of matched edges. We are going to give a dynamic programming solution to OTCM. Therefore, notice that any inner node  $v$  of  $T_2$  defines a subproblem in the following sense: let  $L$  be the leaves from  $T_1$  that are matched to leaves from  $L(T_2[v])$ ; then, consider the two-tree  $(T_1, T_2)[v] = (T_1 \langle L \rangle, T_2[v], M \cap (L \times L(T_2[v])))$ , with the order  $<_1$  restricted to  $L$ . For an inner node  $v$  of  $T_2$ ,  $cr(T_2(l_v, r_v))$  denotes the number of pairwise crossings of the matching edges incident with leaves from  $L(T_2[l_v])$  and  $L(T_2[r_v])$ . Note that the total number of crossings for a certain embedding can be expressed as  $\sum_v cr(T_2(l_v, r_v))$ . Hence, we can express the minimum crossing number by the following recursion:

$$cr((T_1, T_2)[v], <_1, \cdot) = cr((T_1, T_2)[l_v], <_1, \cdot) + cr((T_1, T_2)[r_v], <_1, \cdot) + \min\{cr(T_2(l_v, r_v)), cr(T_2(r_v, l_v))\}$$

This recursion can be solved in a naive way in time  $O(n^2)$ . We give a sketch of the proof and show how to improve this time complexity.

Firstly, we show how to compute  $cr(T_2(l_v, r_v))$ .  $cr(T_2(l_v, r_v))$  can be expressed as the sum over all  $\ell \in L(T_2[l_v])$ , where each term gives the number of  $r \in L(T_2[r_v])$  to the left of  $\ell$ ; we call this number the *rank* of  $\ell$ . This sum can be determined by a simple sweep in time linear in  $|L(T_2[v])|$ . Since the subtrees  $T_2[l_v]$  and  $T_2[r_v]$  are disjoint, the total time complexity is  $O(n^2)$ , where  $n$  is the number of leaves.

Alternatively, we can keep the leaves in a balanced search tree. We have to determine the rank of the leaves in  $L(T_2[r_v])$  within  $L(T_2[l_v])$ . For each single leaf in  $L(T_2[r_v])$  this can be done in time  $\mathcal{O}(\log n)$ . Hence, it takes  $\mathcal{O}(|L(T_2[r_v])| \log n)$  time. Vice-versa we can compute the ranks of the leaves from  $L(T_2[l_v])$  within  $L(T_2[r_v])$  in time  $\mathcal{O}(|L(T_2[l_v])| \log n)$ . Since both sums are of the same type, we can choose the cheaper alternative and solve the recursion in time  $T(|L(T_2[v])|) = T(|L(T_2[l_v])|) + T(|L(T_2[r_v])|) + \min\{|L(T_2[l_v])|, |L(T_2[r_v])|\} \log n$ . By induction, we can easily prove that  $T(n) = \mathcal{O}(n \log^2 n)$ .

**Theorem 1.** *In time  $\mathcal{O}(n \log^2 n)$ , we can solve the OTCM problem, where  $n$  is the number of leaves.*

## 3 An Efficient Algorithm for the Non-crossing Case

**Theorem 2.** *Given a two-tree  $(T_1, T_2; M)$ , its drawability can be decided in linear time.*

*Proof.* The two input trees together with the matching edges can be naturally directed upward having the two roots as single source and sink respectively. Then, we can directly apply the linear time algorithm for upward planarity of acyclic digraphs with a single source [2]. ■

## 4 The General Case

**Theorem 3.** TWO-TREE CROSSING MINIMIZATION is  $\mathcal{NP}$ -complete.

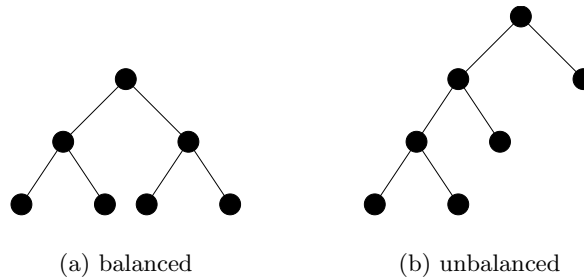
*Proof.* Membership in  $\mathcal{NP}$  is clear. Next, we give a reduction of the MAXCUT problem with unit weights. The MAXCUT problem is to partition the vertex set  $V$  into  $V_1$  and  $V_2$  for a given graph  $G = (V, E)$ , such that  $|\{e = (v, w) \in E \text{ with } v \in V_1 \text{ and } w \in V_2\}|$  is maximized.

