Drawing 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 with no edge crossing 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 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 prove that under the Unique Games Conjecture there is no constant-factor approximation for general binary trees. We show that the maximization version of the 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

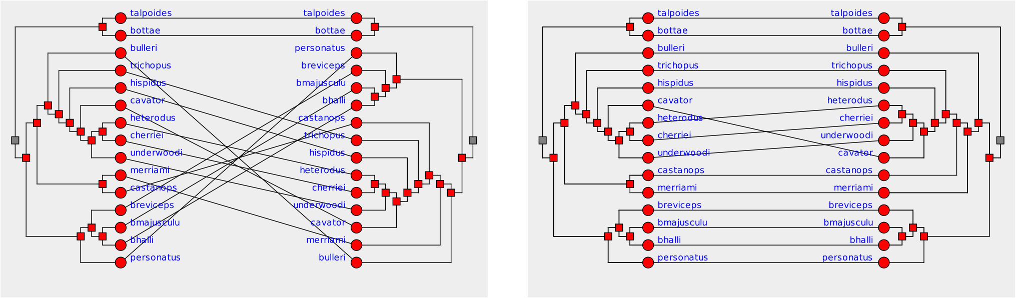
In this paper we are interested in drawing so-called *tanglegrams* [16], that is, pairs of trees whose leaf sets are in one-to-one correspondence. The need to visually compare

[⋆] Work 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].

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, with no edge crossing), (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., [16, 14]), 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. [18] and are widely used for drawing hierarchical graphs. In 2SCM, vertices of a bipartite graph are to be placed on two parallel lines (called *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. 1SCM is also NP-hard [6].

In contrast to TL, a vertex in an instance of 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 [15] 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 the TL problem 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 the TL problem is NP-hard and gave a fixed-parameter algorithm that runs in $O^\star(c^k)$ time, where c is a constant that they estimate 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. [14] gave a quadratic-time algorithm for the same special case, to which they refer as *planar tanglegram layout.* Holten and van Wijk [11] presented a visualization tool for two (partially) matched hierarchical data sets that uses a barycenter-like heuristic for crossing reduction prior to applying an edge bundling step that yields their final layout.

Our results. We first take a closer look at the complexity of the TL problem, see Section 2. By a new reduction from Max2Sat we show that the TL problem is NP-hard even when restricted to *complete* binary trees. We further show that without this restriction, the TL problem 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.

We then give a simple recursive heuristic for binary TL. It works very well in practice and is very fast when combined with branch-and-bound. For an experimental evaluation, see the companion paper [10]. Our main result in this paper is that our heuristic is in fact a 2-approximation for complete binary TL, see Section 3. On complete binary tanglegrams our algorithm runs in $O(n^3)$ time. When drawing pairs of complete d-ary trees our algorithm achieves a factor-$(1 + \binom{d}{2})$ approximation in $O(n^{1+2\log_d(d!)})$ time. For $d \geq 3$ the running time is upper-bounded by $O(n^{2d-1.7})$.

Finally, we give a new fixed-parameter algorithm for complete binary TL that is both much simpler and much 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 Section 4.

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, $\Lambda = \{1, \dots, n\}$. 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 is to find 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 additionally insist that the leaves in $L(S)$ are placed on the vertical 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. We use the notation $\langle S, T \rangle$ when referring to such an instance of the TL problem.

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 $\{1, \dots, n\}$, 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 the TL problem for complete and for general binary trees. Fernau et al. [7] have shown that the TL problem is NP-complete for general binary trees. Their proof, however, uses extremely unbalanced trees and does not extend to complete binary trees. We show that the TL problem remains hard even when restricted to complete binary trees. We reduce from Max2Sat with at most 3 occurrences of each variable. Our construction (see the appendix) is completely different from that of Fernau et al., who reduce from MaxCut. We construct a TL instance in which one pair of aligned subtrees contains the variable gadgets. The two pairs of aligned subtrees to both sides of the variable gadgets contain the clause gadgets. The fourth pair of aligned subtrees on the same level has no crossings. Each clause gadget is modeled by a pair of smaller subtrees, see Fig. 12. These are connected by inter-tree edges to the gadgets of the two corresponding variables. These edges cause exactly one additional crossing for each unsatisfied clause in an optimal solution. Thus we can infer the maximum number of satisfied clauses from an optimal TL solution.

Theorem 1. *The TL problem is NP-complete even for complete binary trees.*

Next we consider the complexity of the TL problem for two (not necessarily complete) binary trees. We show that this problem is essentially as hard as the MinUncut problem. As a result, we relate the existence of a constant-factor approximation for TL to the Unique Games Conjecture (UGC) by Khot [12]. 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 [13], 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 2. *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. Note that 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 will 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 the crossings: 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. Then it can produce a drawing $D(S, T)$, for the above defined instance $\langle S, T\rangle$, 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.

