A faster fixed-parameter approach to drawing binary tanglegrams

Sebastian Böcker[1], Falk Hüffner[2], Anke Truss[1], and Magnus Wahlström[3]

[1] Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, 07743 Jena, Germany, {sebastian.boecker,anke.truss}@uni-jena.de
[2] Algorithms in Computational Genomics group, School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, hueffner@tau.ac.il
[3] Max-Planck-Institut für Informatik, Department 1: Algorithms and Complexity, Building 46.1, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, wahl@mpi-inf.mpg.de

Abstract. Given two binary phylogenetic trees covering the same n species, it is useful to compare them by drawing them with leaves arranged side-by-side. To facilitate comparison, we would like to arrange the trees to minimize the number of crossings k induced by connecting pairs of identical species. This is the NP-hard Tanglegram Layout problem. By providing a fast transformation to the Balanced Subgraph problem, we show that the problem admits an $O(2^k n^4)$ algorithm, improving upon a previous fixed-parameter approach with running time $O(c^k n^{O(1)})$ where $c \approx 1000$. We enhance a Balanced Subgraph implementation based on data reduction and iterative compression with improvements tailored towards these instances, and run experiments with real-world data to show the practical applicability of this approach. All practically relevant ($k \leq 1000$) Tanglegram Layout instances can be solved exactly within seconds. Additionally, we provide a kernel-like bound by showing how to reduce the Balanced Subgraph instances for Tanglegram Layout on complete binary trees to a size of $O(k \log k)$.

1 Introduction

In phylogenetics, researchers often wish to compare different phylogenetic trees with the same set of leaves: This can be two trees that resulted from applying different tree-building methods to the same dataset, a gene tree vs. species tree comparison, or a host-parasite comparison. The Tanglegram Layout problem (TL) deals with visually comparing a pair of binary rooted trees with identical leaf sets [5,10]: The trees are drawn such that the leaves of both trees face each other, and each leaf is connected to the corresponding leaf in the opposing tree by an edge, see Fig. 1. A layout with many crossings of connecting edges can be hard or even impossible to analyze. Hence, our goal is to find a layout of the two trees such that we can draw connecting edges with as few crossings as possible.

The Tanglegram Layout problem is NP-hard, even if both trees are complete [3]. In the same publication, Buchin et al. provide a $O(n^3)$ 2-approximation

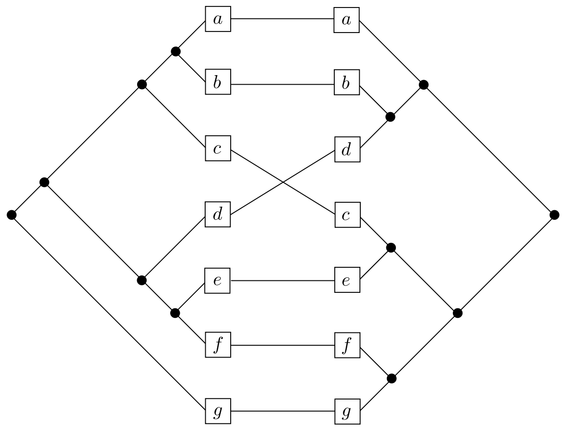

Fig. 1. A *tanglegram.*

and a $O(4^k n^3)$ fixed-parameter algorithm for this special case, where k is the number of crossings in an optimal tanglegram. They also show that under the Unique Games Conjecture, there is no constant-factor approximation for the problem on general binary trees, that is, trees that are not necessarily complete. For general binary trees, the fastest fixed-parameter algorithm is due to Fernau et al. [5] and has running time $O(c^k n^{O(1)})$, where c was estimated by the authors to be about 1024.

On the application side, Nöllenburg et al. [9] compare different heuristics and exact algorithms for Tanglegram Layout. Baumann et al. [2] and Bansal et al. [1] study the generalization where the perfect matching between leaves is replaced by an arbitrary bipartite graph, and present heuristics and Integer Linear Programs for its solution.