So, let  $G = (V, E)$  with  $V = \{1, \dots, n\}$  be an instance of MAXCUT. From  $G$ , we construct an instance of the TWO-TREE CROSSING MINIMIZATION problem, so that we have a 'backbone'-path  $a_1, b_1, \dots, a_n, b_n, C$  in both of the trees  $T_1, T_2$ , which, with the leaf-layers, partitions the drawing area into four parts that later give us the membership of each vertex to  $V_1$  or  $V_2$ , respectively. Let  $a_1$  be the root in each tree,  $C$  is one of the leaves. For each vertex  $v \in V$ , we connect two representative nodes  $a'_v, b'_v$  to the corresponding backbone nodes  $a_v, b_v$  in each tree. Moreover, to each of those representative nodes  $a'_v, b'_v$ , we have to connect further representatives in the leaf-layer, say  $A_v, B_v$ ;  $A_v$ -leaves shall be matched to the  $B_v$ -leaves of the other tree and vice versa with  $n^5$  edges for each  $v = 1, \dots, n$ . For each edge  $e = \{v, w\} \in E$ , we create leaf-vertices  $a_{v,w}$  and  $a_{w,v}$  and connect both pairwise by matching edges. Furthermore, we connect  $a'_v$  with the leaf nodes  $a_{v,w}$  for all  $e = \{v, w\} \in E$ . We also connect the two  $C$  leaves of the backbone to each other via  $n^7$  edges. To make the trees binary, every vertex that is connected to more than one other vertex is substituted by a small binary subtree with an appropriate number of leaves. We can observe the following facts:

1. In any optimum solution, there is no crossing between edges adjacent to  $C$ -nodes and  $(A_v - B_v)$ -edges.
2. In any optimum solution, the  $a'_v$ - and  $b'_v$ - vertices are on different sides of the backbone in both trees. The side for  $a'_v$  in  $T_1$  is different from that in  $T_2$ .
3. In any optimum solution, the number of edges  $(a_{v,w}, a_{w,v})$  crossing the backbone connection is minimized, and the number of edges  $(a_{v,w}, a_{w,v})$  which do not cross the backbone is maximized.

Now, we define  $V_1 = \{i \mid a_v \text{ is on the left-hand side of the backbone in } T_1\}$  and  $V_2 = \{i \mid a_v \text{ is on the right-hand side of the backbone in } T_1\}$  for splitting  $V$  into two disjoint sets  $V_1$  and  $V_2$ . From our observations, we can see that this is an optimal solution for the MAXCUT problem. Obviously, this construction can be built in polynomial time. Hence the reduction of MAXCUT to TTCM is completed. ■

Parameterized complexity and algorithmics is now an established way of dealing with hard problems that have a natural parameter in its definition. The idea is that, for small parameter values, we can get away with a polynomial-time algorithm, where the degree of the polynomial is independent of the parameter. This is the approach we take in the next section.



**Fig. 1.** The two types of trees to be analyzed.

## 5 Fixed-Parameter Tractability

A *parameterized problem*  $P$  is a subset of  $\Sigma^* \times \mathbb{N}$ , where  $\Sigma$  is a fixed alphabet and  $\mathbb{N}$  is the set of all non-negative integers. Therefore, each instance of the parameterized problem  $P$  is a pair  $(I, k)$ , where the second component  $k$  is called the *parameter*. The language  $L(P)$  is the set of all YES-instances of  $P$ . We say that the parameterized problem  $P$  is *fixed-parameter tractable* [5] if there is an algorithm that decides whether an input  $(I, k)$  is a member of  $L(P)$  in time  $f(k)|I|^c$ , where  $c$  is a fixed constant and  $f(k)$  is a recursive function independent of the overall input length  $|I|$ . The class of all fixed-parameter tractable problems is denoted by  $\mathcal{FPT}$ . Ignoring the polynomial part of a parameterized algorithm, we write  $\mathcal{O}^*(f(k))$  to indicate the non-polynomial running time estimate.