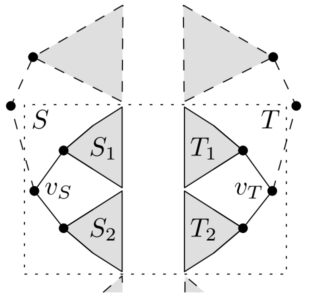


Fig. 2: Context of subproblem $\langle S, T\rangle = \langle (S_1, S_2), (T_1, T_2)\rangle$.

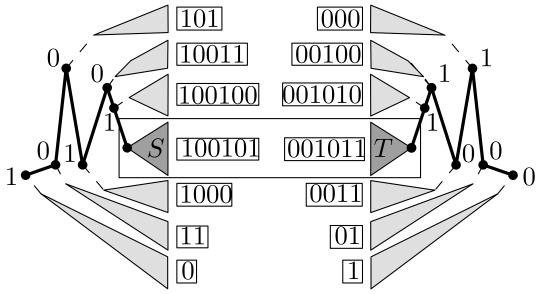


Fig. 3: Labels for a particular subproblem $\langle S, T\rangle$. The numbers at the nodes show the choice taken (swap/do not swap children) at that step of the recursion that led to S and T.

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

3 Approximation

We now present our main result, a 2-approximation algorithm for TL that runs in $O(n^3)$ time. The idea is to split the problem recursively at the root of the trees into two subproblems, each consisting of a pair of complete binary trees.

Let $\langle S_0, T_0\rangle$ be the TL instance we want to solve. At a given level $\langle S, T\rangle$ in the recursion, we have two trees S and T, typically part of larger trees (that is, $S \subseteq S_0$ and $T \subseteq T_0$). Let the roots of S and T be v_S and v_T, respectively. Besides the two trees, we will use some additional information.

Firstly, associated with v_S and v_T we will have labels ℓ_S and ℓ_T that indicate what choices in the recursion so far led to the current subproblems. A label is a binary string, where '0' or '1' represents each of the two choices at each node in the path from the root of the original tree, to the current root. The length of the labels (denoted $|\ell_S|$ and $|\ell_T|$) gives the depth of the recursion (see Fig. 3).

We also assign labels to some other subtrees of $\langle S_0, T_0\rangle$ besides S and T. Given a leaf $v \in T_0 \setminus T$, we define the *nc-subtree* of v, with respect to T, as the largest complete binary subtree of T_0 that does not contain T and contains v (defined analogously for leaves in S_0). Each different nc-subtree receives a label, in the same way as S and T. For a given $\langle S, T\rangle$, there are $2(|\ell_S| + 1) = 2(|\ell_T| + 1)$ different labels. Note that the labels of the nc-subtrees are relative to the labels of v_S and v_T (different S or T will lead to different labels). We will sometimes refer to the label of leaf v, meaning the label of the nc-subtree of v.

Secondly, since S and T are part of a larger tree, some of the leaves of S may not have the matching leaf in T (and vice versa). This means that at some previous step of the algorithm, it was decided that such leaves will be matched to leaves in some other subtrees, above or below $\langle S, T\rangle$. We will not know exactly to which leaves they

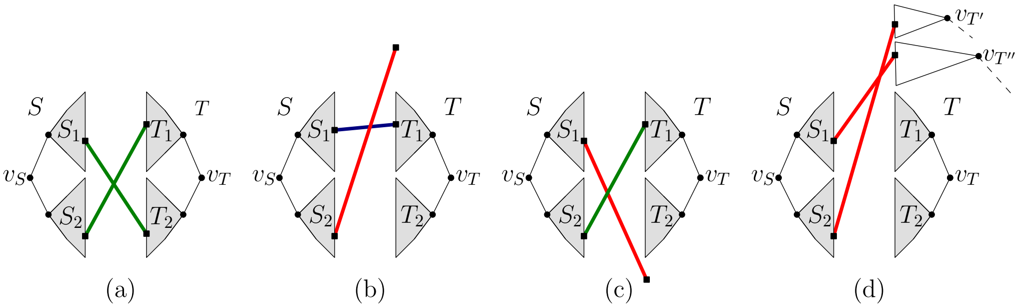


Fig. 4: Different types of current-level crossings. For the fourth type, (d), the crossing is considered current-level only if the right leaves of the edges that cross have different labels, that is, $\ell_{T'} \neq \ell_{T''}$.

are matched, but we will 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. 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. We will try each of them, invoking the algorithm recursively for the top half and for the bottom half. Then we will return the configuration with the lowest number of crossings.

When counting the crossings that each option creates, we will distinguish two types: *current-level* and *lower-level* crossings.

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 different types of current-level crossings. For the fourth type, (d), shown in Fig. 4, we remark that the crossings are considered to be *current-level* only if the nc-subtrees that contain the endpoints of the edges outside S and T are different. Crossings that have the shape of type (d) but with both endpoints going to the same nc-subtree cannot be counted at this point, and will be called *indeterminate crossings*.

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

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.

It is important to notice that the labels are needed to propagate as much information as possible to the smaller subproblems. 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 subproblem, 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.

