Generalized Binary Tanglegrams: Algorithms and Applications

Mukul S. Bansal, Wen-Chieh Chang, Oliver Eulenstein, and David Fernández-Baca

Department of Computer Science, Iowa State University, Ames, IA - 50011, USA
{bansal, wcchang, oeulenst, fernande}@cs.iastate.edu

Abstract. Several applications require the joint display of two phylogenetic trees whose leaves are matched by inter-tree edges. This issue arises, for example, when comparing gene trees and species trees or when studying the co-speciation of hosts and parasites. The *tanglegram layout* problem seeks to produce a layout of the two trees that minimizes the number of crossings between the inter-tree edges. This problem is well-studied for the case when the mappings between the leaves of the two trees is one-to-one. However, in typical biological applications, this mapping is seldom one-to-one. In this work we (i) define a generalization of the tanglegram layout problem, called the *Generalized Tanglegram Layout (GTL)* problem, which allows for arbitrary interconnections between the leaves of the two trees, (ii) provide efficient algorithms for the case when the layout of one tree is fixed, (iii) discuss the fixed parameter tractability and approximability of the GTL problem, (iv) formulate heuristic solutions for the GTL problem, and (v) evaluate our algorithms experimentally.

1 Introduction

The simultaneous examination of a species phylogeny and a gene phylogeny can offer biologists insights into evolutionary processes — such as gene duplication and loss, lateral gene transfer, and deep coalescence — that the inspection of either tree alone cannot provide [6, 13, 9]. The interaction between a gene tree and a species tree is customarily represented by a two-dimensional layout where the leaves of each tree are drawn on separate parallel lines, and where a straight line, called an *inter-tree edge*, connects each leaf (i.e., gene) in the gene tree with the leaf in the species tree (i.e., species) from which the gene was sampled. Each leaf in the species tree may in fact share inter-tree edges with multiple leaves in the gene tree, because a species may have several associated genes. The number of crossings between the inter-tree edges depends on the layout of the trees. A layout with many crossings can be nearly impossible to analyze. In extreme cases, the number of crossings in one drawing can be quadratic in the number of inter-tree edges, while a redrawing of the trees eliminates all crossings. The *tanglegram layout (TL) problem* is to find a layout of the two trees that minimizes the number of crossings.[1] As illustrated in Fig. 1, the layout corresponding to an optimum solution to the TL problem can be a dramatic improvement over an unoptimized layout.

The TL problem has been studied by other researchers (see below); however, previous work has only dealt with the case where the mapping between the leaves of the two

[1] Throughout this work, all trees are assumed to be binary.

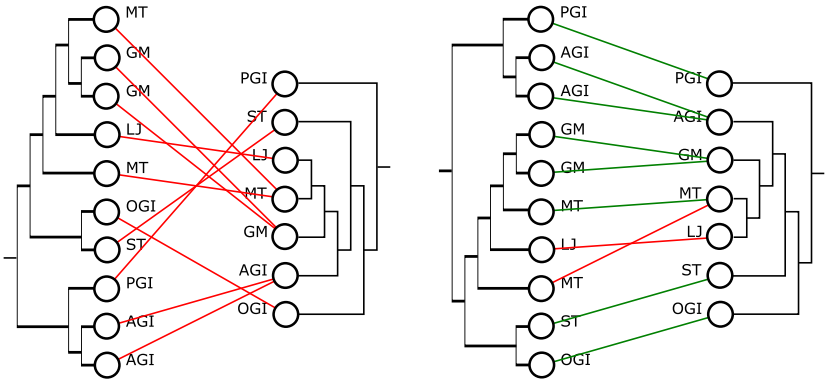

Fig. 1. Two layouts of a gene tree and a species tree, displaying the interactions between their leaves. An arbitrary layout is shown on the left; on the right is the optimized layout that results from solving the TL problem. Data is from [12].

trees is one-to-one. This special case does not handle the more general, and extremely useful, case where the mapping is many-to-many. Here, we define and study the general version of the TL problem.

Background. The version of the TL problem in which the inter-tree edges define a one-to-one mapping between the leaves of the two trees has been extensively studied [4, 5, 8, 2, 10, 7]. We refer to this restricted version as the *single-labeled tanglegram layout (SLTL) problem*. It is known that the SLTL problem is NP-hard [5], even when both the trees are complete [2]. On the positive side, the *one-tree* version of the SLTL problem, where the layout of one of the trees is fixed, can be solved in $O(n \log^2 n)$ time, where n is the number of leaves in the species tree [5]. Moreover, the existence of a planar layout can be verified in linear time [5]. The SLTL is also known to be fixed parameter tractable, where the parameter of interest is the minimum number of crossings in any layout of the two trees [5, 2]. When restricted to complete trees, the SLTL problem is also known to be 2-approximable [2]. Additionally, there is published software that attempts to produce a layout where the number of crossings is small [4, 7].