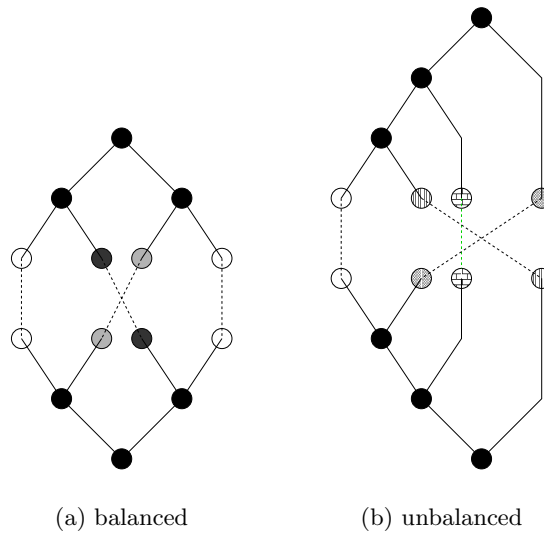
### 5.1 Quadruple Trees

To establish our results, we need a complete analysis of what happens if we restrict ourselves to the case of two trees  $T_1$  and  $T_2$  each having four leaves labeled  $a, b, c, d$ . We will also refer to such small trees as *quadruple trees* and the four leaf labels are then called a *quadruple*. Since our trees  $T = T_i$  are binary, we can have only two different cases:  $T$  has depth two and is hence *balanced*;  $T$  has depth three and is *unbalanced*, see Fig 1. We will show that there are only two types of non-drawable quadruple trees as depicted in Fig. 2.

Let us first analyze the balanced trees. If we assume a labeling  $a, b, c, d$  in this sequence along the leaves of a balanced tree (in the sequence as shown in

Fig. 1), then let  $[abcd]$  denote the different labelings of leaves that can be obtained by different drawings of that particular tree, i.e., by redefining (swapping) the left/right child relations in that tree. We can observe the following possibilities, where  $B_1 \cap B_2 = B_1 \cap B_3 = B_2 \cap B_3 = \emptyset$ :

- $B_1 = [abcd] = \{abcd, abdc, bacd, badc, cdab, cdba, dcab, dcba\}$ ,
- $B_2 = [acbd] = \{acbd, acdb, bdac, bdca, cabd, cadb, dbac, dbca\}$ , and
- $B_3 = [adbc] = \{adbc, adcb, bcad, bcda, cbad, cbda, dabc, dacb\}$ .



**Fig. 2.** The two types of contradicting quadruple two-trees.

**Lemma 1.**  $B_1, B_2, B_3$  are the three mutually disjoint equivalence classes of leaf labelings for balanced trees that together exhaust all 24 permutations of  $a, b, c, d$ .

If we assume that both  $T_1$  and  $T_2$  are balanced quadruple trees, then we can conclude from Lemma 1:

**Corollary 1.** Let  $T_1$  and  $T_2$  be balanced quadruple trees. Let  $L_i^1$  ( $L_i^2$ , resp.) be the two two-element label sets that label leaves reachable from the first (second, resp.) child of the root of  $T_i$ . Then, the pair  $(T_1, T_2)$  can be drawn without crossings iff  $L_1^1 = L_2^1$  or  $L_1^1 = L_2^2$ . Hence, non-drawability in the balanced quadruple tree case means that, with  $L_1^1 = \{x, y\}$ ,  $x \in L_2^1$  and  $y \in L_2^2$ .

The unbalanced quadruple trees can be also treated with a case-by-case analysis.