It remains to describe how to compute the number of current-level crossings efficiently. This can be done as follows. We go through all inter-tree edges incident to leaves of each of the four subtrees and put each edge into one of at most $O(\log n)$ different classes depending on the labels of the other endpoints of the edges. Depending on where (that is, above or below) the nc-subtrees go, 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 in linear time.

The running time of the algorithm satisfies the recurrence relation $T(n) \leq 8T(n/2) + O(n)$, which resolves to $T(n) = O(n^3)$ by the master method [2].

Theorem 3. *The recursive algorithm computes a solution to the complete binary TL problem in $O(n^3)$ time. The resulting drawing has at most twice as many crossings as an optimal drawing.*

Proof. The algorithm will try, for a given subproblem $\langle S, T\rangle$, all four possible layouts of $S = (S_1, S_2)$ and $T = (T_1, T_2)$. Hence we can assume we know the order of the children of v_S and v_T in an optimal solution. Assume, w.l.o.g., that it is $\langle (S_1, S_2), (T_1, T_2)\rangle$. 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). This allows us to classify the edges into 16 groups (two of which, 0–0 and 3–3, are not relevant). We will denote the number of edges from area i to area j by n_{ij} (for $i, j \in \{0, 1, 2, 3\}$). Figure 6 shows the 14 different 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/below $\langle S, T\rangle$, that are in the same nc-subtree (thus both leaves have the same label). The occurrence of such a crossing cannot be determined from the current subproblem because it depends on the relative location of the other two endpoints of the edges. However, we can bound the number of these crossings.

We observe that any crossing of that type at the current subproblem 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 $\binom{n_{12}}{2} + \binom{n_{21}}{2}$. Let ALG be the number of crossings in the solution produced by the algorithm, and OPT the one in an optimal solution. We have

$$ALG \leq OPT + \binom{n_{12}}{2} + \binom{n_{21}}{2} \leq OPT + (n_{12}^2 + n_{21}^2)/2 \qquad (1)$$

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 $n_{12} \leq n_{01} - n_{10} + n_{21} + n_{31}$

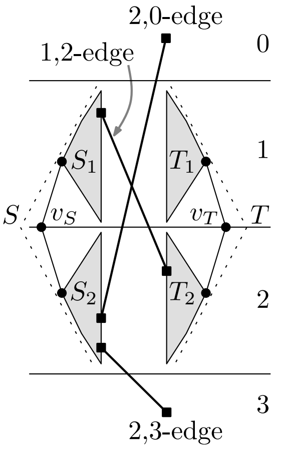


Fig. 5: The possible locations for the endpoints of the edges are divided into four areas (numbered from 0 to 3). Each edge can be classified according to the areas of its endpoints.

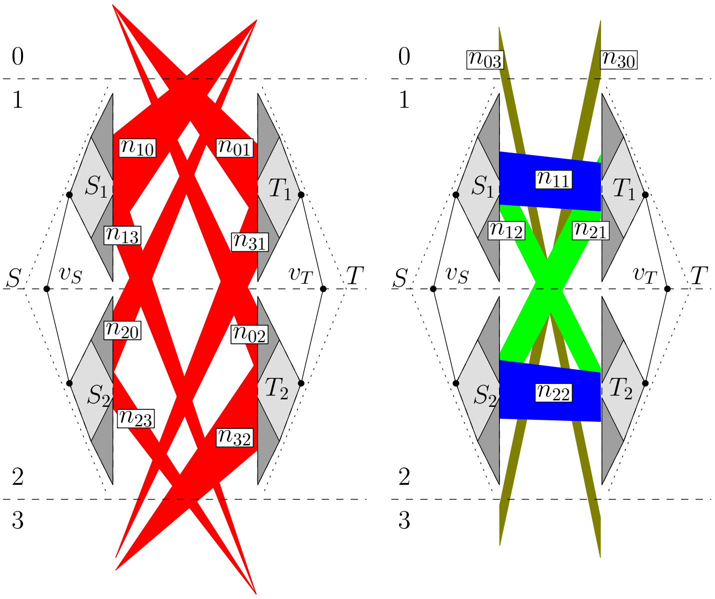


Fig. 6: The 14 different (relevant) groups of edges in $\langle(S_1, S_2), (T_1, T_2)\rangle$.

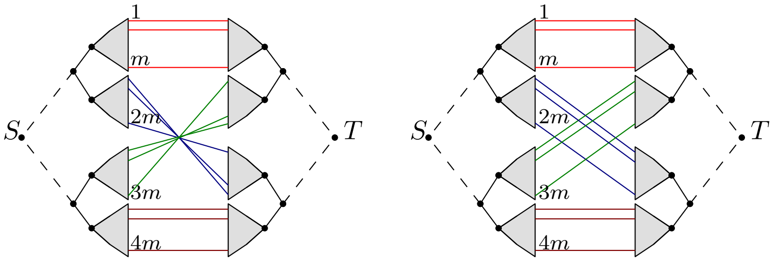


Fig. 7: Example of trees for which the approximation algorithm can output a solution (left) that has roughly twice as many crossings as the optimal one (right).