Our Contribution. In this paper, we define and study the *generalized tanglegram layout (GTL) problem*, in which the number of leaves in the trees may be different and a leaf in either of the trees may share inter-tree edges with multiple leaves in the other tree. The goal again is to produce a layout that minimizes the number of crossings between the inter-tree edges. This generalization of the SLTL problem makes it possible to address not only the gene tree/species tree layout problem, but also those problems in which the inter-tree edges between the trees can be completely arbitrary; such general instances arise in several settings, notably in the analysis of host-parasite co-speciation [11]. The GTL problem has not, to our knowledge, been studied before.

After defining the GTL problem formally, we present two efficient algorithms for its one-tree version, where the layout of one of the trees is fixed. Our algorithms run in time $O(k \log^2 k / \log \log k)$ and $O(kh)$, respectively, where k is the number of inter-tree

edges between the two trees, and h is the height of the tree whose layout can change. Note that for the SLTL problem, k equals n, the number of leaves in each tree. Thus, our first algorithm improves on the best known solution for the one-tree SLTL problem by a factor of $\Theta(\log \log n)$. Based on the result of Fernau et al. [5], we show that the existence of a planar layout (i.e., a layout with no crossings) can be verified in $O(k)$ time. Along the lines of [5, 2], we also discuss the fixed parameter tractability of the GTL problem, along with the approximability of a version of the GTL problem which seeks to maximize the number of non-crossing edges.

We have found that, in practice, one-tree algorithms produce layouts that leave considerable room for improvement. Thus, we have designed and implemented fast heuristics that exploit our one-tree algorithm to reduce the number of crossings by rearranging the layouts of both of the input trees. Although our heuristics do not guarantee an optimal layout, we are able to show empirically that they perform quite well for the range of input sizes that one might expect to encounter in practice. To further strengthen this claim, we make use of an integer quadratic programming formulation of the two-tree problem, based on the work of Nöllenburg et al. [10], to solve the GTL problem exactly for realistic input sizes. Our experiments show that our heuristics perform well in practice and, in all but a few cases, our most comprehensive heuristic returns optimum or near-optimum solutions.

2 Basic Notation and Preliminaries

Given a rooted tree T, we write $V(T)$, $E(T)$, and $L(T)$ to denote its node set, edge set, and leaf set, respectively. A node in $V(T)$ that is not a leaf is called an *internal* node. The root node of T is denoted by $rt(T)$. Given a node $v \in V(T)$, $pa(v)$ denotes the parent of v in T, $Ch(v)$ is the set of children of v, and $T(v)$ denotes the subtree of T rooted at v. If two nodes in T have the same parent, they are called *siblings*. T is *fully binary* if every internal node has exactly two children. Throughout this paper, unless otherwise noted, the term tree refers to a rooted, fully binary tree.

The generalized tanglegram layout problem. Let S and T be two uniquely leaf-labeled trees such that $L(S) = \{1, \ldots, m\}$ and $L(T) = \{1, \ldots, n\}$. Furthermore, let $I(S, T)$ be the set of *inter-tree edges* such that $I(S, T) \subseteq L(S) \times L(T)$ and each leaf node of S and T is incident on at least one edge in $I(S, T)$. Given uniquely leaf-labeled trees S and T and the set of inter-tree edges $I(S, T)$, we denote the resulting instance of the GTL problem by $\langle S, T, I(S, T)\rangle$.

Given a tree T, we say that a linear order τ on $L(T)$ is *compatible* with T if, for each $v \in V(T)$, the leaves in $T(v)$ form an interval (i.e., appear in a continuous block) in τ. We write $u <_\tau v$ to mean that $u \in L(T)$ appears before $v \in L(T)$ in the order τ.

Given compatible linear orders σ and τ on trees S and T respectively, the *number of crossings* between σ and τ with respect to $I(S, T)$, denoted $cr(\sigma, \tau, I(S, T))$, is defined to be $|\{(u, v) \in I(S, T) \colon \text{ either } u <_\tau v, \text{ and } v <_\sigma u, \text{ or } u <_\sigma v, \text{ and } v <_\tau u\}|$.

Problem 1 (GTL Problem) *Given an instance $\langle S, T, I(S, T)\rangle$, find compatible linear orders σ and τ on trees S and T, respectively, such that $cr(\sigma, \tau, I(S, T))$ is minimized.*

Next, we define a restricted version of the GTL problem in which the linear order for one of the input trees is fixed. We call this restricted problem the *One-Tree Generalized Tanglegram Layout (OT-GTL)* Problem. Given uniquely leaf-labeled trees S and T, the set of inter-tree edges $I(S,T)$, and a compatible linear order τ on T, we denote the resulting instance of the OT-GTL problem by $\langle S, T, I(S,T), \tau \rangle$.

