Drawing (Complete) Binary Tanglegrams:
Hardness, Approximation, Fixed-Parameter Tractability*

Kevin Buchin[1**], Maike Buchin[1**], Jaroslaw Byrka[2,3], Martin Nöllenburg[4***], Yoshio Okamoto[5†], Rodrigo I. Silveira[1**], and Alexander Wolff[2]

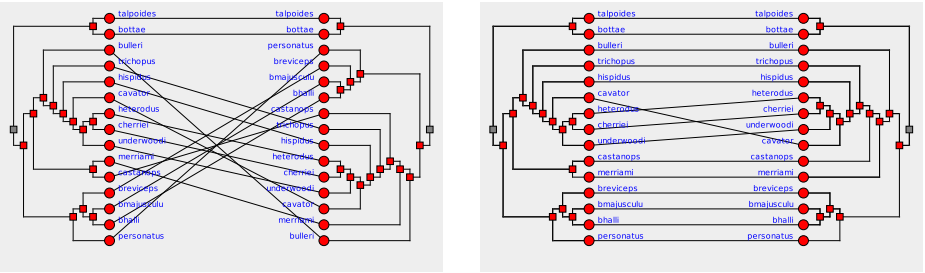
[1] Dept. Computer Science, Utrecht University, The Netherlands.
{buchin, maike, rodrigo}@cs.uu.nl
[2] Faculteit Wiskunde en Informatica, TU Eindhoven, The Netherlands.
http://www.win.tue.nl/algo
[3] Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands.
j.byrka@cwi.nl
[4] Fakultät für Informatik, Universität Karlsruhe, Germany.
noellenburg@iti.uka.de
[5] Grad. School of Infor. Sci. and Engineering, Tokyo Inst. of Technology, Japan.
okamoto@is.titech.ac.jp

Abstract. A *binary tanglegram* is a pair $\langle S, T \rangle$ of binary trees whose leaf sets are in one-to-one correspondence; matching leaves are connected by inter-tree edges. For applications, for example in phylogenetics, it is essential that both trees are drawn without edge crossings and that the inter-tree edges have as few crossings as possible. It is known that finding a drawing with the minimum number of crossings is NP-hard and that the problem is fixed-parameter tractable with respect to that number. We prove that under the Unique Games Conjecture there is no constant-factor approximation for general binary trees. We show that the problem is hard even if both trees are complete binary trees. For this case we give an $O(n^3)$-time 2-approximation and a new and simple fixed-parameter algorithm. We show that the maximization version of the dual problem for general binary trees can be reduced to a version of MaxCut for which the algorithm of Goemans and Williamson yields a 0.878-approximation.

1 Introduction

In this paper we are interested in drawing so-called *tanglegrams* [16], that is, pairs of trees whose leaf sets are in one-to-one correspondence. The need to

* This work was started at the 10th Korean Workshop on Computational Geometry, organized by H. Haverkort and Ch. Knauer in Schloss Dagstuhl, Germany, July 2007.
** Supported by the Netherlands' Organisation for Scientific Research (NWO) under BRICKS/FOCUS project no. 642.065.503 and under the project GOGO.
*** Supported by grant WO 758/4-3 of the German Research Foundation (DFG).
† Partially supported by Grant-in-Aid for Scientific Research and Global COE Program from Ministry of Education, Science and Culture, Japan, and Japan Society for the Promotion of Science.



(a) arbitrary drawing (b) drawing of our 2-approximation

Fig. 1: A binary tanglegram showing two evolutionary trees for pocket gophers [9].

visually compare pairs of trees arises in applications such as the analysis of software projects, phylogenetics, or clustering. In the first application, trees may represent package-class-method hierarchies or the decomposition of a project into layers, units, and modules. The aim is to analyze changes in hierarchy over time or to compare human-made decompositions with automatically generated ones. Whereas trees in software analysis can have nodes of arbitrary degree, trees from our second application, that is, (rooted) phylogenetic trees, are binary trees. This makes binary tanglegrams an interesting special case, see Fig. 1. Hierarchical clusterings, our third application, are usually visualized by a binary tree-like structure called *dendrogram*, where elements are represented by the leaves and each internal node of the tree represents the cluster containing the leaves in its subtree. Pairs of dendrograms stemming from different clustering processes of the same data can be compared visually using tanglegrams.

In this paper we consider binary tanglegrams if not stated otherwise. From the application point of view it makes sense to insist that (a) the trees under consideration are drawn plane (namely, without edge crossings), (b) each leaf of one tree is connected by an additional edge to the corresponding leaf in the other tree, and (c) the number of crossings among the additional edges is minimized. As in the bioinformatics literature (e.g., [13, 16]), we call this the *tanglegram layout* (TL) problem; Fernau et al. [7] refer to it as *two-tree crossing minimization*. Note that we are interested in the minimum number of crossings for visualization purposes. The number is not intended to be a tree-distance measure. Examples for such measures are nearest-neighbor interchange and subtree transfer [3].

