Generalized k-ary tanglegrams on level graphs: a satisfiability-based approach and its evaluation[☆]

Andreas Wotzlaw[a,1,*], Ewald Speckenmeyer[a], Stefan Porschen[b]

[a] *Institut für Informatik, Universität zu Köln, D-50969 Köln, Germany*
[b] *Fachgruppe Mathematik, Fachbereich 4, HTW-Berlin, D-10318 Berlin, Germany*

Abstract

A tanglegram is a pair of (not necessarily binary) trees on the same set of leaves with matching leaves in the two trees joined by an edge. Tanglegrams are widely used in computational biology to compare evolutionary histories of species. In this work we present a formulation of two related combinatorial embedding problems concerning tanglegrams in terms of CNF-formulas. The first problem is known as the planar embedding and the second as the crossing minimization problem. We show that our satisfiability-base encoding of these problems can handle a much more general case with more than two, not necessarily binary or complete, trees defined on arbitrary sets of leaves and allowed to vary their layouts. Furthermore, we present an experimental comparison of our technique and several known heuristics for solving generalized binary tanglegrams, showing its competitive performance and efficiency and thus proving its practical usability.

Keywords: satisfiability, mixed Horn formula, 2-CNF, level graph, planar embedding, tanglegram, crossing minimization, graph drawing, computational biology, combinatorial optimization

1. Introduction

In recent time the interest in designing exact algorithms and efficient heuristics providing better performance ratio for various variants of the tanglegram problem has increased. This is primarily motivated by their broad applications in computational biology, especially in phylogenetics.

In this work we introduce *generalized k-ary tanglegrams on level graphs*, a generalization of the well-known binary tanglegrams, and study two combina-

[☆] A preliminary version of this paper appeared in [1].

[*] Corresponding author

Email addresses: wotzlaw@informatik.uni-koeln.de (Andreas Wotzlaw), esp@informatik.uni-koeln.de (Ewald Speckenmeyer), porschen@htw-berlin.de (Stefan Porschen)

[1] The first author was partially supported by the DFG project under the grant No. 317/7-2

torial embedding problems connected with them. A *binary tanglegram* [2] is an embedding (drawing) in the plane of a pair of rooted binary trees whose leaf sets are in one-to-one correspondence (perfect matching), such that the matching leaves are connected by *inter-tree edges*, called sometimes *tangles* or *tangle edges*. Clearly, the number of crossings between the inter-tree edges depends on the layout of the trees. From a practical point of view, an embedding with many crossings can hardly be analyzed. Figure 1 shows an example of a binary tanglegram coming from phylogenetic studies done by Charleston and Perkins [3]. Taking this into account, the first problem one can consider here consists of determining an embedding of one or both trees such that the inter-tree edges do not cross, if such an embedding exists. This problem is known as the *planar embedding* problem. If such a planar embedding is not possible, then we may want to find an embedding with as few crossing inter-tree edges as possible. This second problem, *crossing minimization*, is known in the literature also as the *tanglegram layout* problem [4, 5, 6].

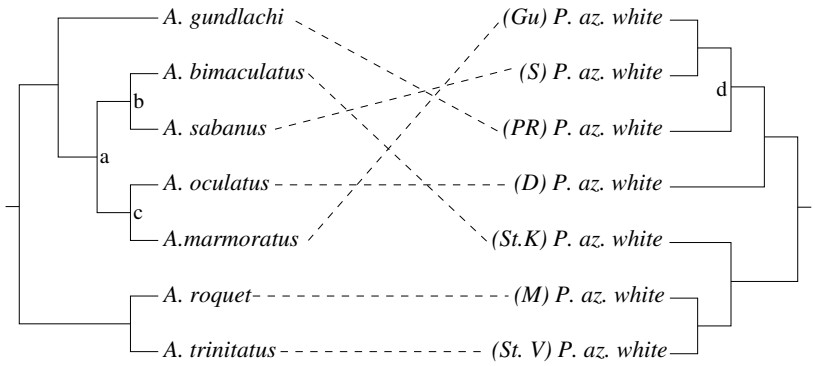

Figure 1: A binary tanglegram from [3] showing phylogenetic trees for lizards (left tree) and strains of malaria (right tree) found in the Caribbean tropics. The dashed lines represent the host-parasite relationship. Here, the number of crossings is 7. This can be reduced to 1 by interchanging the children of nodes a, b, c, and d.