Problem 2 (OT-GTL Problem) *Given an instance $\langle S, T, I(S,T), \tau \rangle$, where τ is a compatible linear order on T, find a compatible linear order σ on the tree S such that $cr(\sigma, \tau, I(S,T))$ is minimized.*

3 Solving the OT-GTL Problem

In this section, we provide two algorithms for the OT-GTL problem on the instance $\langle S, T, I(S,T), \tau \rangle$, one with time complexity $O(k \log^2 k / \log\log k)$ and the other with $O(kh)$, where k denotes the cardinality of the set $I(S,T)$ and h is the height of tree S. We also discuss two important special cases of the OT-GTL problem.

With any given planar layout of a tree, we associate a unique linear order. Therefore, when talking about the tree S, we assume that the tree is drawn in the plane with the root node on top and leaves on the bottom. The unique linear order associated with S is then given by the left-to-right order of the leaf labels. Similarly, when talking about the tree T, we assume that the tree T is drawn in the plane with the root at the bottom and the leaves on the top. The unique linear order associated with T is then given by the left-to-right order of the leaf labels. Also observe that, under this setting, every linear order compatible with S (T) defines a unique planar layout for S (T). Thus, if we rotate the layout shown in Figure 1 clockwise by 90 degrees then the tree on the top would be S and the one on the bottom would be T.

We need some notation. Let σ denote the linear order corresponding to some planar layout of S. For any $v \in V(S(v))$, $I(S(v),T)$ denotes the set $\{(u,w) \in I(S,T) \colon u \in L(S(v))\}$. We define the number of crossings $cr(\sigma, \tau, I(S,T), v)$ at node v to be $|\{(u,w) \in I(S(v),T) \colon \text{ either } u <_\tau w, \text{ and } w <_\sigma u, \text{ or } u <_\sigma w, \text{ and } w <_\tau u\}|$.

Let $\overline{\sigma_v}$ denote the linear order corresponding to the planar layout of S obtained by starting with the planar layout corresponding to σ and then swapping the left and right child at the internal node $v \in V(S)$.

The OT-GTL problem seeks to re-draw the tree S such that the linear order σ_{opt} associated with the new layout minimizes the number of crossings $cr(\sigma_{opt}, \tau, I(S,T))$. The task then is to decide, at each internal node, which one of its two children is to be the left child and which one the right child. It is easy to show that this decision can be taken independently at each internal node, irrespective of the decision at the other nodes. Thus, our algorithm starts with a planar layout of S, with the corresponding linear order σ, and then computes, at each internal node v of S, the values $cr(\sigma, \tau, I(S,T), v)$, and $cr(\overline{\sigma_v}, \tau, I(S,T), v)$. If $cr(\sigma, \tau, I(S,T), v) > cr(\overline{\sigma_v}, \tau, I(S,T), v)$, then we swap the left and right child of v. Once this is done at all internal nodes of S, the compatible linear order associated with the new planar layout of S gives the required solution.

A note on handling the input. Recall that $I(S,T)$ is the set of inter-tree edges. Our algorithms assume that given any leaf node in S (or T), we can access all its p neighbors

in the other tree within $O(p)$ time. This can be easily accomplished by associating with each leaf node a set of its neighbors in the other tree. It is possible to construct all these sets within $O(k)$ time. Since all our algorithms require $\Omega(k)$ time, we ignore this additional additive factor. Note, however, that if the leaf labels are arbitrary (and not, as we assume, $\{1, \ldots, n\}$ and $\{1, \ldots, m\}$), then handling the input could require $\Theta(k \log(nm))$ time in the worst case.

3.1 An $O(k \log^2 k / \log\log k)$-Time Algorithm

Our algorithm uses a data structure Ψ for the *subset rank* problem [3]. Such a data structure allows one to maintain a subset $A \subseteq \{1, \ldots, n\}$ under the following operations: *Insert*(i, Ψ), which inserts i into Ψ, *Delete*(i, Ψ), which deletes i from Ψ, and *Rank*(i, Ψ), which, given some $i \in A$, returns the number of elements in A that are less than or equal to i. It is known that each of these operations can be performed in $O(\log n / \log\log n)$ time [3].

We will assume, without any loss of generality, that the initial layout of S is such that at each internal node the number of inter-tree edges incident on leaves of the left subtree is at least as large as the number of inter-tree edges incident on leaves of the right subtree. The linear order corresponding to this layout of S is σ. Our algorithm computes, at each internal node v of S, the value $cr(\sigma, \tau, I(S,T), v)$. Later we explain how a slight modification allows our algorithm to compute the value $cr(\overline{\sigma_v}, \tau, I(S,T), v)$ as well. The algorithm proceeds in three different steps:

Step 1: Modify the tree S into a tree S' as follows: For each $x \in L(S)$, let $A(x)$ denote the set of leaf node neighbors of x according to the set of inter-tree edges $I(S,T)$. For each $x \in L(S)$, replace the leaf x with an arbitrary tree, X on the leaf set $A(x)$ such that the linear order associated with X is a subsequence of the order τ. We refer to this modified version of S as S'. Observe that S' need not be uniquely leaf-labeled, and that the number of leaf nodes in S' must be exactly k (i.e., $|I(S,T)|$). Observe, also, that all the internal nodes in S must be internal nodes in S' as well. The inter-tree edges between S' and T now simply connect those leaf nodes of S' and T that have the same labels. Now, we uniquely relabel the tree S' so that the compatible linear order σ' associated with the layout of S' becomes the ordered list $(1, 2, \ldots, k)$. The set $I(S', T)$ of inter-tree edges between S' and T is correspondingly updated. Note that each leaf node in S' must be incident on exactly one edge of $I(S', T)$.