**Corollary 2.** Let  $T_1$  and  $T_2$  be unbalanced quadruple trees. Assume (w.l.o.g.) that  $T_1$  and  $T_2$  are drawn as in Fig. 1 (of course,  $T_2$  must be flipped to get the

root at the bottom), with labels  $a, b, c, d$  attached to the leaves of  $T_1$  in that order. Then, the pair  $(T_1, T_2)$  is drawable iff either the third leaf of  $T_2$  is not labeled  $c$  or the last leaf of  $T_2$  is not labeled with one of the first two labels of  $T_1$ .

Hence, such a pair cannot be drawn iff the third leaf coincides and the last leaf of  $T_2$  is among the first two leaves of  $T_1$ . Comparing the classes of unbalanced trees with those of balanced trees, one can see:

**Corollary 3.** *Let  $T_1$  and  $T_2$  be quadruple trees, where  $T_1$  is balanced and  $T_2$  is unbalanced (or vice versa). Then, the pair  $(T_1, T_2)$  is drawable.*

### 5.2 Quadruples Are All About Inconsistencies

The following theorem shows that the drawability of a two-tree only depends on the drawability of the induced quadruple trees.

**Theorem 4.** *Let  $(T_1, T_2; M)$  be a two-tree with labelings  $\lambda_i : L(T_i) \rightarrow \Lambda$ . If  $cr(T_1, T_2, M, \cdot, \cdot) > 0$ , then there exists a quadruple  $Q = \{a, b, c, d\} \subseteq \Lambda$  such that  $cr(T_1 \langle \lambda_1^{-1}(Q) \rangle, T_2 \langle \lambda_2^{-1}(Q) \rangle, M \cap (\lambda_1^{-1}(Q) \times \lambda_2^{-1}(Q)), \cdot, \cdot) > 0$ .*

We will present a recursive algorithm which either finds such an quadruple, or it provides a drawing of the two-tree. This algorithm will not only prove this structural result that gives a characterization of drawable two-trees in terms of forbidden substructures, but also provides the backbone of two  $\mathcal{FPT}$  algorithms that are presented in the following.

*Proof.* Let  $(T_1, T_2; M)$  be the two-tree and  $\Lambda$  the set of labels associated to the leaves. W.l.o.g.,  $|\Lambda| > 1$ .

We assume that each inner node provides links to its two children and in addition permanent and temporary information attached to each such link. The permanent information  $p(\ell)$  attached to link  $\ell$  is either  $L, R$  or  $*$ , meaning:

$L$	This link leads to the left child.
$R$	This link leads to the right child.
$*$	It is not yet determined if this link leads to the left or to the right child.

The permanent information indicates a commitment how the links are to be drawn (either forced or without loss of generality) in order to obtain a crossing-free embedding of the two-tree; once fixed, it will not be changed in later stages of the algorithm. The temporary information is only used to either produce further evidence that allows to make further commitments of the permanent information or to provide a contradictory quadruple.

In the initialization phase of our algorithm, we (arbitrarily) set  $p(\ell_1) = L$  and  $p(\ell_2) = R$  for the two links emanating from the root of  $T_2$ . All other permanent information is set to  $*$ . This defines the function  $p$  (that we use both for  $T_1$  and for  $T_2$ ). Thus initiated, we call  $embed(T_1, T_2, p)$ . The way how the permanent information is initiated and updated shows that the following is always true:

**Claim 1:** Let  $n$  be an inner node with emanating links  $\ell_1$  and  $\ell_2$ . Then,  $p(\ell_1) = L$  iff  $p(\ell_2) = R$ , and  $p(\ell_1) = R$  iff  $p(\ell_2) = L$ .

The temporary information  $t(\ell)$  attached to link  $\ell$  is either  $l, r$  or  $m$ , meaning:

$l$	All links $\ell'$ below (i.e., in direction to the leaves) satisfy $t(\ell') = l$ .
$r$	All links $\ell'$ below satisfy $t(\ell') = r$ .
$m$	Mixed case: some links below are marked $l$ and some are marked $r$ .