Both problems belong to the area of graph drawing [7] and are motivated by the desire to find a good display of hierarchical structures, e.g., in software engineering, project management, or database design. For instance, tanglegrams occur when analyzing software projects in which trees are used to represent package, class, or method hierarchies. Changes in hierarchies can be analyzed over time, or automatically generated decompositions can be compared with human-made ones. This application yields tanglegrams on trees that are not binary in general [8].

Matching and aligning trees is also a recurrent problem in computational biology [2]. Embeddings with fewer crossings or with matching leaves close together are useful in biological analysis [6]. Here, prominent applications are in particular the comparisons of phylogenetic trees [3, 9, 10], which are used to represent a hypothesis of the evolutionary history (phylogeny) of a set of species.

These species are drawn as the leaves of the tree whereas their ancestors are represented by the inner nodes. Different hypotheses may lead to a set of different candidate trees. An embedding imposes an order among the leaves of the trees. Therefore, comparing the drawings of the trees is equivalent to comparing the permutations of the leaves. In general, the simultaneous examination of a species phylogeny and a gene phylogeny can offer biologists more insights into evolutionary processes, e.g., gene duplication, gene extinction, or cross-species gene transfer, that the inspection of either tree alone cannot provide [11, 12].

Bansal et al. [4] analyzed *generalized tanglegrams* where the number of leaves in the two binary trees may be different and a leaf in one tree may match multiple leaves in the other tree, thus no perfect matching is required here. They pointed out that such a generalization of the problem makes it possible to address not only the gene tree and species tree embedding 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, e.g., in the analysis of host-parasite cospeciation [2, 13].

Related work. Crossing minimization in tanglegrams has parallels to crossing minimization in graphs. Computing the minimum number of crossings in a graph is NP-complete [14]. However, it can be verified in linear time whether a graph has a planar embedding [15]. The last assertion holds also for a more special case of level graphs [16, 17]. Computing the minimum number of crossings is fixed-parameter tractable [5, 18]. Analogously, crossing minimization in tanglegrams is NP-complete, as shown by Fernau et al. [19] by a reduction from the MAX-CUT problem [20], while the special case of planarity test can be executed in linear time [19].

Furthermore, the problem of minimizing the number of crossings where one tree is fixed and the layout of the other tree is allowed to vary can be solved efficiently. For binary trees with arbitrary topology, Fernau et al. [19] showed an $O(n \log^2(n))$ solution, further improved to $O(n \log^2(n)/\log\log(n))$ by Bansal et al. [4]. Here, n denotes the number of leaves in each tree. Venkatachalam et al. [6] provided recently an algorithm working on the integer linear programming (ILP) formulation of the problem with the so far best-known time bound of $O(n \log(n))$.

Recently Buchin et al. [5] have proved that under the widely accepted Unique Games Conjecture [21] there is no constant factor approximation algorithm for minimizing number of crossings of binary tanglegrams. Nöllenburg et al. [8] gave an extensive experimental evaluation of some heuristics, an exact branch-and-bound algorithm, and an ILP-based approach for binary tanglegrams. Finally, Baumann et al. [22] by exploiting the fact that crossing minimization in (not necessarily binary) tanglegrams can be seen as a generalization of bipartite crossing minimization, formulated it as a quadratic linear ordering problem with some additional side constraints and evaluated it by using semidefinite optimization and the mathematical programming software CPLEX [23].

For the case of generalized tanglegrams where the layout of one tree is fixed, Bansal et al. [4] presented two algorithms with running times $O(mh)$

and $O(m \log^2(m)/\log\log(m))$, where m is the number of edges between the two trees and h is the height of the tree whose layout can change. To obtain those polynomial running times, the layout of the other tree must be fixed. Based on the result of Fernau et al. [19], they also showed that the existence of a planar embedding can be verified in $O(m)$ time.