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 $n_{12}^2 \leq OPT$, and symmetrically $n_{21}^2 \leq OPT$. Plugging this into (1), we get $ALG \leq 2 \cdot OPT$. □

The approximation factor of 2 is tight: let $n = 4m$ and let S have leaves ordered $1, \ldots, 4m$ and let T have leaves ordered $1, \ldots, m, 3m, \ldots, 2m+1, m+1, \ldots, 2m, 3m+1, \ldots, 4m$. Then our algorithm can construct a drawing with $m^2 + 2\binom{m}{2} = m(2m-1)$ crossings, while the optimal drawing has only m^2 crossings (see Fig. 7).

General binary trees. Obviously, our recursive algorithm can also be applied to general, non-complete tanglegrams. In this case, however, the approximation factor does not hold any more, which is also indicated by Theorem 2. The companion paper [10] contains an extensive experimental evaluation of several heuristic algorithms for TLs in which our recursive algorithm turned out to be a successful method for both complete and general binary tanglegrams.

Generalization to d-ary trees. The recursive algorithm can be generalized to complete d-ary trees. The recurrence relation of the algorithm's 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 of which triggers d subinstances of size n/d. Again, by the master method, this resolves to $T(n) = O(n^{1+2\log_d(d!)})$. At the same time the approximation factor increases to $1 + \binom{d}{2}$.

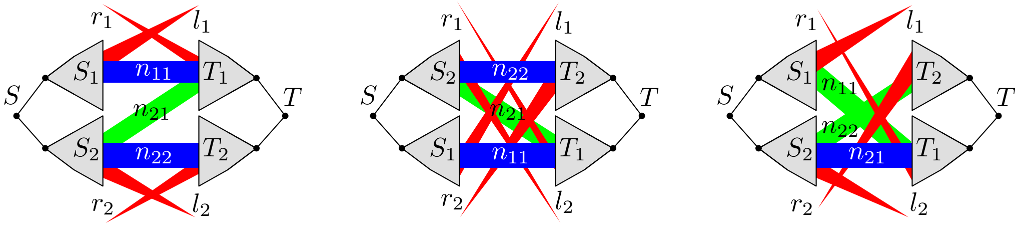
Maximization version. Instead of the original TL problem, which minimizes the number of pairs of edges that cross each other, we may consider the dual problem TL* of maximizing the number of pairs of edges that do not cross. The tasks of finding optimal solutions for these problems are equivalent, but 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 2, the TL* problem has a constant-factor approximation algorithm. We show this (see the appendix) by reducing TL* to a constrained version of the MaxCut problem, which can be approximately solved with a semidefinite programming rounding algorithm by Goemans and Williamson [8].

Theorem 4. *There exists a* 0.878*-approximation algorithm 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 approximation algorithm in Section 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 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. This could be seen as a preprocess at each



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

branching occasion, and is also exploited in some existing fixed-parameter algorithms. Note that the lemma does not hold for general binary trees.

Lemma 1. *Given a pair $\langle S, T \rangle$ of two complete binary trees as an instance of the TL problem and two nodes v_S, v_T of S, T, respectively, with the same distance to their respective root. Let (S_1, S_2) be the subtrees incident to v_S and (T_1, T_2) 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 any ordering of the leaves of this subinstance does not have more crossings than the same ordering of the leaves of one of the other subinstances $\langle (S_1, S_2), (T_2, T_1) \rangle$, $\langle (S_2, S_1), (T_1, T_2) \rangle$, or $\langle (S_2, S_1), (T_2, T_1) \rangle$.*

Proof. If the subinstance $\langle (S_1, S_2), (T_1, T_2) \rangle$ does not incur any current-level crossings, the edges originating from these four subtrees are edges of the types shown in Fig. 8a (or the symmetric case with no edges between S_2 and T_1). Let $n_{11}, n_{21}, n_{22}, l_1, l_2, r_1, r_2$ be the numbers of edges as in Fig. 8. Since we consider complete binary trees we obtain the following equalities: $l_1 = r_1 + n_{21}$, $r_2 = l_2 + n_{21}$, and $r_1 + n_{11} = l_2 + n_{22}$.

Take any fixed ordering of the leaves of the subtrees S_1, S_2, T_1, T_2. We first compare the number of crossings of the subinstance $\langle (S_1, S_2), (T_1, T_2) \rangle$ with the number of crossings of the subinstance $\langle (S_2, S_1), (T_2, T_1) \rangle$ in Fig. 8b. 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 $\langle (S_1, S_2), (T_1, T_2) \rangle$. Inserting 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 same ordering of leaves does not give more crossings for $\langle (S_1, S_2), (T_1, T_2) \rangle$ than it does for $\langle (S_2, S_1), (T_2, T_1) \rangle$.