The temporary information is processed bottom-up as follows:

1. As described in Alg. 1 in detail, the links leading to the leaves of the tree to be processed are assigned either  $l$  or  $r$ .
2. Let  $\mathbf{n}$  be an inner node (besides the root) where to both links  $\ell_1$  and  $\ell_2$  emanating to its two children, the temporary information has been assigned, such that the previous situation does not apply. Then, to the link  $\ell$  that leads to  $\mathbf{n}$  we assign  $t(\ell)$  according to the following table:

$$\begin{array}{c} t(\ell_1) \\ t(\ell_2) \\ \hline t(\ell) \end{array} \left\| \begin{array}{c|c|c} l & l & l \\ l & r & m \\ \hline l & m & m \end{array} \right| \begin{array}{c|c|c} r & r & r \\ l & r & m \\ \hline m & r & m \end{array} \left\| \begin{array}{c|c|c} m & m & m \\ l & r & m \\ \hline m & m & e \end{array} \right.$$

Here,  $e$  signals an error case: we have found a quadruple situation corresponding to the balanced tree case in Fig. 2. Hence, there is no way of finding a crossing-free embedding of the two-trees, and we can abort here.

Interestingly, we can also update the permanent information of two siblings. Let  $\mathbf{n}$  be an inner node where to both links  $\ell_1$  and  $\ell_2$  emanating to its two children, the temporary information has been assigned. We update  $p(\ell_1)$  and  $p(\ell_2)$  according to the following table:

$$\begin{array}{c} t(\ell_1) \\ t(\ell_2) \\ \hline p(\ell_1) \\ p(\ell_2) \\ \hline p(\ell_1) \\ p(\ell_2) \end{array} \left\| \begin{array}{c|c|c} l & l & l \\ l & l & l \\ \hline L & R & * \\ R & L & * \\ \hline L & R & * \\ R & L & * \end{array} \right| \begin{array}{c|c|c} l & l & l \\ r & r & r \\ \hline L & R & * \\ R & L & * \\ \hline L & E & L \\ R & E & R \end{array} \left\| \begin{array}{c|c|c} l & l & l \\ m & m & m \\ \hline L & R & * \\ R & L & * \\ \hline L & E & L \\ R & E & R \end{array} \right.$$

Observe that there are more cases for assigning temporary information, with the roles of  $\ell_1$  and  $\ell_2$  being interchanged. Furthermore, notice that the list of cases of assignments to  $\ell_1$  and  $\ell_2$  is complete with respect to the permanent information because of Claim 1. The table should be read as follows: the first four lines give the current values of  $t$  and  $p$  on the two links. The last two lines give the updated values of  $p$ . Here, an  $E$  signals that we found a contradictory situation; more specifically (as we will see below), we have found a quadruple situation corresponding to the unbalanced tree case in Fig. 2 Hence, there is no way of finding a crossing-free embedding of the two-trees, and we can abort here. Claim 2: Observe that the graph that is induced by the edges (links) to which non-\* permanent information has been attached to is a tree before and after each complete bottom-up tree processing (as described above). Moreover, if this induced tree is non-empty, then it also contains the root.

How to actually use the bottom-up processing of the temporary and permanent information is explained in Alg. 1. Observe that the roles of the trees alternate. We will make use of the following property of our algorithm:

**Claim 3:** Each time that it is again say the upper tree's turn to get new temporary labels, the former root of that tree (and possibly more nodes) will no longer be taken into consideration.

We still have to show that the two aborts (error cases) described above are indeed based on finding a contradictory quadruple two-tree as explained in Fig. 2. The (omitted) proofs of the following claims show the details of this construction.

---

**Algorithm 1** Procedure “embed-TT”

---

**Require:** A two-tree  $(T_1, T_2; M)$  and a permanent link information function  $p$ .  
**Ensure:** YES iff a crossing-free drawing of  $(T_1, T_2; M)$  respecting  $p$  can be obtained. Moreover, either (implicitly) a crossing-free drawing of  $(T_1, T_2; M)$  respecting  $p$  is found or a contradictory quadruple two-tree is produced.