Our contribution. In our generalization of the tanglegram problem we go even further than Bansal et al. [4]. In *generalized k-ary tanglegrams on level graphs* we consider problem instances with more than two k-ary trees where every tree is defined on an arbitrary set of leaves. Notice that here the pairwise disjoint leaf sets and the corresponding inter-tree edges (no perfect matching required any more) connecting two neighboring leaf sets constitute a level graph [17] where each level is defined by some leaf set. Thus, each tree defined on some level implies additional constraints reducing considerably the set of possible embeddings. For instance, k-ary trees with n leaves allow for at most $k!^{\frac{n-1}{k-1}}$ different leaf orderings implied by different orderings of the subtrees, i.e., 2^{n-1} in case of binary trees, compared with $n!$ permutations if no restrictions are imposed on the order of the leaves. Further, in our extended definition we do not restrict the tanglegrams only to binary trees, but consider rooted k-ary trees in which each node has not more than k children, for some fixed integer $k > 1$.

In this study we are interested in planar embeddability problems of generalized tanglegrams on level graphs. More specifically, we investigate the simultaneous existence of a planar embedding of the inter-tree nodes on some horizontal plane with planar embeddings of the trees on separate vertical planes, one for each tree. Our intention is to present all of them nicely on at least two orthogonal planes. In the following, we call the existence of such an embedding shortly a planar embedding of a generalized tanglegram on a level graph.

In our approach we encode the planarity test and the crossing minimization problem on generalized tanglegrams on level graphs in terms of CNF-formulas by incorporating ideas used already for level graphs in [17, 24]. By doing this, the planarity test essentially reduces to testing satisfiability of some 2-CNF formula. The crossing minimization problem has a formulation as a PARTIAL MAX-SAT problem of some CNF formula with a mandatory part of 3- and 2-clauses that must be satisfied for the solution to be reasonable, and a second part of 2-clauses such that its truth assignment must satisfy as many of these clauses as possible. In the mandatory part, the 3-clauses reflect transitivity conditions forced by the genus of the surface, whereas the 2-clauses reflect antisymmetry conditions. These clauses have to be satisfied in order to obtain a layout. The second part of 2-clauses reflects non-crossing conditions. Each unsatisfied clause from this part represents one arc crossing. This formulation offers a simple alternative for finding reasonable approximate solutions of the crossing minimization problem. We show that the planarity test of a generalized tanglegram on a level graph having a total of n vertices and with k-ary trees defined on each level, for some fixed integer $k > 1$, can be solved in $O(n^2)$ time by an elementary 2-SAT algorithm. Finally, to the best of our knowledge, this is the first time that the generalized tanglegram problem has been treated by

means of a satisfiability encoding. According to the experimental comparison of our technique with several well performing heuristics of Bansal et al. [4] and some ILP-based approach for solving generalized binary tanglegrams, and our satisfiability-based approach shows its competitive performance and efficiency and thus proves its practical usability.

Roadmap. The rest of the paper is organized as follows. In Section 2 we provide some basic notation and definitions of relevant computational problems for generalized tanglegrams on level graphs. The satisfiability-based formulation of the two main problems on generalized tanglegrams on level graphs is given in Section 3. In Section 4, we present the results of an experimental comparison of our technique with the heuristics designed by Bansal et al. [4] and some ILP-based approach for solving generalized binary tanglegrams. Here, the most important criterion is the performance ratio with respect to the optimal solution in terms of the number of crossings. Finally, in Section 5 we conclude our work and state some open questions.

2. Preliminaries and basic notation

Formally, a *level graph* is a triple (G, λ, L) where $G = (V, E)$ is a directed graph, $L = \{1, ..., |L|\}$ is the set of levels, and $\lambda : V \to L$ is the level-mapping, that assigns the vertices to levels such that each arc is directed from a lower to a higher level, i.e., $\forall e = (u, v) : \lambda(v) > \lambda(u)$. For simplicity, we identify the above triple by G having the other two components in mind. Observe that there exists no arc between vertices on the same level. If in addition, for every arc $e = (u, v) \in E, \lambda(v) = \lambda(u) + 1$ holds, then the level graph is called *proper*. In the present paper we consider proper level graphs only, hence we simply will speak of level graphs. This restriction means no loss of generality since an arbitrary level graph can be turned into a proper one preserving the crossing number by simply adding dummy vertices as shown in [25, 17].