Related problems. In graph drawing the so-called *two-sided crossing minimization problem* (2SCM) is an important problem that occurs when computing layered graph layouts. Such layouts have been introduced by Sugiyama et al. [17] and are widely used for drawing hierarchical graphs. In 2SCM, vertices of a bipartite graph are to be placed on two parallel lines (*layers*) such that vertices on one line are incident only to vertices on the other line. As in TL the objective is to minimize the number of edge crossings provided that edges are drawn as straight-line segments. In one-sided crossing minimization (1SCM) the order of

the vertices on one of the layers is fixed. Even 1SCM is NP-hard [6]. In contrast to TL, a vertex in 1SCM or 2SCM can have several incident edges and the linear order of the vertices in the non-fixed layer is not restricted by the internal structure of a tree. The following is known about 1SCM. The median heuristic of Eades and Wormald [6] yields a 3-approximation and a randomized algorithm of Nagamochi [14] yields an expected 1.4664-approximation. Dujmovič et al. [4] gave an FPT algorithm that runs in $O^\star(1.4664^k)$ time, where k is the minimum number of crossings in any 2-layer drawing of the given graph that respects the vertex order of the fixed layer. The $O^\star(\cdot)$-notation ignores polynomial factors.

Previous work. Dwyer and Schreiber [5] studied drawing a series of tanglegrams in 2.5 dimensions, i.e., the trees are drawn on a set of stacked two-dimensional planes. They considered a one-sided version of TL by fixing the layout of the first tree in the stack, and then, layer-by-layer, computing the leaf order of the next tree in $O(n^2 \log n)$ time each. Fernau et al. [7] showed that TL is NP-hard and gave a fixed-parameter algorithm that runs in $O^\star(c^k)$ time, where c is a constant estimated to be 1024 and k is the minimum number of crossings in any drawing of the given tanglegram. They showed that the problem can be solved in $O(n \log^2 n)$ time if the leaf order of one tree is fixed. This improves the result of Dwyer and Schreiber [5]. They also made the simple observation that the edges of the tanglegram can be directed from one root to the other. Thus the existence of a planar drawing can be verified using a linear-time upward-planarity test for single-source directed acyclic graphs [1]. Later, apparently not knowing these previous results, Lozano et al. [13] gave a quadratic-time algorithm for the same special case, to which they refer as *planar tanglegram layout*. Holten and van Wijk [10] presented a visualization tool for general tanglegrams that heuristically reduces crossings (using the barycenter method for 1SCM on a per-level base) and draws inter-tree edges in bundles (using Bézier curves).

Our results. Let us call the restriction of TL to (complete) binary trees the *(complete) binary TL problem*. We first analyze the complexity of binary TL, see Sect. 2. We show that binary TL is essentially as hard as the MinUncut problem. If the (widely accepted) Unique Games Conjecture holds, it is NP-hard to approximate MinUncut—and thus TL—within any constant factor [12]. This motivates us to consider complete binary TL. It turns out that this special case has a rich structure. We start our investigation by giving a new reduction from Max2Sat that establishes the NP-hardness of complete binary TL.

The main result of this paper is a simple recursive factor-2 approximation algorithm for complete binary TL, see Sect. 3. It runs in $O(n^3)$ time and extends to d-ary trees. Our algorithm can also process general binary tanglegrams—without guaranteeing any approximation ratio. It works very well in practice and is quite fast when combined with a branch-and-bound procedure [15].

Next we consider a dual problem: maximize the number of edge pairs that do *not* cross. We show that this problem (for *general* binary trees) can be reduced to a version of MaxCut for which the algorithm of Goemans and Williamson yields a 0.878-approximation.

Finally, we investigate the parameterized complexity of complete binary TL. Our parameter is the number k of crossings in an optimal drawing. We give a new FPT algorithm for complete binary TL that is much simpler and faster than the FPT algorithm for *general* binary TL by Fernau et al. [7]. The running time of our algorithm is $O(4^k n^2)$, see Sect. 4. An interesting feature of the algorithm is that the parameter does *not* drop in each level of the recursion.

Formalization. We denote the set of leaves of a tree T by $L(T)$. We are given two rooted trees S and T with n leaves each. We require that S and T are *uniquely leaf-labeled*, that is, there are bijective labeling functions $\lambda_S : L(S) \to \Lambda$ and $\lambda_T : L(T) \to \Lambda$, where Λ is a set of labels, for example, $\Lambda = \{1, \ldots, n\}$. These labelings define a set of new edges $\{uv \mid u \in L(S), v \in L(T), \lambda_S(u) = \lambda_T(v)\}$, the *inter-tree edges*. The TL problem consists of finding plane drawings of S and T that minimize the number of induced crossings of the inter-tree edges, assuming that edges are drawn as straight-line segments. We insist that the leaves in $L(S)$ are placed on the line $x = 0$ and those in $L(T)$ on the line $x = 1$. The trees S and T themselves are drawn to the left of $x = 0$ and to the right of $x = 1$, respectively. For an example see Fig. 1. Given uniquely leaf labeled trees S and T, we denote the resulting instance of TL by $\langle S, T \rangle$.