Next, we compare the number of crossings of the subinstance $\langle (S_1, S_2), (T_1, T_2) \rangle$ with the number of crossings of the subinstance $\langle (S_1, S_2), (T_2, T_1) \rangle$ in Fig. 8c. 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, again $\langle (S_1, S_2), (T_1, T_2) \rangle$ does not have more crossings than $\langle (S_1, S_2), (T_2, T_1) \rangle$ for the same leaf ordering. By symmetric reasoning the same holds for $\langle (S_2, S_1), (T_1, T_2) \rangle$. □

Thus, to decompose the instance to 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." We summarize:

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

5 Conclusions and Open Problems

[something here?]

Acknowledgments. We thank Danny Holten and Jack van Wijk for introducing us to this exciting problem and David Bryant for pointing us to the work of Roderic Page on host and parasite trees.

References

[1] P. Bertolazzi, G. Di Battista, C. Mannino, and R. Tamassia. Optimal upward planarity testing of single-source digraphs. *SIAM J. Comput.*, 27(1):132–169, 1998.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2nd edition, 2001.

[3] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang. On distances between phylogenetic trees. In *Proc. 18th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA'97)*, pages 427–436, 1997.

[4] V. Dujmović, H. Fernau, and M. Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. In G. Liotta, editor, *Proc. 11th Internat. Sympos. Graph Drawing (GD'03)*, volume 2912 of *Lecture Notes Comput. Sci.*, pages 332–344. Springer-Verlag, 2004.

[5] T. Dwyer and F. Schreiber. Optimal leaf ordering for two and a half dimensional phylogenetic tree visualization. In N. Churcher and C. Churcher, editors, *Proc. Australasian Sympos. Inform. Visual. (InVis.au'04)*, volume 35 of *CRPIT*, pages 109–115. Australian Computer Society, 2004.

[6] P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 10:379–403, 1994.

[7] H. Fernau, M. Kaufmann, and M. Poths. Comparing trees via crossing minimization. In R. Ramanujam and S. Sen, editors, *Proc. 25th Intern. Conf. Found. Softw. Techn. Theoret. Comput. Sci. (FSTTCS'05)*, volume 3821 of *Lecture Notes Comput. Sci.*, pages 457–469, 2005.

[8] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

[9] M. S. Hafner, P. D. Sudman, F. X. Villablanca, T. A. Spradling, J. W. Demastes, and S. A. Nadler. Disparate rates of molecular evolution in cospeciating hosts and parasites. *Science*, 265:1087–1090, 1994.

[10] D. Holten, M. Nöllenburg, M. Völker, and A. Wolff. Drawing binary tanglegrams: an experimental evaluation. Submitted to GD'08, May 2008. Available at http://arxiv.org/abs/...

[11] D. Holten and J. J. van Wijk. Visual comparison of hierarchically organized data. In *Proc. 10th Eurographics/IEEE-VGTC Sympos. Visualization (EuroVis'08)*, 2008. To appear.

[12] S. Khot. On the power of unique 2-prover 1-round games. In *Proc. 34th Annu. ACM Sympos. Theory Comput. (STOC'02)*, pages 767–775, 2002.

[13] S. Khot and N. K. Vishnoi. 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)*, pages 53–62, 2005.

[14] A. Lozano, R. Y. Pinter, O. Rokhlenko, G. Valiente, and M. Ziv-Ukelson. Seeded tree alignment and planar tanglegram layout. In R. Giancarlo and S. Hannenhalli, editors, *Proc. 7th Internat. Workshop Algorithms Bioinformatics (WABI'07)*, volume 4645 of *Lecture Notes Comput. Sci.*, pages 98–110. Springer-Verlag, 2007.

[15] H. Nagamochi. An improved bound on the one-sided minimum crossing number in two-layered drawings. *Discrete Comput. Geom.*, 33(4):565–591, 2005.

[16] R. D. M. Page, editor. *Tangled Trees: Phylogeny, Cospeciation, and Coevolution.* University of Chicago Press, 2002.

[17] V. Raman, B. Ravikumar, and S. S. Rao. A simplified NP-complete MAXSAT problem. *Inform. Process. Lett.*, 65:1–6, 1998.

[18] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.

Appendix

Theorem 1. *The TL problem is NP-complete even for complete binary trees.*

Proof. Recall the MAX2SAT problem which is defined as follows. Given a set $U = \{x_1, \ldots, x_n\}$ of Boolean variables, a set $C = \{c_1, \ldots, c_m\}$ of disjunctive clauses containing two literals each, and an integer K, the question is whether there is a truth assignment of the variables such that at least K clauses are satisfied. We consider a restricted version of MAX2SAT, where each variable appears in at most three clauses. This version remains NP-complete [17].

Our reduction constructs two complete binary trees S and T, in which certain aligned subtrees serve as variable gadgets and others as clause gadgets. We further determine an integer K' such that the instance $\langle S, T \rangle$ has less than K' crossings if and only if the corresponding MAX2SAT instance has a truth assignment that satisfies at least K clauses.