Level graphs are drawn in the Euclidean x, y-plane by linear order, i.e., all vertices on the same level $j \in L$ are placed at arbitrary different positions on the line $y = j$; the x-coordinate of vertex u is denoted as $x(u)$. Arcs are represented by straight lines between the points representing their incident vertices. Often arrows at arc heads are omitted since the direction is implicitly fixed by the levels. For two vertices u, v on the same level, we simply write $u < v$ iff $x(u) < x(v)$. One is especially interested in level-graph drawings such that no two arc lines cross outside their endpoints. A level graph for which such a drawing exists is called *level-planar*. It is not hard to see that a level graph with $|E| > 2|V| - 4$ cannot be level-planar [17]. Therefore, for most level graphs all what one can hope for is to find a plane embedding such that the number of arc-crossings is minimized. Moreover, by reduction from the FEEDBACK ARC SET problem [20], Eades and Wormald [25] showed that crossing minimization in level graphs is NP-hard, even if there are only two levels with a fixed order of nodes on one level.

In generalized tanglegrams on level graphs, we define additionally on the nodes of each level $i \in L$ of a level graph G a tree T_i with nodes of level i as its leaf set. Clearly, the presence of a tree on each level reduces the search space of admissible embeddings considerably. More formally, a generalized tanglegram on a level graph G is a quadruple (G, λ, L, F) where $F = \{T_1, ..., T_{|L|}\}$ is a forest of *level-trees* and G, λ, and L are defined as above. We say that a rooted level-tree is *complete* if all its leaves have the same depth. Given a rooted, unordered tree $T \in F$, we write $V(T)$, and $E(T)$ to denote its node set, and edge set, respectively. Furthermore, for two trees T_i and T_{i+1} from F defined on two adjacent levels i and $i+1$ of level graph G, we define the set of *inter-tree arcs* as

$$E(T_i, T_{i+1}) := \{(u, v) \in E(G) : \lambda(u) = i, \lambda(v) = i + 1\}.$$

Observe that for a proper graph G holds $E(G) = \bigcup_{i=1,...,|L|-1} E(T_i, T_{i+1})$.

For each node $v \in V(T)$, let $T(v)$ denote the subtree of T rooted at v. Given a tree T, we say that a linear order σ on the leaves of T is *compatible* with T if for each node $v \in V(T)$ the leaves in $T(v)$ form an interval (i.e., appear as a consecutive block) in σ. We write $u <_\sigma v$ to mean that leaf u appears before leaf v in the linear order σ on the leaves of T. Given compatible linear orders σ_i and σ_{i+1} on two trees T_i and T_{i+1} from F defined on two adjacent levels i and $i+1$ of level graph G, respectively, the *number of crossings* between σ_i and σ_{i+1} among the inter-tree arcs $E(T_i, T_{i+1})$ is defined as

$$\tau(\sigma_i, \sigma_{i+1}) := \big|\big\{\{(u, a), (v, b)\} \subseteq E(T_i, T_{i+1}) : \neg\big((u <_{\sigma_i} v) \leftrightarrow (a <_{\sigma_{i+1}} b)\big)\big\}\big|.$$

Note that a pair of arcs cross at most once (see Figure 2). Moreover, since we assume here that G is a proper level graph, only adjacent levels can induce crossings. Finally, the overall number of crossings for an instance (G, λ, L, F) and a set $S := \{\sigma_1, ..., \sigma_{|L|}\}$ of compatible orders for each level in L (tree in F) is defined as

$$\tau(G, \lambda, L, F, S) := \sum_{i=1,...,|L|-1} \tau(\sigma_i, \sigma_{i+1}).$$

Now the main combinatorial problems addressed already informally in Section 1 can be defined as follows:

Problem 1 (Planarity Test). Given an instance (G, λ, L, F), verify if there exists a planar embedding, i.e., if there exists some set S of compatible linear orders σ_i for each level $i \in L$ (tree $T_i \in F$) such that $\tau(G, \lambda, L, F, S) = 0$.

Problem 2 (Crossing Minimization). Given an instance (G, λ, L, F), find a set S of compatible linear orders σ_i for each level $i \in L$ (tree $T_i \in F$) such that $\tau(G, \lambda, L, F, S)$ is minimized.