This step reduces the instance $\langle S, T, I(S,T), \tau \rangle$ of the OT-GTL problem into the instance $\langle S', T, I(S', T), \tau \rangle$. The following lemma relates these two instances.

Lemma 1. *Given S and S' and any internal node $v \in V(S)$, if v' is the corresponding internal node in S', then $cr(\sigma, \tau, I(S,T), v) = cr(\sigma', \tau, I(S', T), v')$.*

Thus, to solve the OT-GTL problem on instance $\langle S, T, I(S,T), \tau \rangle$ it is sufficient to solve it on the instance $\langle S', T, I(S', T), \tau \rangle$.

Step 2: We start with the planar layout of S' corresponding to the linear order $\sigma' = (1, 2, \ldots, k)$ and modify the tree T into a tree T' as follows: For each $x \in L(T)$, let $B(x)$ denote the set of leaf node neighbors of x according to the set of inter-tree edges $I(S', T)$. For each $x \in L(T)$, replace the leaf x with an arbitrary tree, X on the leaf set

$B(x)$ such that the linear order associated with X is a subsequence of the order σ'. We refer to this modified version of T as T', and the corresponding linear order as τ'. The set of inter-tree edges, denoted by $I(S', T')$ now simply connects those leaf nodes in S' and T' that have the same labels. Next, we traverse through S' in post order, and store at each internal node of S', the range of leaf labels in the subtree rooted at that node. Note: This range can be completely specified by two "bounding" integers.

This step reduces the instance $\langle S', T, I(S', T), \tau \rangle$ of the OT-GTL problem into the instance $\langle S', T', I(S', T'), \tau' \rangle$. The following lemma relates these two instances.

Lemma 2. *Given S', T, and T', and any internal node $v \in V(S')$, we must have $cr(\sigma', \tau, I(S', T), v) = cr(\sigma', \tau', I(S', T'), v)$.*

It is worth observing that both S' and T' are uniquely leaf labeled, $L(S') = L(T') = \{1, \ldots, k\}$, and each leaf node in S' and T' is incident on exactly one edge of $I(S', T')$. Thus, by performing steps 1. and 2. we have effectively reduced an instance of the OT-GTL problem into an instance of the one-tree SLTL problem. Our algorithm on this simplified instance $\langle S', T', I(S', T'), \tau' \rangle$ of the OT-GTL problem proceeds as follows.

Step 3: Consider the path from the root of S' to the left most leaf node (in the layout corresponding to σ'). We now compute the value $cr(\sigma', \tau', I(S', T'), v)$ at each internal node v along this path by calling Procedure COMPUTECROSSINGS on S', σ' and τ'. Remove all the nodes along this left most path. This decomposes S' into a forest of subtrees each having at most $k/2$ leaves. Also break up the linear order τ' into separate linear orders, each corresponding to a particular subtree and restricted to its leaf set. Now apply step 3 of this algorithm recursively on each of these subtrees.

Procedure COMPUTECROSSINGS: This procedure takes as input a tree S' drawn according to σ' and a linear order τ', and computes the value $cr(\sigma', \tau', I(S', T'), v)$ at each internal node v along the path P from the root to the left most leaf of S'. Procedure COMPUTECROSSINGS uses the subset rank data structure as follows: Given a leaf x in the tree S', suppose x appears in the right subtree of an internal node y on P. Let z be the largest leaf label in the left subtree of y. Observe that all the leaves in S' with a label smaller than z must be in the left subtree of y. We will use the subset rank data structure to find the number of nodes that appear after the node x in the linear order τ', whose labels are smaller than or equal to z. Each such node, when paired with x, must be a crossing pair at node y. It can be shown that procedure COMPUTECROSSINGS has time complexity $O(l \log l / \log \log l)$, where $l = |L(S')|$. Further details are omitted from this extended abstract.

Our three-step algorithm computes at each internal node $v \in V(S')$, the value $cr(\sigma', \tau', I(S', T'), v)$ in $O(k \log^2 k / \log \log k)$ time. In light of Lemmas 1 and 2, this immediately yields the value $cr(\sigma, \tau, I(S, T), v)$ for each internal node $v \in V(S)$.

The following lemma shows how our algorithm can also compute, at each internal node $v \in V(S)$, the value $cr(\overline{\sigma_v}, \tau, I(S, T), v)$, yielding Theorem 1.