In this paper, we transform Tanglegram Layout instances to the Balanced Subgraph problem. The fastest fixed-parameter algorithm for the latter problem is due to Hüffner et al. [8] and has running time $O(2^k\, m^2)$ time, where k is the number of edge labels violated in an optimal solution and m is the number of graph edges. As an algorithm engineering technique, the authors also provide a set of efficient data reduction rules for Balanced Subgraph.

Our contributions. We show in Sec. 3 that we can transform a Tanglegram Layout instance into a Balanced Subgraph instance in polynomial time, so that Tanglegram Layout is solvable in $O(2^k n^4)$ time, where k is the minimum number of crossings in a tanglegram of the two input trees and n the number of leaves in each input tree. In Sec. 4, we present a $O(k \log k)$ kernel-like bound on the size of Balanced Subgraph instances derived from tanglegrams with complete binary trees. In experiments described in Sec. 5, we give some improvements to the Balanced Subgraph solver tailored towards our application. We then apply the algorithm to synthetic and real-world tanglegram datasets and thus show that we can compute exact solutions for all practically relevant tanglegram instances within seconds.

2 Preliminaries

For an inner node v of a binary tree, let $\mathcal{T}(v)$ be the subtree rooted at v, and $l(v)$ and $r(v)$ the left and right child of v, respectively. Then $L(v) := \mathcal{T}(l(v))$ and $R(v) := \mathcal{T}(r(v))$ are the subtrees rooted at $l(v)$ and $r(v)$, respectively. Let $\mathcal{L}(v)$ denote the set of leaf labels in $\mathcal{T}(v)$. We identify leaves with their labels for the sake of readability.

The *last common ancestor* $\text{lca}_T(l_1, l_2)$ of two leaves l_1, l_2 is the inner node v of T where $\{l_1, l_2\} \subseteq \mathcal{L}(v)$ but $\{l_1, l_2\} \nsubseteq \mathcal{L}(l(v))$ and $\{l_1, l_2\} \nsubseteq \mathcal{L}(r(v))$. The last common ancestor of two nodes is defined accordingly. To *switch* v means that we interchange the order of the children $l(v)$ and $r(v)$ such that the former $L(v)$ becomes $R(v)$, and vice versa, without changing node and leaf orders in $L(v)$ or $R(v)$.

Given two not necessarily complete binary trees S, T with identical leaf sets, a *tanglegram* is a planar embedding of S and T, where the two trees are contrasted in such a way that the leaves of each tree are arranged on one of two parallel straight lines, and identical leaves are connected by additional edges, see Figure 1. The task of the TANGLEGRAM LAYOUT problem is to find a tanglegram which minimizes the number of crossings between leaf label edges.

The BALANCED SUBGRAPH problem is defined as follows: Given an undirected multigraph with m edges, each of which is either labeled with $=$ or with $\neq$, the task is to find a 2-coloring that breaks as few edge labels as possible, that is, the two vertices incident to an $=$-edge must have the same color and those incident to an $\neq$-edge must have different colors.

For ease of presentation, in the following we will assume all graphs to be multigraphs.

3 Transformation

In this section, we present the transformation from a TANGLEGRAM LAYOUT instance to a BALANCED SUBGRAPH instance. This, in turn, can be solved in time $O(2^k n^4)$ for input trees with n leaves that admit a tanglegram with at most k crossings [8]. A similar construction was used by Nöllenburg et al. [9] and by Buchin et al. [4] in the context of approximation and integer linear programming modelling, respectively.

Given two trees S, T, our starting point is an arbitrary tanglegram of the trees. We now transform this tanglegram into an instance of the BALANCED SUBGRAPH problem, that is, an undirected graph G where all edges are labeled '$=$' or '$\neq$'. This graph may contain multiple edges between two vertices.