Let  $r$  be the root of  $T_2$ .  
**if**  $(T_1, T_2; M)$  has at most four leaves **then**  
    return answer by table look-up (produced according to Subsec. 5.1)  
**else if**  $r$  has only one child **then**  
5: Delete  $r$  to produce  $T'_2$ .  
    Modify  $p$  and  $M$  accordingly, yielding  $p'$  and  $M'$ .  
    return embed-TT( $T_1, T'_2, M', p'$ );  
**else**  
    {Let  $\ell_1$  and  $\ell_2$  be the two links emanating from the root  $x$  of  $T_2$ .}  
10: **if**  $p(\ell_1) = *$  **then**  
     $p(\ell_1) := L$  and  $p(\ell_2) := R$  (without loss of generality).  
    **end if**  
    {Let  $\ell_L = xy_L$  with  $p(\ell_L) = L$  and  $\ell_R = xy_R$  with  $p(\ell_R) = R$ .}  
    Let  $L_L := L(T_2[y_L])$  and  $L_R := L(T_2[y_R])$ .  
15: Let  $L_l = \{u \in L(T_1) \mid \exists u' \in L_L : (u, u') \in M\}$  and  $L_r = L(T_1) \setminus L_l$ .  
    **for all** links  $\ell = uv$  of  $T_1$  where  $v$  is closer to the root than  $u$  **do**  
        **if**  $u \in L(T_1)$  **then**  
             $t(\ell) := z \in \{l, r\}$  such that  $u \in L_z$ .  
        **else**  
20:          $t(\ell) := *$   
        **end if**  
    **end for**  
    Update the temporary and permanent information within  $T_1$  as in the proof.  
    **if** contradiction is reached **then**  
25:         Report contradictory quadruple two-tree (see the proof).  
        return NO  
    **else**  
        Let  $p_L$  be the permanent information  $p$  updated to cover the two-tree  $(T_2[y_L], T_1 \langle L_l \rangle; M_L)$  with  $M_L^{-1} = M \cap (L_l \times L_L)$ ; similarly, define  $p_R, M_R$ .  
        return embed-TT( $T_2[y_L], T_1 \langle L_l \rangle, M_L, p_L$ )  $\wedge$  embed-TT( $T_2[y_R], T_1 \langle L_r \rangle, M_R, p_R$ )  
30:         **end if**  
    **end if**

---

**Claim 4:** Whenever an error occurs within the temporary label information update, we can exhibit a balanced quadruple two-tree.

**Claim 5:** Whenever a contradiction is found between the temporary label information and the already existent permanent label information, we can exhibit an unbalanced quadruple two-tree. ■

Theorem 4 shows an immediate result for the following problem that is closely related to TTCM: An instance of TWO-TREE DRAWING BY DELETING EDGES (TTDE) is given by a two-tree  $(T_1, T_2; M)$ , and the parameter, a positive integer  $k$ . Question: Is there a set  $M' = L'_1 \times L'_2 \subseteq M$  with  $|M'| \leq k$  such that the two-tree  $(T_1 \langle L(T_1) \setminus L'_1 \rangle, T_2 \langle L(T_1) \setminus L'_2 \rangle)$  is drawable? Namely, we can translate any TWO-TREE DRAWING BY DELETING EDGES instance into 4-HITTING SET: simply cycle through all  $\mathcal{O}(n^4)$  possible quadruple two-trees (given a concrete two-tree  $(T_1, T_2)$  with  $n$  leaves): If a quadruple two-tree is contradictory, then it corresponds to a hyperedge with four vertices, the leaf labels forming that quadruple. All  $n$  leaf labels together are the vertices of the hypergraph. Using known parameterized algorithms for 4-HITTING SET, see [10], we can thus show:

**Corollary 4.** TTDE is solvable in  $\mathcal{O}^*(3.115^k)$  time.

Let us mention that we can similarly translate the variant ONE-TREE DRAWING BY DELETING EDGES of TTDE where one tree is fixed into 3-HITTING SET.