The high-level structure of the two trees is depicted in Fig. 9. From top to bottom, the four subtrees at level 2 on both sides are a clause subtree, a variable subtree, another clause subtree, and finally a dummy subtree. The subtrees are connected to each other by edges such that in any optimal solution they must be aligned in the depicted (or mirrored) order. Each clause gadget appears twice, once in each clause subtree, and is connected to the variable gadgets belonging to its two literals. Pairs of corresponding gadgets in S and T are connected to each other. Finally, non-crossing dummy edges connect unused leaves to complete S and T. In the following we describe the gadgets in more detail.

Variable gadgets. The basic structure of a variable gadget consists of two complete binary trees with 32 leaves each as shown in Fig. 10. Each tree has three highlighted subtrees of size 2 labeled a, b, c and a', b', c', respectively. From each of these subtrees there is one red *connector* edge leaving the gadget at the top and one leaving it at the bottom. As long as two connector edges from the same tree do not cross each other, they transfer the vertical order of the labeled subtrees towards a clause gadget. We define the configuration in Fig. 10a as *true* and the configuration in Fig. 10b as *false*. If the configuration is in its *true* state, the induced vertical order of the connector edges is $a < b < c$, otherwise the order is inverse: $c < b < a$. It can easily be verified that both states have the same number of crossings. To see that it is optimal observe that each pair of connector edges from the same subtree (for example, subtree a) always crosses all 26 gray edges in the gadget. Furthermore all 24 crossings of two connector edges in the figure are mandatory. Finally, the four crossings among the gray edges between subtrees 1 and $2'$ and subtrees 2 and $1'$ are also optimal. (Otherwise, if subtree 1 is opposite of subtree $2'$, there are at least 120 gray–gray crossings in addition to the 24 red–red crossings and the 156 red–gray crossings as opposed to a total of 184 crossings in either configuration of Fig. 10.)

Note that so far the gadget in the figure is designed for a single appearance of the variable since the four connector-edge triplets are required for a single clause. However, for the MAX2SAT reduction each variable can appear up to three times in different clauses. By appending a complete binary tree with four leaves as in Fig. 11 to each leaf of the gadget in Fig. 10 and copying each edge accordingly the above arguments still hold for the enlarged trees with 128 leaves each. Unused connector edges in opposite subtrees are linked to each other (a to a' etc.) as in Fig. 10b such that the number of crossings in the gadget remains balanced for both states.

Clause gadgets. For each clause $c_i = l_{i1} \vee l_{i2}$, where l_{i1} and l_{i2} denote the two literals, we create two clause gadgets: one in the upper clause subtrees and one in the lower clause subtrees (recall Fig. 9). Each gadget itself consists of two parts: one part that uses the connectors from the first variable in the left tree and those from the second variable in the right tree and vice versa. Fig. 12 shows one such part of the gadget in the lower clause subtrees, where the connector edges lead upwards. The gadget in the upper clause subtree is simply a mirrored version.

The basic structure consists of two aligned subtrees with eight leaves as depicted in Fig. 12. Three of the leaves on each side serve as the missing endpoints for the triplets of connector edges from the corresponding variables. Recall that for a positive literal with value *true* the order of the connector edges is $a < b < c$, and for a positive literal with value *false* it is $c < b < a$. (For negative literals the meaning of the orders is inverted.) The two connector leaves for the edges labeled a and b are in the same subtree with four leaves, the connector leaf for c is in the other subtree. Three cases need to be distinguished. If (1) both literals are *true*, then the configuration in Fig. 12a is optimal with 21 crossings. If (2) only one literal is *true*, then Fig. 12b shows an optimal configuration with 21 crossings again. Here the tree on the right side is rotated in its root