Let G be a bipartite graph with vertex set $V_S \cup V_T$, where the vertices in V_S (or V_T) correspond to all inner vertices of S (or T, respectively). For each pair

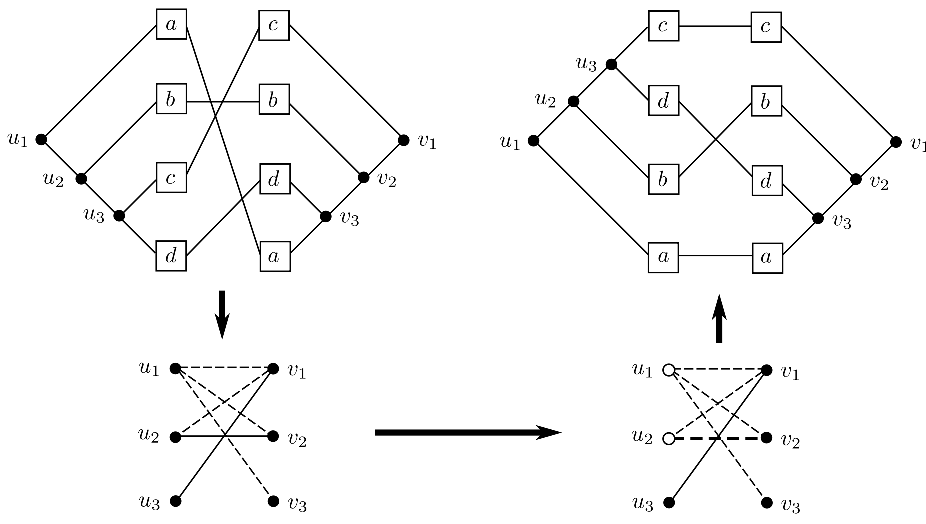

Fig. 2. An example of the transformation to Balanced Subgraph. An arbitrary tanglegram of the input trees (upper left) is transformed into a bipartite graph (lower left). Continuous lines denote =-edges, dashed lines ≠-edges. This instance can be solved by breaking one edge, e.g. $\{u_2, v_2\}$, which leads to a valid 2-coloring of the vertices (lower right). The vertices of one color, here u_1 and u_2, are switched to obtain an optimal tanglegram (upper right).

of leaf labels l_1, l_2, we draw an edge between the vertices corresponding to the last common ancestors $\text{lca}_S(l_1, l_2)$ and $\text{lca}_T(l_1, l_2)$ of the leaves labelled l_1, l_2 in S and T. For each such edge we test whether there is a crossing between edges $l_1 - l_1$ and $l_2 - l_2$ in the current tanglegram. If so, we label the edge with '$\neq$', else with '$=$'.

We use G as input for a Balanced Subgraph algorithm which returns an optimal two-coloring of the vertices of G, such that the vertices incident to =-edges have the same color, whereas those incident to ≠-edges have different colors, while at the same time, breaking as few edges as possible. This corresponds to leaving as few crossings as possible in the tanglegram: The optimal tanglegram is obtained by picking one of the colors, say white, and switching all inner vertices in S and T which correspond to white vertices in G. See Fig. 2 for an example.

Lemma 1. *The balanced subgraph instance generated by the transformation above is solvable with breaking at most k edges if and only if the original trees admit a tanglegram with at most k crossings. The transformation can be done in quadratic time.*

Proof. Let S, T be two trees of a tanglegram instance and $G = (V_S \cup V_T, E)$ be the graph after the transformation.

It is easy to see that a crossing between two labels l_1, l_2 in the initial tanglegram is resolved if and only if exactly one of the last common ancestors $u = \text{lca}_S(l_1, l_2)$ and $v = \text{lca}_T(l_1, l_2)$ is switched. Analogously, if the edges between these labels are not crossed initially, they stay uncrossed if and only if both or none of the inner nodes u and v are switched. In the first case the transformation introduces an $\neq$-edge $\{u, v\}$ to G, in the latter case an $=$-edge $\{u, v\}$.