## 6 TWO-TREE CROSSING MINIMIZATION Is in $\mathcal{FPT}$

This result is heavily based on the structural results of the previous section. More precisely, we will sketch a parameterized algorithm that branches on small contradicting structures (primarily, at contradicting quadruples) as long as these incur new crossings. In a second phase, we attempt at drawing the remaining two-tree with using a variant of the algorithm embed, possibly finding new small contradicting structures. The validity of this approach relies on the fact that we are able to separate contradicting structures from the rest of the two-tree by attachment links that are described as follows.

Let  $u$  be a node in a tree  $T$ . The parent link  $\mathbf{p}(u)$  is the unique edge leading to  $u$  in  $T$ ; if  $u$  is the root, then there is no parent link to  $u$ . Let  $V' \subseteq V(T)$ . The set of all parent links of  $V'$  is  $\mathfrak{P}(V') = \{\mathbf{p}(u) \mid u \in V'\}$ . The set of *attachment links* of a subgraph  $G$  of  $T$  is  $\mathfrak{A}(G) = \mathfrak{P}(\text{lca}(V(G)))$ .

As above, we work with the permanent information  $p(\ell)$  attached to an edge  $\ell$ , initialized with  $p = *$  and later gradually updated to either  $L$  or  $R$ . Sometimes, it is more convenient to think of this labeling information being attached to the inner nodes in the sense that a bit (called *flip*) is associated with each inner node that tells, when defined, which child is to the left and which is to the right.

Given a two-tree  $(T_1, T_2; M)$  with labelings  $\lambda_i$ , our algorithm will basically branch on all possible settings of  $p$  to either  $L$  or  $R$  on the attachment links  $\mathfrak{A}(G_i)$  (that do not contradict earlier settings  $\neq *$ ) for the subgraphs  $G_i = T_i \langle \lambda_i^{-1}(Q) \rangle$  of  $T_i$  for all possible contradicting quadruples  $Q$ . Observe that whenever a parent link of some node is assigned  $L$ , then the parent link of its sibling will be assigned  $R$  for reasons of consistency (and vice versa).

The problem we are facing is that we have to ensure that the natural parameter of this problem, i.e., the given budget  $k$  of tolerable crossings, is decremented in each branching step. So, our strategy will be to only branch at contradicting structures if this gives us a gain in each branch. To simplify matters, we assume that only those leaves that participated in those contradicting structures we earlier branched on have been accounted for in the branching process.

As *contradicting structures*, we will view contradicting quadruples (as before) and contradicting pairs, i.e., pairs of labels  $a, b$  where say the upper tree fixes  $a < b$  and the lower tree fixes  $b < a$  (due to the flips that are fixed in both trees).

---

**Algorithm 2** Sketch of procedure “embed-TTCM”

---

**Require:** A two-tree  $(T_1, T_2; M)$ ; permanent link information  $p$ ; a parameter  $k$ .

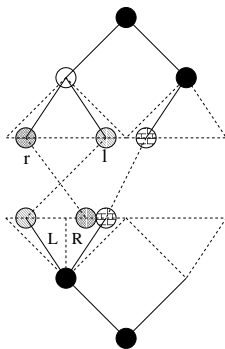
**Ensure:** YES iff  $(T_1, T_2; M)$  can be drawn with  $\leq k$  crossings so that  $p$  is respected.

```

if  $k < 0$  then
    return NO
else if  $k = 0$  then
    return embed-TT( $T_1, T_2, M, p$ ) {see to Alg. 1}
5: else
    if there is a small contradicting structure  $S$  then
        branch on all possible flips for  $\mathfrak{A}(T_i \langle S \rangle)$  with recursive calls on embed-TTCM,
        where  $S$  is deleted from the new instances and  $M, p$  are accordingly modified.
    else
        return embed-TT'( $T_1, T_2, M, p, k$ )
10: { $k$  is only needed if embed-TT'' recursively calls embed-TTCM.}
    end if
end if

```

---



To fully understand Alg. 2, we still have to explain what to do when there are no longer contradicting quadruples or pairs to be found (line 9): we will then start a procedure very similar to embed-TT. The only difference is that it will not find any of the contradictions described in the proof of Theorem 4 (since there are no contradicting quadruples). Now, a temporary labeling could contradict the already established permanent labeling. In that case, the permanent labeling would already exist upon first calling embed-TT', so that we face the situation depicted in the figure to the left. This figure is understood as follows: the white inner node indicates a flip that has been determined.