The TL problem is purely combinatorial: Given a tree T, we say that a linear order of $L(T)$ is *compatible* with T if for each node v of T the nodes in the subtree of v form an interval in the order. Given a permutation π of $\{1, \ldots, n\}$, we call (i, j) an *inversion* in π if $i < j$ and $\pi(i) > \pi(j)$. For fixed orders σ of $L(S)$ and τ of $L(T)$ we define the permutation $\pi_{\tau,\sigma}$, which for a given position in τ returns the position in σ of the leaf having the same label. Now the TL problem consists of finding an order σ of $L(S)$ compatible with S and an order τ of $L(T)$ compatible with T such that the number of inversions in $\pi_{\tau,\sigma}$ is minimum.

2 Complexity

In this section we consider the complexity of binary TL, which Fernau et al. [7] have shown to be NP-complete for general binary tanglegrams. We strengthen their findings in two ways. First, we show that it is unlikely that an efficient constant-factor approximation for general binary TL exists. Second, we show that TL remains hard even when restricted to *complete* binary tanglegrams.

We start by showing that binary TL is essentially as hard as the MinUncut problem. This relates the existence of a constant-factor approximation for TL to the Unique Games Conjecture (UGC) by Khot [11]. The UGC became famous when it was discovered that it implies optimal hardness-of-approximation results for problems such as MaxCut and VertexCover, and forbids constant factor-approximation algorithms for problems such as MinUncut and SparsestCut. We reduce the MinUncut problem to the TL problem, which, by the result of Khot and Vishnoi [12], makes it unlikely that an efficient constant-factor approximation for TL exists.

The MinUncut problem is defined as follows. Given an undirected graph $G = (V, E)$, find a partition (V_1, V_2) of the vertex set V that minimizes the

number of edges that are not cut by the partition, that is, $\min_{(V_1,V_2)} |\{uv \in E : u, v \in V_1 \text{ or } u, v \in V_2\}|$. Note that computing an optimal solution to MinUncut is equivalent to computing an optimal solution to MaxCut. Nevertheless, the MinUncut problem is more difficult to approximate.

Theorem 1. *Under the Unique Games Conjecture it is NP-hard to approximate the TL problem for general binary trees within any constant factor.*

Proof. As mentioned above we reduce from the MinUncut problem. Our reduction is similar to the one in the NP-hardness proof by Fernau et al. [7].

Consider an instance $G = (V, E)$ of the MinUncut problem. We construct a TL instance $\langle S, T\rangle$ as follows. The two trees S and T are identical and there are three groups of edges connecting leaves of S to leaves of T. For simplicity we define multiple edges between a pair of leaves. In the actual trees we can replace each such leaf by a binary tree with the appropriate number of leaves.

Suppose $V = \{v_1, v_2, \ldots, v_n\}$, then both S and T are constructed as follows. There is a *backbone* path $(v_1^1, v_1^2, v_2^1, v_2^2, \ldots, v_n^1, v_n^2, a)$ from the root node v_1^1 to a leaf a. Additionally, there are leaves $l_S(v_i^j)$ and $l_T(v_i^j)$ attached to each node v_i^j for $i \in \{1, \ldots, n\}$ and $j \in \{1, 2\}$ in S and T, respectively. The edges form the following three groups.

Group A contains n^{11} edges connecting $l_S(a)$ with $l_T(a)$.
Group B contains for every $v_i \in V$ n^7 edges connecting $l_S(v_i^1)$ with $l_T(v_i^2)$, and n^7 edges connecting $l_S(v_i^2)$ with $l_T(v_i^1)$.
Group C contains for every $v_iv_j \in E$ a single edge from $l_S(v_i^1)$ to $l_T(v_j^1)$.

Next we show how to transform an optimal solution of the MinUncut instance into a solution of the corresponding TL instance. Suppose that in the optimal partition (V_1^*, V_2^*) of G there are k edges that are not cut. Then we claim that there exists a drawing of $\langle S, T\rangle$ such that $k \cdot n^{11} + O(n^{10})$ pairs of edges cross. It suffices to draw, for each vertex $v_i \in V_1^*$ ($v_i \in V_2^*$), the leaves $l_S(v_i^1)$ and $l_T(v_i^2)$ above (below) the backbones, and the nodes $l_S(v_i^2)$ and $l_T(v_i^1)$ below (above) the backbones. It remains to count: there are $k \cdot n^{11}$ A–C crossings, no A–B crossings, $O(n^{10})$ B–C crossings, and $O(n^4)$ C–C crossings.