If an optimal solution for G breaks an edge $\{u, v\}$ that is an $\neq$-edge, u and v are colored equally, so either both u and v or none of them will be switched in the tanglegram. If the broken edge $\{u, v\}$ is an $=$-edge, u and v are colored differently, and thus exactly one of these nodes will be switched. In both cases there is a crossing between the two labels that caused $\{u, v\}$ in the new tanglegram. Similarly, an edge $\{u, v\}$ that is not broken corresponds to a leaf pair that will not have a crossing in the new tanglegram. This shows that there is an exact one-to-one correspondence between leaf pairs in S, T and edges in G: An edge is broken if and only if there is a crossing between the respective leaf labels.

The running time of our transformation for trees S, T with n leaves is $O(n^2)$: Given an arbitrary tanglegram of S, T, we compute the last common ancestors of all leaf pairs in linear time [6]. The sequence of labels in each tree can be obtained in linear time with a standard search algorithm. With this information, we can easily compute in $O(n^2)$ time whether there are crossings between each pair of labels and add the respective edges to the bipartite graph. □

The Balanced Subgraph instance we generate has $2n-2$ vertices and $\binom{n}{2}$ edges, as a binary tree has $n-1$ inner nodes. The running time of the Balanced Subgraph algorithm from [8] is $O(2^k m^2)$ for a graph with m edges, so the total running time of our algorithm is $O(2^k n^4)$. Note, however, that the n^4 factor can be reduced in practice; see Section 5.

Baumann et al. [2] present an ILP for the generalized problem where leaves of the two trees are not necessarily connected by a perfect matching but instead by an arbitrary bipartite graph. We note that the above transformation can be applied to instances of this more general problem, too. This possibility has been also noted in [1].

Our transformation allows us to prove that Balanced Subgraph is NP-complete on bipartite graphs, but this can also be seen directly: For an arbitrary instance of Balanced Subgraph, insert dummy vertices into every edge, and replace an $=$-edge by two $=$-edges, and an $\neq$-edge by an $=$-edge and an $\neq$-edge. The resulting instance is bipartite, and has a solution if and only if the original instance has a solution.

Grötschel and Pulleyblank [7] showed that Edge Bipartization can be solved in polynomial time for weakly bipartite graphs, a class of graphs that includes both bipartite and planar graphs. Hüffner et al. [8] wrongly claimed that using this result, Balanced Subgraph can be solved in polynomial time for (weakly) bipartite input graphs. The reason is that starting from a weakly bipartite instance of Balanced Subgraph, the resulting Edge Bipartization instance is no longer weakly bipartite.

4 A Kernel-like Result for Complete Binary Trees

In parameterized complexity, a *kernel* is a polynomial-time self-reduction of a parameterized problem after which the problem size can be bounded by a function only depending on the parameter. Here, we give a bound of $O(k \log k)$ on the size of the Balanced Subgraph instance which is the result of the transformation from Sect. 3, for the case that the input binary trees are complete. To show this, we begin with some definitions.

For a pair of leaves a, b with $u = \mathrm{lca}_S(a, b)$ and $v = \mathrm{lca}_T(a, b)$, there is a corresponding edge $\{u, v\}$ in G; call this the *edge associated with* a, b. Likewise, a, b is a leaf pair associated with the edge $\{u, v\}$. Obviously, there can be many leaf pairs associated with a single edge.

A pair of *contradictory edges* are two edges between the same pair of nodes in G, with different edge labels. A node u is involved in contradictory edges if there is a pair of contradictory edges with one end in u.

Let a *mirror node* of a node u be a node with identical leaf set. If u has a mirror node v, then the nodes of $\mathcal{T}(u)$ and $\mathcal{T}(v)$ generate a subgraph of G which is separate from the rest of the graph (the subgraph is not necessarily connected, but every edge of G that is incident to a node of $\mathcal{T}(u)$ is incident to a node of $\mathcal{T}(v)$ and vice versa). If furthermore $l(u)$ and $r(u)$ have mirror nodes (necessarily among $l(v)$ and $r(v)$), then v is an *identical mirror* of u, and u and v will end up in a two-vertex component in G. Such a component is a *trivial component* and can be solved immediately. Note that one of $l(u)$ and $r(u)$ has a mirror node if and only if both have mirror nodes.