As the labels L and R indicate, this flip would go the other way round according to the temporary link information propagation. The permanent labeling must have a reason (otherwise, we could just interchange the meaning of L and R):

namely, there must be a third leaf (the brick-pattern label) that is residing somewhere in the “right branch” of the upper tree but in the “left branch” in the bottom tree (on a previous level of recursion). Hence, we also get a crossing if we draw the left part of bottom tree coherently with the “white flip.” Upon finding such an erroneous situation, we would consider the depicted leaves as a contradicting structure and branch on all attachment points as before, recursively calling embed-TTCM again.

**Theorem 5.** TWO-TREE CROSSING MINIMIZATION is in  $\mathcal{FPT}$ . More precisely, the problem can be solved in time  $\mathcal{O}^*(c^k)$  for some constant  $c$ .

Unfortunately, the constant  $c$  is still quite huge (about  $2^{10}$ ) due to the many attachment links whose combinations must be tested. We can see our result therefore as a preliminary one, mainly providing a classification of the problem.

## 7 Conclusion

We have considered two-layer crossing minimization problems for the purpose of comparing two unordered trees. We derived  $\mathcal{NP}$ -completeness results and efficient polynomial-time algorithms as well as  $\mathcal{FPT}$ -algorithms. The following open problems are worthwhile to consider: (1) Determine the (parameterized) complexity of the maximum planar submatching variant TTDE and prove  $\mathcal{NP}$ -completeness. (2) Extend the techniques to  $d$ -ary trees. From the first sight, an additional factor of  $d!$  shows up. This problem has clustering applications, see [1]. (3) Consider the weighted version of TTCM, where the crossings have higher weights if they occur between edges of larger different subtrees. (4) The connections to HITTING SET we exhibited for TTDE provides a factor-4 approximation; we do not see any constant-factor approximation for TTCM.

*Acknowledgment:* Thanks to G. Liotta and D. Huson for motivation and help.

## References

1. Z. Bar-Joseph, D. Gifford, and T. Jaakkola. Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics*, 17:22–29, 2001.
2. P. Bertolazzi, G. Di Battista, G. Liotta and C. Mannino. Optimal upward planarity testing of single-source digraphs. *SIAM Journal of Computing*, 27:132–169, 1998.
3. P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337:217–239, 2005.
4. G. Di Battista, Eades P., Tamassia R. and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
5. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
6. T. Dwyer and F. Schreiber. Optimal leaf ordering for two and a half dimensional phylogenetic tree visualisation. In: *Proc. Australasian Symp. on Information Visualisation (InVis.au 2004)*, CRPIT 35:109–115, 2004.
7. P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 13:361–374, 1994.

8. P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 10:379–403, 1994.
9. M. R. Fellows. The Robertson-Seymour theorems: a survey of applications. *Contemp. Math.*, 89:1–18, 1989.
10. H. Fernau. *Parameterized Algorithmics: A Graph-Theoretic Approach*. Habilitationsschrift, Universität Tübingen, Germany, 2005. Submitted.
11. D. Huson. Private Communication, 2005.
12. T. Jiang, L. Wang, and K. Zhang. Alignment of trees—an alternative to tree edit. *Theoretical Computer Science*, 143:137–148, 1995.
13. P. Kilpeläinen and H. Mannila. Ordered and unordered tree inclusion. *SIAM Journal of Computing*, 24:340–356, 1995.
14. P. Klein, S. Tirthapura, D. Sharvit, and B. Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In: *Proc. of 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 696–704, 2000.
15. R. Ramesh and I. Ramakrishnan. Nonlinear pattern matching in trees. *Journal of the ACM*, 39:295–316, 1992.
16. K. Zhang and D. Shasha. Simple fast algorithms for the edit distance between trees and related problems *SIAM Journal of Computing*, 18:1245–1262, 1989.