Lemma 3. *Let $\overline{\tau}$ denote the linear order obtained by reversing τ. Then, for any $v \in V(S)$, we must have $cr(\overline{\sigma_v}, \tau, I(S, T), v) = cr(\sigma, \overline{\tau}, I(S, T), v)$.*

Theorem 1. *The OT-GTL problem can be solved in $O(k \log^2 k / \log \log k)$ time.*

3.2 An $O(kh)$-Time Algorithm

We now show that it is possible to solve the OT-GTL problem in $O(kh)$ time, where h is the height of the tree S. This algorithm asymptotically outperforms our $O(k \log^2 k / \log\log k)$-time algorithm when the tree S is (roughly) balanced.

The main idea is to spend at most $O(k)$ time for all the nodes at any fixed level of S. The algorithm follows:

1. Uniquely relabel the tree S so that the compatible linear order σ associated with the layout of S becomes the ordered list $(1, 2, \ldots, m)$. The set $I(S, T)$ of inter-tree edges between S and T is correspondingly updated.
2. Modify the tree T into a tree T' as shown in Step 2. of the $O(k \log^2 k / \log\log k)$-time algorithm. The set of inter-tree edges, denoted by $I(S, T')$, now simply connects those leaf nodes in S and T' that have the same labels.
3. Traverse through S in post order, and store at each internal node the range of leaf labels in the subtree rooted at that node. Note: This range can be completely specified by two "bounding" integers.
4. Let $x = rt(S)$. In the linear order τ', for each element i, one can now determine in constant time whether i is in the left or the right subtree of x in the layout (according to σ) of S. This can be used to compute the value $cr(\sigma, \tau', I(S, T'), x)$ as follows:
   Consider each element i of τ' in order. If i is in the left subtree of x then do nothing and skip to the next i. If i is in the right subtree of x, then, by using counters (after an initial $O(k)$ preprocessing step), we can obtain in $O(1)$ time the number of elements j that occur after i in τ' such that j is in the left subtree of x. Set $cr(\sigma, \tau', I(S, T'), x)$ to be the sum of these values over all i in τ'.
5. Split the linear order τ' into two linear orders, one containing all the leaves from the left subtree of x and the other the leaves from the right subtree. Recursively repeat steps 4 and 5 of this algorithm on the subtrees $S(u)$ and $S(v)$, where $u, v \in \mathit{Ch}(x)$.

Our algorithm computes at each internal node $v \in V(S)$, the value $cr(\sigma, \tau', I(S, T'), v)$ in $O(kh)$ time. The following lemma relates these values to the ones we actually want to compute, and yields Theorem 2.

Lemma 4. *Given S, T and T', and any internal node $v \in V(S)$, we must have $cr(\sigma, \tau, I(S, T), v) = cr(\sigma, \tau', I(S, T'), v)$.*

Theorem 2. *The OT-GTL problem can be solved in $O(kh)$ time, where h is the height of the tree S.*

3.3 Interesting Special Cases

We discuss two special cases of the OT-GTL problem. The first consists of those instances in which the set of inter-tree edges $I(S, T)$ is such that each leaf node of S and T is incident on exactly one edge in $I(S, T)$. This is exactly the SLTL problem discussed in the Introduction. The one-tree SLTL problem has been studied in [5, 4]. Observe that, in this case, we must have $|L(S)| = |L(T)| = |I(S, T)|$. The best known algorithm for this problem runs in $O(n \log^2 n)$ time [5], where $n = |L(S)| =$

$|L(T)|$. Our $O(k \log^2 k / \log\log k)$ time algorithm for the OT-GTL problem becomes an $O(n \log^2 n / \log\log n)$-time algorithm when restricted to this case.

The second case consists of those instances in which the set of inter-tree edges $I(S,T)$ is such that each leaf node of S is incident on exactly one edge in $I(S,T)$. This restricted version of the OT-GTL problem arises naturally when one or more gene trees must be visually compared with a species tree. As seen in the Introduction, species trees have leaves that are labeled uniquely, while each gene trees may have several leaves with the same leaf label. In this scenario, one wishes to produce a planar layout of each given gene tree aligned with the species tree, such that if one were to draw edges between the corresponding leaf labels of the gene tree the species tree, the number of crossings between these edges is minimized. Thus, the gene tree is the tree S and the species tree is the tree T. Note that, in this case, $|I(S,T)| = |L(S)|$. Our $O(k \log^2 k / \log\log k)$ time algorithm for the OT-GTL problem becomes an $O(m \log^2 m / \log\log m)$-time algorithm when restricted to this case, where $m = |L(S)|$. Likewise, our $O(kh)$-time algorithm yields an $O(mh)$ time algorithm.

4 The GTL Problem

The GTL problem is NP-hard, even for simpler special cases [5, 2]. Nöllenburg et al. [10] gave an integer quadratic programming (IQP) based exact solution for the restricted version of the GTL problem in which the set of inter-tree edges defines a one-to-one mapping between the leaves of S and T. Their IQP based approach extends easily to exactly solve the GTL problem as well.