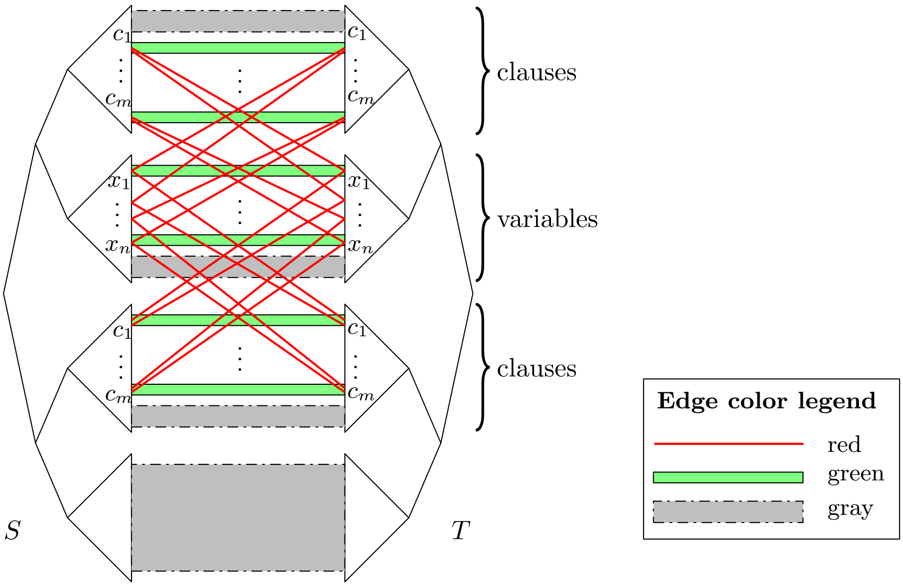
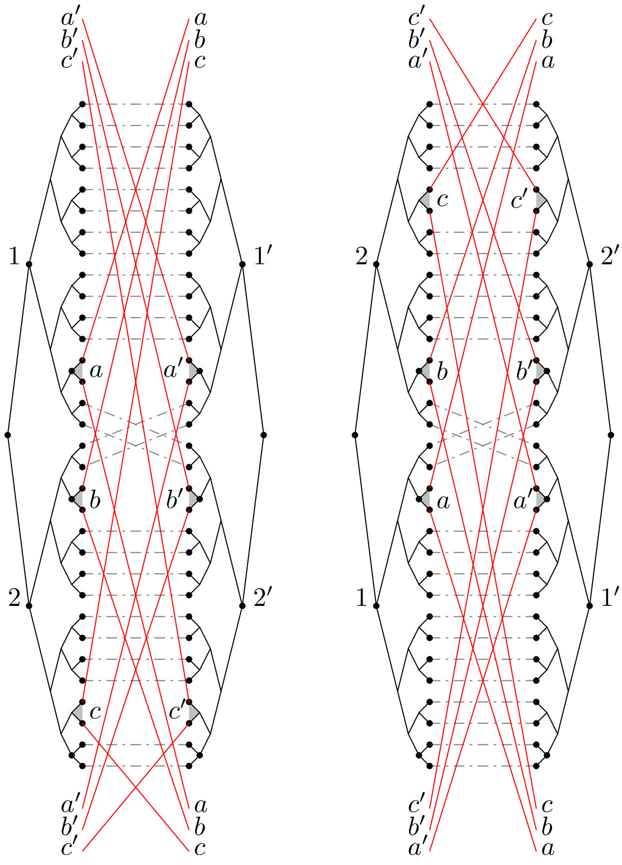


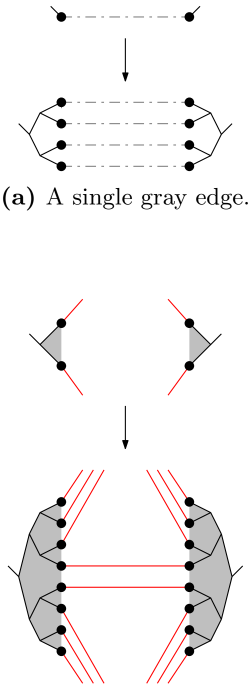
Fig. 9: High-level structure of the two trees S and T. Red edges connect clause and variable gadgets, green edges connect corresponding gadget halves, and gray edges are dummy edges to complete the trees.



(a) $x = \mathit{true}$

(b) $x = \mathit{false}$

Fig. 10: The variable gadget in its two optimal configurations with 184 crossings. Red edges are drawn solid, whereas dash-dot style is used for gray edges.



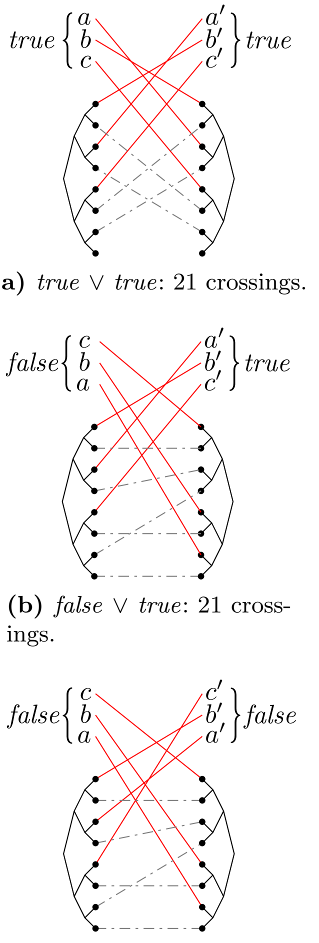
(a) A single gray edge.

(b) Two pairs of connector edges for a variable used in three clauses.

Fig. 11: Replacing each edge by four edges.

node. Finally, if (3) both literals are *false*, there are at least 22 crossings in the gadget as shown in Fig. 12c. Since this substructure is repeated four times for each clause we have 84 induced crossings for satisfied clauses and 88 induced crossings for unsatisfied clauses.

We construct the gadgets for all variables and clauses and link them together as two trees S and T, which are filled up such that they become complete binary trees. The general layout is as depicted in Fig. 9, where each dummy leaf in S is connected to the opposite dummy leaf in T such that there are no crossings among dummy edges. In each



(a) *true* ∨ *true*: 21 crossings.

(b) *false* ∨ *true*: 21 crossings.

(c) *false* ∨ *false*: 22 crossings.

Fig. 12: The clause gadget for a clause $c_i = l_{i1} \vee l_{i2}$.