Now suppose there exists an α-approximation algorithm for the TL problem with some constant α. Applying this algorithm to the instance $\langle S, T\rangle$ defined above yields a drawing $D(S, T)$ with at most $\alpha \cdot k \cdot n^{11} + O(n^{10})$ crossings. Let us assume that n is much larger than α. We show that from such a drawing $D(S, T)$ we would be able to reconstruct a cut (V_1, V_2) in G with at most $\alpha \cdot k$ edges uncut. First, observe that if a node $l_S(v_i^1)$ is drawn above (below) the backbone in $D(S, T)$, then $l_T(v_i^2)$ must be drawn on the same side of the backbone, otherwise it would result in n^{18} A–B crossings. Similarly $l_S(v_i^2)$ must be on the same side as $l_T(v_i^1)$. Then observe that if a node $l_S(v_i^1)$ is drawn above (below) the backbone in $D(S, T)$, then $l_S(v_i^2)$ must be drawn below (above) the backbone, otherwise there would be $O(n^{14})$ B–B crossings. Finally, observe that if we interpret the set of vertices v_i for which $l_S(v_i^1)$ is drawn above the backbone as a set V_1 of a partition of G, then this partition leaves at most $\alpha \cdot k$ edges from E uncut.

Hence, an α-approximation for the TL problem provides an α-approximation for the MinUncut problem, which contradicts the UGC. □

The above negative result for (general) binary TL is our motivation to investigate the complexity of complete binary TL. It turns out that even this special case is hard. Unlike Fernau et al. [7] who show hardness of binary TL by a reduction from MaxCut using extremely unbalanced trees, we use a quite different reduction from a variant of Max2Sat (see full version for the proof [2]).

Theorem 2. *The TL problem is NP-hard even for complete binary tanglegrams.*

3 Approximation

We now present our main result, a 2-approximation algorithm for complete binary TL that runs in $O(n^3)$ time. The idea is to split a given tanglegram recursively at the roots of the two trees into two subinstances, each again consisting of a pair of complete binary trees. Let $\langle S, T\rangle$ be a subinstance of $\langle S_0, T_0\rangle$ with subtrees $S \subseteq S_0$ and $T \subseteq T_0$ rooted at nodes $v_S \in S_0$ and $v_T \in T_0$, respectively (see Fig. 2). When treating $\langle S, T\rangle$, we use the following pieces of information.

Firstly, associated with v_S and v_T we have labels ℓ_S and ℓ_T that indicate what choices in the recursion so far led to the current subinstances. A label is a bit string that represent the choices (swap/do not swap children) made at each node, from the first recursive step to the current one (see Fig. 3).

We also assign labels to some other subtrees of $\langle S_0, T_0\rangle$ apart from S and T. Given a leaf $v \in T_0 \setminus T$, we define the *largest T-avoiding tree* of v to be the largest complete binary subtree of T_0 that contains v, but not T. Largest S-avoiding trees are defined analogously for leaves in S_0. Each largest S- or T-avoiding tree receives a label in the same way as S and T. Note that the labels of the avoiding trees are relative to the labels of v_S and v_T, that is, a different subinstance leads to different labels. If we refer (in the context of a subinstance $\langle S, T\rangle$) to the label of a leaf $v \in T_0$, we mean the label of the largest T-avoiding tree of v.

Secondly, since S and T are part of a larger tree, some leaves of S may not have the matching leaf in T (and vice versa). This means that at some previous step such leaves were matched to leaves in some other subtrees, above or below $\langle S, T\rangle$. We do not know exactly to which leaves they are matched, but we do know, for each leaf, the label of the subtree that contains the matching leaf.

At each level of the recursion we have to choose between one out of four configurations. Let the current subinstance be given by $\langle S, T\rangle = \langle (S_1, S_2), (T_1, T_2)\rangle$. At each node v_S on the left side, we must choose between having S_1 above S_2 or the other way around. On the right side for v_T, there are also two different ways of placing T_1 and T_2. For each of the four configurations we invoke the algorithm twice recursively: for the top half and for the bottom half. We return the configuration with the smallest number of crossings.

When counting the crossings that a configuration creates, we distinguish two types: *current-level* and *lower-level* crossings.

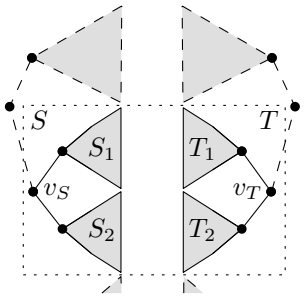


Fig. 2: Context of subinstance $\langle S, T\rangle = \langle (S_1, S_2), (T_1, T_2)\rangle$.

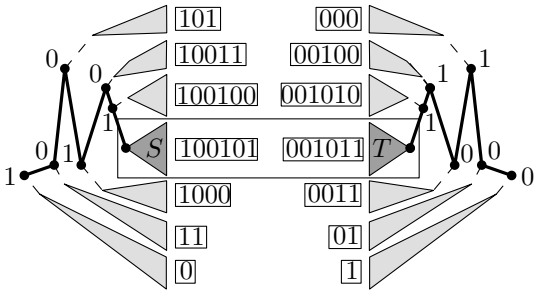


Fig. 3: Labels for a particular subinstance $\langle S, T\rangle$. The numbers at the nodes show the choices taken (swap/do not swap children) that led to S and T.