Recall that our $O(k \log^2 k / \log\log k)$-time algorithm converts the instance $\langle S, T, I(S,T), \tau \rangle$ of the OT-GTL problem into the instance $\langle S', T', I(S',T'), \tau' \rangle$ of the one-tree SLTL problem, before applying Step 3.

By a similar (but slightly more involved) transformation, it is also possible to convert any instance $\langle S, T, I(S,T) \rangle$ of the GTL problem into an instance $\langle S', T', I(S',T') \rangle$ of the SLTL problem such that the number of crossings in an optimum solution of the SLTL problem is the same as the number of crossings in an optimum solution of the original GTL problem.[2]

Fixed parameter tractability and approximability. By the above transformation, Fernau et al.'s FPT algorithm for the SLTL problem [5], which runs in time $O^*(c^p)$, where c is a constant and p is the number of crossings in an optimal layout of the two trees, yields an $O^*(c^p)$-time algorithm for the GTL problem as well.[3] The GTL problem is thus fixed parameter tractable. A similar argument shows that, based on the result of Buchin et al. [2], a dual version of the GTL problem, which seeks to maximize the number of non-crossing edges, admits a 0.878-approximate algorithm.

Existence of a planar layout. Fernau et al. [5] showed that, for the SLTL problem, the existence of a planar layout can be verified with-in $O(n)$ time, where $n = |L(S)| =$

[2] Essentially, this new transformation is identical to the one for the one-tree setting, except that instead of replacing the leaves with arbitrary trees on the relevant leaf set, we choose a tree topology which mimics the topology of the other tree restricted to the same leaf set.

[3] The O^* notation ignores the polynomial component of the fixed parameter algorithm.

$|L(T)|$. Their approach also works to test the existence of a planar layout for the GTL problem in $O(k)$ time.

Heuristics. We propose three different heuristic approaches for the GTL problem, which utilize our fast exact algorithms for the OT-GTL problem.

Alternating Heuristic (AH): This heuristic first optimizes the layout of tree S by solving the OT-GTL problem, then optimizes the layout of tree T with respect to the new layout of S, then optimizes S again, and so on. Thus, after each iteration the total number of crossings decreases. The heuristic terminates when there is no further reduction in the crossing value.

Local-Search Heuristic (LH): This is a local search based hill-climbing heuristic in which the local neighborhood is defined by the set of all layouts of T obtained from the layout of T corresponding to τ by swapping the left and right subtrees at an internal node. Each of these $O(n)$ trees in the neighborhood is then evaluated by solving the OT-GTL problem on the tree S with respect to that tree. Among these $O(n)$ layouts of T, the one which enables the OT-GTL algorithm (when run on tree S) to find the lowest crossing value, is set to be the new layout of T for the next local search step. The algorithm terminates when no better layouts can be found during the local search step.

Local-Search Alternating Heuristic (LAH): This heuristic consists of running the Local-Search Heuristic as described above, until it terminates, and then reversing the roles of trees S and T and running the Local-Search heuristic again, and so on. The heuristic terminates when no further improvements in the layout can be observed by performing the role reversal.

Let us assume that we use an $O(x)$ time algorithm to solve the OT-GTL problem. Then, the time complexity of (i) AH is $O(ax)$ where a is the number of iterations, (ii) LH is $O(bnx)$ where b is the number of local search steps, and (iii) LAH is $O(cx(m+n))$ where c is the total number of local search steps performed.

5 Experimental Evaluation

To test the effectiveness and utility of our heuristics for the GTL problem, we implemented and tested them on three kinds of datasets: (i) randomly generated input instances, (ii) gene trees/species trees built using a simple probabilistic model for gene-duplication/loss, and (iii) a real world gene tree/species tree dataset. To evaluate the performance of our heuristics, on each dataset, we compute the corresponding performance ratio $\rho = (cr+1)/(OPT+1)$, where cr denotes the crossing value for the heuristic and OPT the number of crossings in an optimum layout.

Our heuristics use our fast exact algorithms for the OT-GTL problem. For our experiments, we chose to use the $O(kh)$-time algorithm for the OT-GTL problem.[4] This algorithm, as well as the three heuristics were implemented in Python and all experiments we performed on an Intel Core2 Duo 2.4 GHz PC. To test the performance of our heuristics, we also used the IQP based approach to solve the GTL problem optimally. The IQP was solved using the mathematical programming software CPLEX.

[4] Our decision was based on the observation that the $O(kh)$-time algorithm seems to outperform the $O(k \log^2 k / \log\log k)$-time algorithm on the input instance sizes used in our experiments.