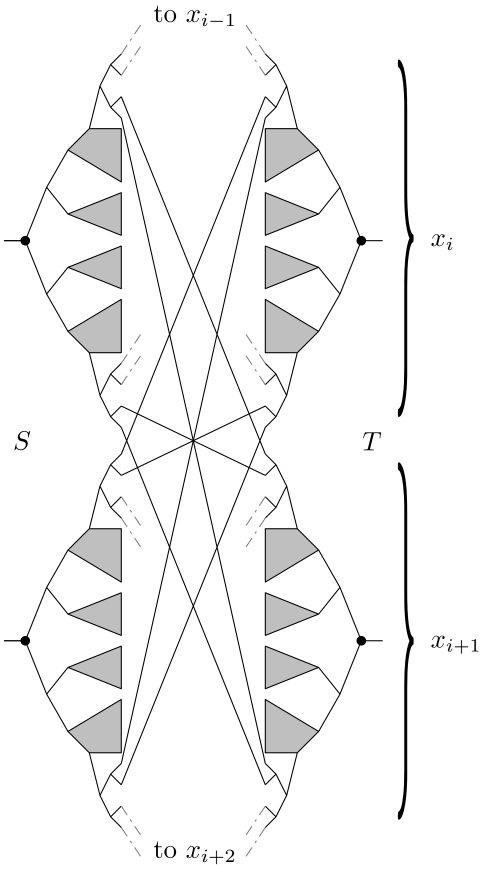


Fig. 13: Linking adjacent variable gadgets for x_i and x_{i+1}.

of the four main subtrees all dummy edges are consecutive. Thus of all dummy edges only those in the variable subtree have crossings with exactly half the connector edges.

It remains to compute the minimum number M of crossings that are always necessary, even if all clauses are satisfied. Then the Max2Sat instance has a solution with at least K satisfied clauses if and only if the constructed TL instance has a solution with at most $K' = M + 4(|C| - K)$ crossings. We get the corresponding variable assignment directly from the layout of the variable gadgets.

The first step for computing M is to fix an order for the variable gadgets in the variable subtree. Let this order be $x_1 < x_2 < \ldots < x_n$. To enforce this as the vertical order of the variable gadgets we need to establish links between adjacent gadgets such that any other order would increase the number of crossings. For these neighbor links we need eight of the 128 leaves in each half of each variable gadget as shown in Fig. 13. Since both subtrees below the root of x_i in S and both subtrees below the root of x_{i+1} in T are connected to each other, the minimum number of crossings of those edges is independent of the truth state of each gadget. However, separating two adjacent variables by tree rotations at higher levels in S and T leads to a large number of extra crossings since the eight neighbor links would cross all variable gadgets between x_i and x_{i+1}.

With the order of the variables fixed we sort all clauses lexicographically and place smaller clauses towards the top of the clause subtrees. Consider two clause gadgets in the same clause subtree. Then in the given clause order there are crossings between their connector-edge triplets if and only if the intervals between their respective variables intersect in the variable order. Since these crossings are unavoidable, the number of connector-triplet crossings in the lexicographic order of the clauses is optimal. Now we can finally compute all necessary crossings between connector edges, dummy edges and intra-gadget edges which yields the number M.

Since each gadget is of constant size the two trees and the number M can be computed in polynomial time.

The fact that the complete binary TL problem belongs to the class $\mathcal{NP}$ follows immediately from the NP-completeness of the general TL problem [7]. □

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

Proof. Fix any drawing of the two trees S and T in an instance of the TL* problem. Any internal node of each of the trees corresponds to a decision variable. The decision to make in each such node is whether to flip the subtree rooted in that node or not. We model this situation by a graph; a flip decision corresponds to deciding to which side of a cut the corresponding vertex is assigned.

For each internal node v of a tree in the instance of TL* the constructed graph G contains two vertices v and v'. For each pair of edges connecting leaves of the two trees, there is one edge in G. Let l_1 and l_2 (r_1 and r_2) denote the leaves of S (T) incident to this pair of edges. Let l be the lowest common ancestor of l_1 and l_2 in S ($l = \mathrm{LCA}(l_1, l_2)$) and let $r = \mathrm{LCA}(r_1, r_2)$ in T. If the considered pair of edges crosses in the initial drawing, then we have an edge $\{l, r\}$ in G. If the pair of edges does not cross in the initial drawing, then there is an edge $\{l, r'\}$ in G.

It remains to observe that cuts in G that separate each pair v, v' correspond to drawings of S and T in the instance of the TL* problem. Moreover, edges that are cut in G correspond to the pairs of edges that do not cross in the drawing of the two trees.

The resulting optimization problem is the MaxResCut problem (that is, the MaxCut problem with additional constraints forcing certain pairs of vertices to be separated by the cut) studied by Goemans and Williamson [8]. Therefore, we may use their semidefinite programming rounding algorithm to compute a 0.878-approximation of the largest constrained cut in the graph G. This cut determines which of the subtrees in the initial drawing must be flipped to obtain a drawing that is a 0.878-approximation to TL*. □