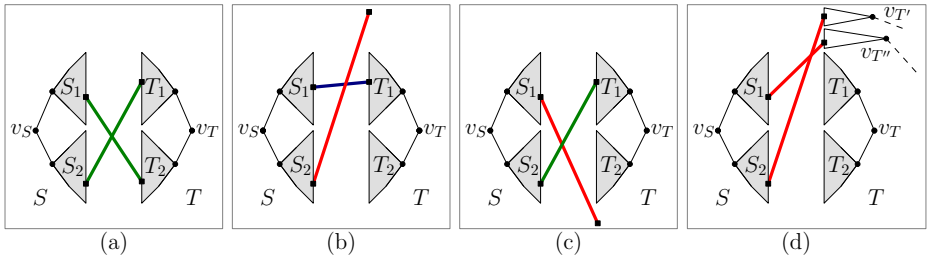


Fig. 4: Different types of current-level crossings. Type (d) is considered current-level only if the right leaves of the crossing edges have different labels, that is, if $\ell_{T'} \neq \ell_{T''}$.

Current-level crossings are crossings that can be avoided at this level by choosing one of the four configurations for the subtrees, independently of the choices to be done elsewhere in the recursion. Figure 4 illustrates the four different types. For type (d), we remark that crossings are considered to be *current-level* only if the largest S- and T-avoiding trees that contain the endpoints of the edges outside S and T are different. Crossings of type (d) where that is not the case cannot be counted at this point. We call them *indeterminate crossings*.

Lower-level crossings are crossings that appear based on choices taken by solving the subinstances of S and T recursively. We cannot do anything about them at this level, but we know their exact number after solving the subinstances.

Here is a sketch of the algorithm.

1. For all four choices of arranging $\{S_1, S_2\}$ and $\{T_1, T_2\}$, compute the total number of lower-level crossings recursively. Before each recursive call $\langle S_i, T_j\rangle$, we assign proper labels to some of the leaves of S and T, as follows. All leaves in S_i that connect to T_{3-j} (that is, T_1 if $j = 2$, T_2 otherwise) get the label ℓ_T with a 0 or 1 appended depending on whether T_j is above or below T_{3-j}. Then we do the analogue for all leaves of T_j connected to S_{3-i}.

2. For each choice $\langle S_i, T_j \rangle$ compute the number of current-level crossings (details below).
3. Return the choice that has the smallest sum of lower-level and current-level crossings.

The labels are needed to propagate as much information as possible to the smaller subinstances. For example, even though at this stage of the recursion it is clear that the leaves of, say T_{3-j}, are above the leaves of the subtrees below T, once we recurse into the top subinstance, this information will be lost, implying that what was a current-level crossing at this stage, will become an indeterminate crossing later. The labeling allows to prevent this loss of information.

The number of current-level crossings can be computed in linear time as follows. We go through all inter-tree edges incident to leaves of S and put each edge into one of at most $O(\log n)$ different classes, depending on the labels of the endpoints outside S. Then we repeat the same for T. This takes linear time. Depending on where the largest S- or T-avoiding trees go (above or below), all edge pairs belonging to a specific pair of labels do or do not intersect. Hence we can count the total number of current-level crossings by multiplying the cardinalities of the $O(\log^2 n)$ pairs of classes whose edges all intersect each other.

The running time of the algorithm satisfies the recurrence $T(n) \leq 8T(n/2) + O(n)$, which solves to $T(n) = O(n^3)$. We now prove that the algorithms yields a 2-approximation. In the full version [2] we show that our analysis is tight.

Theorem 3. *Given a complete binary tanglegram $\langle S_0, T_0 \rangle$ with n inter-tree edges, the recursive algorithm computes in $O(n^3)$ time a drawing of $\langle S_0, T_0 \rangle$ that has at most twice as many crossings as an optimal drawing.*

Proof. Fix an optimal drawing δ of $\langle S_0, T_0 \rangle$. The algorithm tries, for a given subinstance $\langle S, T \rangle$ of $\langle S_0, T_0 \rangle$, all four possible layouts of $S = (S_1, S_2)$ and $T = (T_1, T_2)$. Assume that in δ, $\langle S, T \rangle$ is drawn as $\langle (S_1, S_2), (T_1, T_2) \rangle$. We distinguish between four different areas for the endpoints of the edges: above $\langle S, T \rangle$, in $\langle S_1, T_1 \rangle$, in $\langle S_2, T_2 \rangle$, and below $\langle S, T \rangle$. We number these regions from 0 to 3 (see Fig. 5(a)). This allows us to classify the edges into 16 groups (two of which, 0–0 and 3–3, are not relevant). We denote the number of i–j edges, that is, edges from area i to area j, by n_{ij} (for $i, j \in \{0, 1, 2, 3\}$). Figures 5(b) and 5(c) show the 14 relevant groups of edges.

The only edge crossings that our recursive algorithm cannot take into account are the indeterminate crossings, which occur when the two edges connect to leaves above or below $\langle S, T \rangle$ that are in the same largest S- or T-avoiding tree. This is the case if both leaves have the same label. Such crossings cannot be predicted from the current subinstance because they depend on the relative position of the other two endpoints of the edges. We can, however, bound the number of these crossings.