Random input instances. We first tested the performance of our heuristics on trees with arbitrary binary topologies and arbitrary inter-tree edges. These instances were generated as follows: We created two random binary trees, both on n leaves and established a random one-to-one correspondence between the leaf sets of the two trees.[5] Next, we created an instance of the GTL problem by adding an additional $\lfloor n \cdot 15/100 \rfloor$ randomly selected inter-tree edges. We performed experiments for values of n ranging from 10 to 800. The results of our experiments are depicted in Table 5. The performance ratio ρ, averaged over all $n \in \{10, 20, 30, 40, 50\}$, for one-tree (IT), AH, LH, and LAH, respectively, are 1.61, 1.12, 1.02, and 1.003. It can be seen that LAH performs remarkably well: Out of the 50 input instances for which the optimum solution was known, LAH found an optimum layout in 41 instances. AH also performs quite well and produces an optimized layout almost instantaneously even for large input instances.

Table 1. Performance of our heuristics on randomly generated input instances. For each n, the table shows (i) the number k of inter-tree edges, (ii) the number of crossings in a random layout, (iii) the number of crossings, the number of iterations performed, and the time, in seconds, required to produce the layout, for the one-tree case (depicted by 1T) and for the three heuristics AH, LH, and LAH, and (iv) the number of crossings in an optimal layout. All values have been averaged over 10 runs. Note that we have not reported the number of iterations for LAH; these were always observed to be between 2 and 3 times the number of iterations for LH. Due to excessive running times, we did not compute optimal solutions for trees with $n > 50$, and did not apply LH and LAH on trees with $n > 200$.

Input			1T		AH			LH			LAH		OPT
n	k	cr	cr	time	cr	iter	time	cr	iter	time	cr	time	cr
10	11	24.9	13.5	0.00	7.9	3.1	0.00	6.2	3.4	0.01	5.8	0.04	5.8
20	23	113.3	85.8	0.00	61.6	3.2	0.00	57.5	8.5	0.11	57.1	0.22	56.8
30	34	254.9	209.9	0.00	164.9	3.7	0.00	158.0	13.4	0.43	157.8	0.64	157.1
40	46	500.6	406.6	0.00	339.5	3.8	0.01	324.1	16.5	1.07	323.7	1.51	322.9
50	57	824.3	623.6	0.00	527.9	3.4	0.01	513.6	18.4	2.03	512.8	2.84	510.7
100	115	3283.3	2765.1	0.00	2482.7	3.7	0.04	2431.7	42.0	23.58	2431.5	26.99	-
200	230	12868.7	11806.9	0.01	10938.9	3.9	0.10	10670.9	83.8	226.53	10668.7	249.74	-
400	460	52139.5	48623.3	0.03	45904.9	3.9	0.29	-	-	-	-	-	-
800	920	211531.8	199728.2	0.12	193667.0	4.1	1.03	-	-	-	-	-	-

Simulated gene trees/species trees. Next, we tested the performance of our heuristics on simulated datasets created by using a simplified birth-death process that mimics gene duplication and loss (see, for example, [1]). To build our trees, we first generated a random binary tree S with n leaves. We then generated a simulated gene tree based on S according to the following probabilistic scheme: At each internal node v in S, the subtree $S(v)$ either duplicates with probability d, is lost with probability r, or remains intact with probability $1 - d - r$. For our experiments we chose d and r to be 0.1 and 0.12 respectively. The value of n ranged from 10 to 800. Table 2 depicts the results of our experiments. The performance ratio ρ, averaged over all $n \in \{10, 20, 30, 40, 50\}$,

[5] Each random tree was created by recursively, and randomly, bi-partitioning its set of leaves.

for one-tree (IT), AH, LH, and LAH, respectively, are 1.33, 1.12, 1.0004, and 1.0004. Both LH and LAH perform equally well in this case; in particular, out of the 50 input instances for which an optimum solution was known, both LH and LAH found an optimum layout in 48 instances.

Table 2. Performance of our heuristics on simulated gene trees/species trees. The layout is identical to that of Table 5, and m denotes the number of leaves in the gene tree.

Input			1T		AH			LH			LAH		OPT
n	m	cr	cr	time	cr	iter	time	cr	iter	time	cr	time	cr
10	10.0	15.2	1.2	0.00	1.2	2.0	0.00	1.2	2.0	0.00	1.2	0.02	1.2
20	16.9	44.6	10.5	0.00	6.9	2.1	0.00	5.6	2.4	0.03	5.6	0.10	5.6
30	34.3	390.9	126.8	0.00	125.8	2.1	0.00	92.7	4.5	0.24	92.7	0.52	92.7
40	44.0	631.1	110.6	0.00	109.7	2.2	0.00	83.5	4.1	0.34	83.5	0.79	83.1
50	70.0	2098.1	913.9	0.00	845.2	2.5	0.01	767.5	7.1	1.98	767.5	3.46	767.4
100	93.5	2757.4	815.9	0.00	748.0	2.5	0.02	515.0	8.1	5.04	515.0	7.89	-
200	269.9	31671.8	9128.1	0.02	8407.6	2.8	0.13	7012.0	22.6	172.51	7012.0	219.27	-
400	360.5	35941.7	2942.7	0.02	2823.5	3.1	0.17	-	-	-	-	-	-
800	741.6	153141.1	29695.2	0.09	27854.6	3.0	0.59	-	-	-	-	-	-