We need the following reduction rules. The first three are generic, i.e. applicable not only in the case of complete trees; their correctness is immediate.

Rule 1 (Remove contradictory edges) *If there are multiple edges $\{u, v\}$, of which $n_1 > 0$ are $=$-edges and $n_2 > 0$ are $\neq$-edges, let $t = \min\{n_1, n_2\}$. Lower k by t, and delete t edges of each kind.*

Assume from now on that Rule 1 has been applied exhaustively.

Rule 2 (High-multiplicity edges) *If there is any edge $\{u, v\}$ of multiplicity more than k, then set it to permanent.*

The following rule describes how edges set to "permanent" can be contracted.

Rule 3 (Vertex merging) *If an $=$-edge between two vertices u, v is set to permanent, then replace each edge $\{v, w\}$, $w \neq u$ with an equally labeled edge $\{u, w\}$ and delete v.*

If an $\neq$-edge between two vertices u, v is set to permanent, then replace each edge $\{v, w\}$, $w \neq u$ with a contrarily labeled edge $\{u, w\}$ ($=$ becomes $\neq$ and vice versa) and delete v.

Finally, we have two rules which are specific for the complete binary case. The essence is that given the restriction on the tree shapes, a node which does not have a mirror node will have leaves involved in edge crossings in any drawing.

Rule 4 *Let S, T be complete binary trees defining a TL instance. Let two nodes be* incomparable *if they are not in an ancestor-offspring relationship. If there are more than $2k$ pairwise incomparable nodes in S without mirror nodes, then reject the instance.*

Rule 5 *Let S, T be complete binary trees defining a TL instance. If there is any node u in S such that for every node v in T of the same depth as u, more than k leaves of $\mathcal{T}(u)$ are missing from $\mathcal{T}(v)$, then reject the instance.*

Lemma 2. *Rules 4 and 5 are correct.*

Proof. Since the trees S and T are both complete, the grouping of leaves to nodes with respect to any ordering of the trees follows the same structure: every node u in S will be placed directly opposite a node v in T (on the same level as u) sharing the same section of the leaf ordering (e.g. if the leftmost eight leaves of S meet in u, then the leftmost eight leaves of T meet in v). Now for every leaf in $\mathcal{T}(u)$ that is not present in $\mathcal{T}(v)$ there is a matching edge from $\mathcal{T}(u)$ reaching to the left or the right side of $\mathcal{T}(v)$, and since $\mathcal{T}(u)$ and $\mathcal{T}(v)$ are identically positioned, there are as many edges entering $\mathcal{T}(v)$ from the left resp. from the right side of $\mathcal{T}(u)$ as there are edges leaving $\mathcal{T}(u)$ reaching the left resp. the right side of $\mathcal{T}(v)$. These form pairs of edges which must cross.

Rule 5 follows immediately. Rule 4 follows since incomparable nodes have disjoint leaf sets; if the rule applies, then there are more than $2k$ separate edges involved in crossings. $\square$

Let U be the set of all lowest internal nodes in S (i.e. closest to the leaves) which have no mirror nodes. These nodes are pairwise incomparable, so $|U| \leq k$. Furthermore, any internal node of S which is beneath a node in U, or incomparable to all nodes in U, belongs to a trivial component in the balancing graph. Repeating the argument from the root of S (which does have a mirror), we find that what remains of S after Rules 4 and 5 have been checked and trivial components removed is a set of binary trees whose leaves are the nodes of U. In principle, this already gives us a kernel of size $O(k^2)$: at most $O(k \log n)$ nodes remain, and if $\log n > k$, then solving the problem exactly in time $O(2^k n^{O(1)})$ counts as polynomial processing in n. We next show that our reduction rules take care of this in a different way, leaving at most $O(k \log k)$ nodes in the balancing graph.