We observe that any crossing of that type at the current subinstance was, in some previous step of the recursion, a crossing between two 1–2 edges or two 2–1 edges. We can upper-bound the number of these crossings by $\binom{n_{12}}{2} + \binom{n_{21}}{2}$.

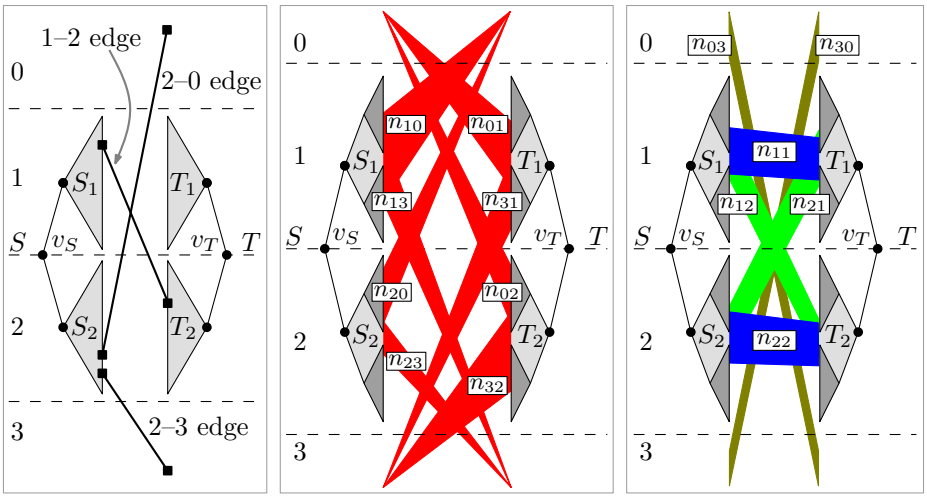


Fig. 5: For an instance $\langle (S_1, S_2), (T_1, T_2) \rangle$ the locations of the edge endpoints are divided into four areas (numbered 0–3); each edge is classified accordingly (a). This defines 14 groups of relevant edges, where n_{ij} denotes the number of i–j edges (b & c).

Let c_{alg} be the number of crossings in the solution produced by the algorithm, and let c_{opt} be the number of crossings of δ. Then

$$c_{\text{alg}} \leq c_{\text{opt}} + \binom{n_{12}}{2} + \binom{n_{21}}{2} \leq c_{\text{opt}} + (n_{12}^2 + n_{21}^2)/2. \tag{1}$$

Since our (sub)trees are complete, we have $n_{10} + n_{12} + n_{13} = n_{01} + n_{21} + n_{31}$ and $n_{01} + n_{02} + n_{03} = n_{10} + n_{20} + n_{30}$. These two equalities yield $n_{12} \leq n_{01} - n_{10} + n_{21} + n_{31}$ and $n_{01} - n_{10} \leq n_{20} + n_{30}$, respectively, and thus we obtain $n_{12} \leq n_{20} + n_{30} + n_{21} + n_{31}$ or, equivalently, $n_{12}^2 \leq n_{12} \cdot (n_{20} + n_{30} + n_{21} + n_{31})$.

It is easy to verify that all the terms on the right-hand side of the last inequality count crossings that cannot be avoided and must be present in the optimal solution as well. Hence $n_{12}^2 \leq c_{\text{opt}}$, and symmetrically $n_{21}^2 \leq c_{\text{opt}}$. Plugging this into (1) yields $c_{\text{alg}} \leq 2 \cdot c_{\text{opt}}$. □

General binary trees. Our recursive algorithm can also be applied to general, non-complete tanglegrams. Then, however, the approximation factor does not hold any more. Nöllenburg et al. [15] have evaluated several heuristics for TL; our recursive algorithm turned out to be a successful method for both complete and general binary tanglegrams.

Generalization to *d*-ary trees. The algorithm can also be generalized to complete d-ary trees. The recurrence relation of the running time changes to $T(n) \leq d \cdot (d!)^2 \cdot T(n/d) + O(n)$ since we need to consider all $d!$ subtree orderings

of both trees, each triggering d subinstances of size n/d. This resolves to $T(n) = O(n^{1+2\log_d(d!)})$. At the same time the approximation factor increases to $1 + \binom{d}{2}$.

Maximization version. Instead of the original TL problem, we now consider the dual problem TL* of maximizing the number of pairs of edges that do not cross. The tasks of finding optimal solutions for these problems are equivalent, but from the perspective of approximation it makes quite a difference which of the two problems we consider. Now we do not assume that we draw *binary* trees. Instead, if an internal node has more than two children, we assume that we may only choose between a given permutation of the children and the reverse permutation obtained by flipping the whole block of children.

In contrast to the TL problem, which is hard to approximate as we have shown in Theorem 1, the TL* problem has a constant-factor approximation algorithm. We show this (see full version [2]) by reducing TL* to a constrained version of the MaxCut problem, which can be approximately solved with the semidefinite programming rounding algorithm of Goemans and Williamson [8].