Empirical Dataset. Finally, we tested our heuristics on a real-world empirical dataset [12] on Angiosperms. This data set consists of 588 gene trees with number of leaves ranging from 4 to 94 and a species tree with 7 taxa. On the unoptimized input, the average performance ratio ρ for each gene tree/species tree pair is 10.12, while the corresponding values for 1T, AH, LH, and LAH are 2.32, 1.62, 1.02 and 1.0005 respectively. In particular, there were only one and six input instances, respectively, for which LAH and LH did not find an optimum layout. On all the 588 input instances together, AH took 0.21 seconds, LH took 2.4 seconds, and LAH took a total of 12.5 seconds. Figure 5 depicts the distribution of crossing values after applying the one-tree algorithm and the three heuristics. Indeed, after applying LH and LAH to the dataset, a majority of the input instances show no crossings, a dramatic improvement over the initial layout.

Comparison against other heuristics. Nöllenburg et al., in [10], introduced a heuristic for the SLTL problem. This heuristic has a time complexity of $O(8^h + n^2h)$, where h is the minimum height of the two trees. Since this is exponential in the height of the tree, they also proposed a branch-and-bound algorithm, referred to as *rec-split-bb*, to curb the time complexity of the heuristic. As shown earlier, the GTL problem can be reduced to an instance of the SLTL problem, and consequently, it is possible to use rec-split-bb as a heuristic for the GTL problem. Though rec-split-bb seems to be quite efficient in practice, there is no polynomial bound on its running time. All our heuristics are guaranteed to terminate within polynomial time. Our heuristics LH and LAH also seem to perform at least as well as the rec-split-bb heuristic (based on the experimental results reported in [10] for random general input trees), if not significantly better.

Acknowledgements. The authors wish to thank Dan Gusfield for introducing them to the tanglegram layout problem and for many helpful discussions, and Balaji Venkat-

achalam for bringing relevant previous work to their attention. This work was supported in part by NSF grant no. 0334832.

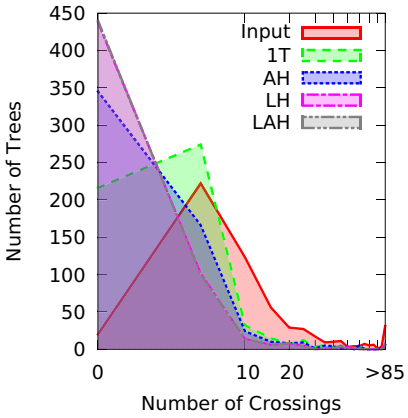

Fig. 2. Distribution of crossing values for the empirical gene tree/species tree dataset after applying various tanglegram layout algorithms. The distributions for LH and LAH are almost identical.

References

1. L. Arvestad, A.-C. Berglund, J. Lagergren, and B. Sennblad. Gene tree reconstruction and orthology analysis based on an integrated model for duplications and sequence evolution. In *RECOMB*, pages 326–335, 2004.
2. 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. In *Graph Drawing*, 2008.
3. P. F. Dietz. Optimal algorithms for list indexing and subset rank. In *WADS*, pages 39–46, 1989.
4. T. Dwyer and F. Schreiber. Optimal leaf ordering for two and a half dimensional phylogenetic tree visualisation. In *InVis.au*, pages 109–115, 2004.
5. H. Fernau, M. Kaufmann, and M. Poths. Comparing trees via crossing minimization. In *FSTTCS*, pages 457–469, 2005.
6. M. Goodman, J. Czelusniak, G. W. Moore, A. E. Romero-Herrera, and G. Matsuda. Fitting the gene lineage into its species lineage. a parsimony strategy illustrated by cladograms constructed from globin sequences. *Systematic Zoology*, 28:132–163, 1979.
7. D. Holten and J. J. van Wijk. Visual comparison of hierarchically organized data. In *10th Eurographics/IEEE-VGTC Symposium on Visualization (EuroVis'08)*, 2008.
8. A. Lozano, R. Y. Pinter, O. Rokhlenko, G. Valiente, and M. Ziv-Ukelson. Seeded tree alignment and planar tanglegram layout. In *WABI*, pages 98–110, 2007.
9. W. P. Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.
10. M. Nöllenburg, D. Holten, M. Völker, and A. Wolff. Drawing binary tanglegrams: An experimental evaluation. *CoRR*, abs/0806.0928, 2008. An updated version of this manuscript will appear in the proceedings of *ALENEX 2009*.

11. R. D. M. Page, editor. *Tangled Trees: Phylogeny, Cospeciation, and Coevolution*. University of Chicago Press, 2002.
12. M. Sanderson and M. McMahon. Inferring angiosperm phylogeny from EST data with widespread gene duplication. *BMC Evolutionary Biology*, 7(Suppl 1):S3, 2007.
13. M. Syvanen. Cross-species gene transfer; implications for a new theory of evolution. *J Theor Biol*, 112, 1985.