Theorem 1. *Let S, T be complete binary trees. Applying rules 4 and 5, processing trivial components, and repeatedly merging heavy edges and removing contradictory edges either leads to a rejection of the instance or leaves a balancing graph with at most $O(k \log k)$ remaining nodes.*

Proof. Call a node *fat* if both children have at least $4k$ leaves. We will essentially show that fat nodes contribute nothing to the size of the final graph (because all but a bounded number of them will be merged into other nodes). We make three claims to show the result.

1. Every fat node has an identifiable *partner* on the same level in the opposite tree, which shares the same leaf set with at most k exceptions. If not, then Rule 5 would apply. The same rule holds for the children of a fat node; by a counting argument, the partner matching must map the children of a fat node u to different children of its partner v. In particular, for any fat partner nodes u, v, there is an edge $\{u, v\}$ with multiplicity more than k.
2. Let u be a fat node, with partner v. Let v' be an ancestor of v. We claim that if there is an edge $\{u, v'\}$, then there is such an edge with multiplicity more than k. Let a, b be a leaf pair associated with an edge $\{u, v'\}$, and assume w.l.o.g. that $a \in L(v')$, $b \in R(v')$, and $v \in R(v')$.
   Let $\hat{L}$ be the set of leaves that $l(u)$ shares with its partner, and$\hat{R}$ the same for $r(u)$. By the above, $|\hat{L}|, |\hat{R}| \geq 3k$, and $\hat{L} \cup \hat{R} \subseteq \mathcal{L}(v)$. If $a \in L(u)$, then every pair of leaves a, c for $c \in \hat{R}$ has an associated edge $\{u, v'\}$; if $a \in R(u)$, then the same holds for a, c for $c \in \hat{L}$. Thus the claim is shown.
3. Let u be a fat node with partner v. If u and v are not mirror nodes, then we claim that there is an edge $\{u, v'\}$ where v' is an ancestor of v. Note that by the previous claim, there must then exist such an edge that is heavy. Also recall that if u and v are mirror nodes, then for any node $u' \in \mathcal{T}(u)$ and any edge $\{u', v'\}$, $v' \in \mathcal{T}(v)$.
   Assume $a \in \mathcal{L}(u)$, $a \notin \mathcal{L}(v)$. Then for any pair of leaves a, b with $\mathrm{lca}_S(a, b) = u$ there is an associated edge $\{u, v'\}$ where $v' \notin \mathcal{T}(v)$. If v' is not itself an ancestor of v, then there is another edge $\{u, \mathrm{lca}_T(v, v')\}$, which can be found by combining leaves associated with edges $\{u, v\}$ and $\{u, v'\}$.

The last claim has strong implications about the structure of the balancing graph. In particular, for fat partner nodes u and v, if there is an edge from u to an ancestor v' of v, then for any node v'' between v and v', there is in turn an edge from v'' to an ancestor of its partner (perhaps to the partner of v'). Thus the fat nodes of each connected component in the balancing graph are merged into one. To finalize the proof, we need to bound the number of connected non-trivial components.

Consider a node u with fat children $l(u)$, $r(u)$. If both children have mirror nodes (which are then their partners) then u has an identical mirror node and ends up in a trivial component. Otherwise, u shares a component with at least one child. In either case, we see that no connected non-trivial component contains u but no child of u. Thus the number of nodes that remain after the merging process is bounded by the number of non-fat ancestor nodes of the nodes U previously defined, which is in turn bounded by $O(k \log k)$. □

5 Implementation and Experiments

For our experiments, we used Falk Hüffner's implementation of the Balanced Subgraph algorithm [8]. It is based on a combination of data reduction and iterative compression for solving the unreducible parts. The program consists of about 1900 lines of Objective Caml code and about 300 lines of C code that implements the time-critical compression routine of the iterative compression method. All experiments were run on a dual AMD Opteron 275 machine with 2.2 GHz, 1024 KB cache, and 6 GB main memory running under the Solaris 10 8/07 operating system (only one core was used). The program was compiled with Objective Caml 3.11.1 and the GNU gcc 3.4.3 compiler using the options "-O3 -march=athlon".