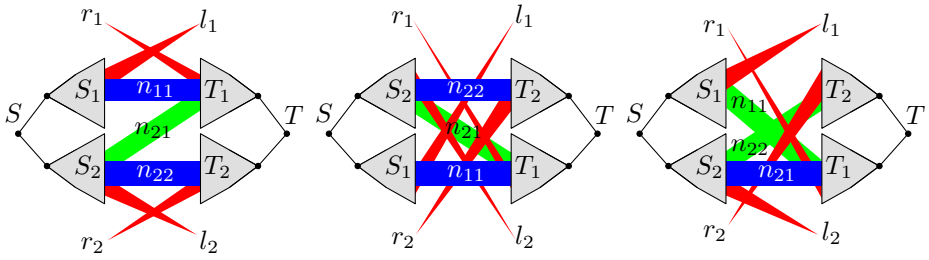
Theorem 4. *There exists a* 0.878*-approximation for the TL* problem.*

4 Fixed-Parameter Tractability

We consider the following parameterized problem. Given a complete binary TL instance $\langle S, T\rangle$ and a non-negative integer k, decide whether there exists a TL of S and T with at most k induced crossings. Our algorithm for this problem uses a labeling strategy, just as our algorithm in Sect. 3. However, here we do not select the subinstance that gives the minimum number of lower-level crossings, but we consider all subinstances and recurse on them. Thus, our algorithm traverses a search tree of branching factor 4. For the search tree to have bounded height, we need to ensure that whenever we go to a subinstance, the parameter value decreases at least by one. For efficient bookkeeping we consider current-level crossings only. At first sight this seems problematic: if a subinstance does not incur any current-level crossings, the parameter will not drop. The following key lemma—which does not hold for general binary trees—shows that there is a way out. It says that if there is a subinstance without current-level crossings, then we can ignore the other three subinstances and do not have to branch.

Lemma 1. *Let $\langle S, T\rangle$ be a complete binary TL instance, and let v_S be a node of S and v_T a node of T such that v_S and v_T have the same distance to their respective root. Further, let (S_1, S_2) be the subtrees incident to v_S and let (T_1, T_2) be the subtrees incident to v_T. If the subinstance $\langle (S_1, S_2), (T_1, T_2)\rangle$ does not incur any current-level crossings, then each of the subinstances $\langle (S_1, S_2), (T_2, T_1)\rangle$, $\langle (S_2, S_1), (T_1, T_2)\rangle$, and $\langle (S_2, S_1), (T_2, T_1)\rangle$ has at least as many crossings as $\langle (S_1, S_2), (T_1, T_2)\rangle$, for any fixed ordering of the leaves of S_1, S_2, T_1 and T_2.*

Proof. If the subinstance $\langle (S_1, S_2), (T_1, T_2)\rangle$ does not incur any current-level crossings, there are no edges between S_1 and T_2 or between S_2 and T_1. We



(a) $\langle (S_1, S_2), (T_1, T_2) \rangle$ (b) $\langle (S_2, S_1), (T_2, T_1) \rangle$ (c) $\langle (S_1, S_2), (T_2, T_1) \rangle$

Fig. 6: Edge types and crossings of the instance $\langle S, T \rangle$.

only consider the first case; the second is symmetric. We categorize the inter-tree edges originating from the four subtrees according to their destination—see Fig. 6(a)—and denote the numbers of edges of the various types by n_{11}, n_{21}, n_{22}, l_1, l_2, r_1, and r_2. Since we consider complete binary trees, we obtain $l_1 = r_1 + n_{21}$, $r_2 = l_2 + n_{21}$, and $r_1 + n_{11} = l_2 + n_{22}$.

We fix an ordering σ of the leaves of the subtrees S_1, S_2, T_1, T_2. We first compare the number of crossings in $\langle (S_1, S_2), (T_1, T_2) \rangle$ with the number of crossings in $\langle (S_2, S_1), (T_2, T_1) \rangle$, see Fig. 6(b). The subinstance $\langle (S_1, S_2), (T_1, T_2) \rangle$ can have at most $n_{21}(n_{11}+n_{22})$ crossings that do not occur in $\langle (S_2, S_1), (T_2, T_1) \rangle$. However, $\langle (S_2, S_1), (T_2, T_1) \rangle$ has at least $l_1(l_2+n_{21}+n_{22})+l_2 n_{11}+r_2(r_1+n_{21}+n_{11})+r_1 n_{22}$ crossings that do not appear in $\langle (S_1, S_2), (T_1, T_2) \rangle$. Plugging in the above equalities for l_1 and r_2, we get $(r_1+n_{21})(l_2+n_{21}+n_{22})+l_2 n_{11}+(l_2+n_{21})(r_1+n_{21}+n_{11})+r_1 n_{22} \geq n_{21}(n_{11}+n_{22})$. Thus, the subinstance $\langle (S_2, S_1), (T_2, T_1) \rangle$ has at least as many crossings with respect to σ as $\langle (S_1, S_2), (T_1, T_2) \rangle$ has.

Next, we compare the number of crossings in $\langle (S_1, S_2), (T_1, T_2) \rangle$ with the number of crossings in $\langle (S_1, S_2), (T_2, T_1) \rangle$, see Fig. 6(c). Now the number of additional crossings of $\langle (S_1, S_2), (T_1, T_2) \rangle$ is at most $n_{21} n_{22}$, and the subinstance $\langle (S_1, S_2), (T_2, T_1) \rangle$ has at least $(r_1+n_{11})(r_2+n_{22})+r_2 n_{21}$ crossings more. With the equality $r_1 + n_{11} = l_2 + n_{22}$ and the inequality $r_2 + n_{22} \geq n_{21}$ we get $(r_1+n_{11})(r_2+n_{22})+r_2 n_{21} \geq n_{22} n_{21}$. Thus, the subinstance $\langle (S_1, S_2), (T_2, T_1) \rangle$ has at least as many crossings with respect to σ as $\langle (S_1, S_2), (T_1, T_2) \rangle$ has.

By symmetry, the same holds for $\langle (S_2, S_1), (T_1, T_2) \rangle$. □

Thus, to decompose the instance into four subinstances we spend $O(n^2)$ time. Therefore we spend $O(4^k n^2)$ time to produce all leaves of our bounded-height search tree (omitting details). At each leaf of the search tree, we obtain a certain layout of $\langle S, T \rangle$, and the accumulated number of current-level crossings is at most k. This, however, does not mean that the total number of crossings is at most k since we did not keep track of the indeterminate crossings. Therefore, at each leaf we still need to check how many crossings the corresponding layout has. This can be done in $O(n \log n)$ time. If one of the leaves yields at most k crossings, the algorithm outputs "Yes" and the layout; otherwise it outputs "No".

Theorem 5. *The algorithm sketched above solves the parameterized version of complete binary TL in $O(4^k n^2)$ time.*

References

1. Bertolazzi, P., Di Battista, G., Mannino, C., Tamassia, R.: Optimal upward planarity testing of single-source digraphs. SIAM J. Comput. 27(1), 132–169 (1998)
2. Buchin, K., Buchin, M., Byrka, J., Nöllenburg, M., Okamoto, Y., Silveira, R.I., Wolff, A.: Drawing (complete) binary tanglegrams: Hardness, approximation, fixed-parameter tractability. Arxiv report (2008), `http://arxiv.org/abs/0806.0920`
3. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Zhang, L.: On distances between phylogenetic trees. In: Proc. 18th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA'97). pp. 427–436 (1997)
4. Dujmovič, V., Fernau, H., Kaufmann, M.: Fixed parameter algorithms for one-sided crossing minimization revisited. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 332–344. Springer, Heidelberg (2004)
5. Dwyer, T., Schreiber, F.: Optimal leaf ordering for two and a half dimensional phylogenetic tree visualization. In: Proc. Australasian Sympos. Inform. Visual. (InVis.au'04). CRPIT, vol. 35, pp. 109–115. Australian Comput. Soc. (2004)
6. Eades, P., Wormald, N.: Edge crossings in drawings of bipartite graphs. Algorithmica 10, 379–403 (1994)
7. Fernau, H., Kaufmann, M., Poths, M.: Comparing trees via crossing minimization. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 457–469. Springer, Heidelberg (2005)
8. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM 42(6), 1115–1145 (1995)
9. Hafner, M.S., Sudman, P.D., Villablanca, F.X., Spradling, T.A., Demastes, J.W., Nadler, S.A.: Disparate rates of molecular evolution in cospeciating hosts and parasites. Science 265, 1087–1090 (1994)
10. Holten, D., van Wijk, J.J.: Visual comparison of hierarchically organized data. In: Proc. 10th Eurographics/IEEE-VGTC Sympos. Visualization (EuroVis'08). pp. 759–766 (2008)
11. Khot, S.: On the power of unique 2-prover 1-round games. In: Proc. 34th Annu. ACM Sympos. Theory Comput. (STOC'02). pp. 767–775 (2002)
12. Khot, S., Vishnoi, N.K.: The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into l_1. In: Proc. 46th Annu. IEEE Sympos. Foundat. Comput. Sci. (FOCS'05). pp. 53–62 (2005)
13. Lozano, A., Pinter, R.Y., Rokhlenko, O., Valiente, G., Ziv-Ukelson, M.: Seeded tree alignment and planar tanglegram layout. In: Giancarlo, R., Hannenhalli, S. (eds.) WABI 2007. LNCS, vol. 4645, pp. 98–110. Springer, Heidelberg (2007)
14. Nagamochi, H.: An improved bound on the one-sided minimum crossing number in two-layered drawings. Discrete Comput. Geom. 33(4), 565–591 (2005)
15. Nöllenburg, M., Holten, D., Völker, M., Wolff, A.: Drawing binary tanglegrams: An experimental evaluation. Arxiv report (2008), `http://arxiv.org/abs/0806.0928`
16. Page, R.D.M. (ed.): Tangled Trees: Phylogeny, Cospeciation, and Coevolution. University of Chicago Press (2002)
17. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics 11(2), 109–125 (1981)