Two properties of the instances obtained by the reduction from Tanglegram Layout are notable here. First, they have a particular degree distribution (at least for well-balanced trees): there are vertices with both very low and very high degrees, and the distribution follows a power law, thus the networks are scale-free. Second, there are edges with very high multiplicity (up to several hundred). This works to our advantage. The data reduction rules of the algorithm depend on the existence of small separators, that is, vertex sets whose deletion disconnects the graph. The existence of many small-degree vertices in our instances makes finding such sets likely. Moreover, the exponential part of the running time of the iterative compression algorithm ($O(2^k)$) can be more precisely bounded by $O(2^c)$, where c is the maximum size of a vertex cover needed to cover an (intermediary) balancing set of edges (see [8] for details). Because of the high multiplicity and the existence of "hubs" (vertices with high degree), these vertex covers are much smaller than k.

Another notable property is that Balanced Subgraph instances resulting from our transformation are bipartite. However, since we noted above that arbitrary instances can be made bipartite, it seems unlikely that this can be exploited.

The special structure also motivated us to add two modifications to the solver, both of which are correct for general Balanced Subgraph instances but tailored towards such instances.

First, we added a data reduction rule that can get rid of edges with high multiplicity, without needing to know the value of k. The correctness is easy to see.

Rule 6 (Cut with heavy edge) *Let G be a* Balanced Subgraph *instance, where all pairs of contradictory edges have been removed. If there is an edge cut of G separating two vertices u and v in which at least half the edges of the cut are edges $\{u, v\}$, then the edges $\{u, v\}$ can be made permanent. In particular, this rule applies if there are vertices u and v such that at least half the edges of u are edges $\{u, v\}$.*

After we have decided that an edge is permanent, we can simplify the instance using Rule 3. This rule applies in particular when two nodes in the two trees are similar (that is, they have similar leaf sets, split roughly the same way).

In fact, we implemented only the special case of Rule 6, since our experiments showed that almost always a cut between u and v when an edge $\{u, v\}$ is present either isolates u or isolates v by deleting all adjacent edges. We also did not implement rules that depend on knowing k in advance, such as Rules 2, 4, and 5. The reason is that we either would have to try increasing values of k, which

Table 1. Running times with 60 s time limit

		k		time [s]	
Set	solved [%]	median	maximum	median	maximum
A	69.0	630	58697	0.04	49.30
B	100	83	4639	0.04	2.41
C	39.9	844	8815	1.00	59.79
D	100	172	975	0.06	0.20
E	100	0	10085	< 0.01	2.56
F	99.7	28	34811	< 0.01	31.45
G	100	0	555	< 0.01	2.49

would add a large polynomial factor, or use a heuristic upper bound on k, which is less likely to yield effective reduction.

The second modification concerns the iterative compression process (we assume familiarity with the approach). When building up the instance, we need to add edges one-by-one. Since some instances have extremely many edges (up to 148 240 before data reduction), it is desirable to avoid this factor of n^2. For this, we start with a heuristic solution and compress it repeatedly until no more compression is possible. This typically requires only up to 20 rounds of compression. The initial solution is found using a simple Kernighan–Lin style algorithm: Starting from a random coloring, repeatedly change the color of a vertex as long as this decreases the number of nonsatisfied edges. The disadvantage is that we forfeit the worst-case bound on the running time, and instances can be constructed for which this would give a slowdown. However, for the dense instances we encountered, this is not a problem. To make for a more robust implementation, we could try both methods in parallel.

Data. The seven datasets we used stem from Nöllenburg et al. [9]: Sets A–D are artificial datasets. Set A contains 600 pairs of random complete binary trees of sizes 16–512, set B consists of pairs of mutated complete binary trees, and sets C and D contain 2900 more naturally generated general binary trees with 20–300 leaves and additional mutations in set D. Sets E–F comprise 1303 tree pairs generated with real-world data of animal families. Set E compares Maximum Likelihood and Neighbor joining trees, set F and G Neighbor Joining trees that used different distances. See [9] for details.

The results of the computations are listed in Table 1. We observe that as expected the algorithm struggles the most with sets A and C, which are synthetic random instances which are not expected to have a low number of crossings. From the real-world instances, only 4 instances from set F remain unsolved within a minute. These have $k \geq 10000$.

Instances with $k > 1000$ are unlikely to be of practical interest, since with more than 1000 crossings, the visualization will not be helpful. If we restrict ourselves to the real-world instances with $k \leq 1000$, we can solve all instances with a median of < 0.01 s and a maximum of 2.55 s. This means we can get optimal solutions for all practically relevant instances within seconds.

In general, performance is similar to the ILP approach of Nöllenburg et al. [9], which also can solve most of the instance with k not too high. The advantage of our approach is that it has useful worst-case running time bounds and does not require the proprietary CPLEX software.

6 Conclusion

With improving the previously best-known fixed-parameter running time for the Tanglegram Layout problem from $O(c^k n^{O(1)})$ with $c \approx 1024$ [5] to $O(2^k n^4)$, where k is the minimum number of crossings in a drawing, we managed to make fixed-parameter algorithms applicable for sizes that are interesting for visualization of phylogenetic trees. Experiments showed that we can usually solve instances with $k \leq 1000$ in well below one second.

Consequential challenges are working towards a problem kernel for general binary trees and extending the algorithm to nonbinary phylogenetic trees. We plan to do further algorithm engineering and to integrate the algorithm into the EPoS[4] framework, a modular framework for phylogenetic analysis and visualization, to make it easily available to biologists.

Acknowledgments. We thank Martin Nöllenburg for providing us with the tanglegram datasets from [9].

References

1. M. S. Bansal, W.-C. Chang, O. Eulenstein, and D. Fernández-Baca. Generalized binary tanglegrams: Algorithms and applications. In *Proc. of BICOP 2009*, volume 5462 of *LNCS*, pages 114–125. Springer, 2009.
2. F. Baumann, C. Buchheim, and F. Liers. Exact crossing minimization in general tanglegrams. Technical Report zaik2009-581, Zentrum für Angewandte Informatik Köln, Mar. 2009.
3. K. Buchin, M. Buchin, J. Byrka, M. Nöllenburg, Y. Okamoto, R. I. Silveira, and A. Wolff. Drawing (complete) binary tanglegrams. In *Proc. of Graph Drawing (GD 2008)*, pages 324–335. Springer, 2009.
4. K. Buchin, M. Buchin, J. Byrka, M. Nöllenburg, Y. Okamoto, R. I. Silveira, and A. Wolff. Drawing (complete) binary tanglegrams: Hardness, approximation, fixed-parameter tractability. *arXiv.org*, arXiv:0806.0920v2 [cs.CG], 2008.
5. H. Fernau, M. Kaufmann, and M. Poths. Comparing trees via crossing minimization. In *Proc. of Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2005)*, pages 457–469, 2005.
6. H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proc. of ACM Symposium on Theory of Computing (STOC 1983)*, pages 246–251. ACM, 1983.

[4] `http://bio.informatik.uni-jena.de/epos/`

7. M. Grötschel and W. R. Pulleyblank. Weakly bipartite graphs and the max-cut problem. *Oper. Res. Lett.*, 1(1):23–27, 1981.
8. F. Hüffner, N. Betzler, and R. Niedermeier. Optimal edge deletions for signed graph balancing. In *Proc. of Workshop on Experimental Algorithms (WEA 2007)*, volume 4525 of *LNCS*, pages 297–310. Springer, 2007.
9. M. Nöllenburg, D. Holten, M. Völker, and A. Wolff. Drawing binary tanglegrams: An experimental evaluation. In *Proc. of Workshop on Algorithm Engineering and Experiments (ALENEX 2009)*, pages 106–119. SIAM, 2009.
10. R. D. M. Page, editor. *Tangled Trees: Phylogeny, Cospeciation, and Coevolution.* University of Chicago Press, Chicago, 2002.