To complete the notation, let CNF denote the set of formulas (free of duplicate clauses) in conjunctive normal form over a set $V = \{x_1, ..., x_n\}$ of propositional variables $x_i \in \{0, 1\}$. Each variable x induces a positive literal (variable

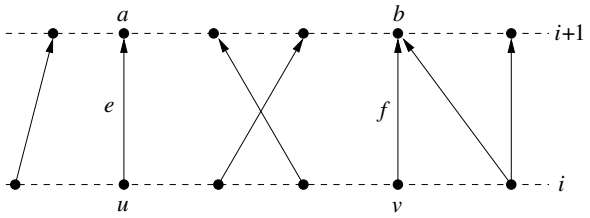

Figure 2: Adjacent levels i and $i+1$ of a level graph G. Arcs $e = (u, a)$ and $f = (v, b)$ have different tails and heads.

x) or a negative literal (negated variable $\overline{x}$). Each formula $C \in$ CNF is considered as a clause set $C = \{c_1, ..., c_{|C|}\}$. Each clause $c \in C$ is a disjunction of different literals l_i, and is also represented as a set $c = \{l_1, ..., l_{|c|}\}$. A clause is termed a k-clause, for some $k \in \mathbb{N}$, if it contains at most k literals. The number of clauses in C is denoted by $|C|$. For $k \in \mathbb{N}$, let k-CNF denote the subset of CNF formulas such that each of its clauses has length at most k. We denote by $V(C)$ the set of variables occurring in formula C. The satisfiability problem (SAT) asks, whether formula C is *satisfiable*, i.e., whether there is a truth assignment $t : V(C) \to \{0, 1\}$ setting at least one literal in each clause of C to 1. Given $C \in$ CNF, the optimization version MAX-SAT searches for a truth assignment t satisfying as many clauses of C as possible.

3. Satisfiability formulation of crossing minimization

In the following we provide a formulation of the crossing minimization problem for generalized tanglegrams on level graphs in terms of propositional logic. We proceed in two steps. Given a generalized tanglegram (G, λ, L, F), we first show the construction of CNF-formulas for the level graph (G, λ, L). In the second step, we describe a similar construction for the forest F of the generalized tanglegram. The conjunction of the resulting subformulas will give then a SAT encoding of the input problem.

Consider in a proper level graph G two subsequent levels i and $i+1$ from L, as shown in Figure 2. Let $e = (u, a)$ and $f = (v, b)$ be two arcs from $E(T_i, T_{i+1})$ directed from level i to level $i+1$ with different tails $u \neq v$ and different heads $a \neq b$. In a drawing of G, e and f do not cross iff

$$u <_\sigma v \quad \Leftrightarrow \quad a <_\sigma b$$

for some linear order σ. Observe that arcs having the same head or tail never cross in any drawing of G.

The construction of a Boolean formula C_G representing the plane embedding of G proceeds as follows:

1. For each level $i \in L$ and every pair $\{u, v\}$ of distinct vertices from level i, i.e., $\lambda(u) = \lambda(v) = i$, create a Boolean variable uv that is true iff $u <_\sigma v$ for some linear order σ.

2. Create the following Boolean subformulas:
   (i) For each level $i \in \{1, ..., |L| - 1\}$ and every two arcs $e = (u, a), f = (v, b)$ from $E(T_i, T_{i+1})$ having their tails $u \neq v$ on level i and heads $a \neq b$ on level $i + 1$, form the non-crossing preserving expression:

$$uv \leftrightarrow ab$$

   (ii) For each level $i \in \{1, ..., |L|\}$ and each pair $\{u, v\}$ of distinct vertices on level i, form the antisymmetry expression:

$$uv \leftrightarrow \overline{vu}$$

   (iii) For each level $i \in \{1, ..., |L|\}$ and each triple $\{u, v, w\}$ of distinct vertices on level i, form the transitivity expression:

$$uv \wedge vw \rightarrow uw$$

Observe that the conjuncted subformulas resulting from (i) and (ii) yield 2-CNF formulas C_i and C_{ii} via application of

$$a \leftrightarrow b \equiv (\overline{a} \vee b) \wedge (\overline{b} \vee a),$$

respectively. Similarly, the subformulas resulting from (iii) yield a Horn formula C_{iii} with clauses of length 3 via elementary equivalence

$$(a \wedge b \rightarrow c) \equiv (\overline{a} \vee \overline{b} \vee c).$$

Recall that each clause of a Horn formula contains at most one positive literal. Hence the formula $C_G = C_i \wedge C_{ii} \wedge C_{iii}$ encoding the plane embedding of a level graph G is a mixed Horn formula [26]. If G has n vertices distributed over $|L|$ levels then C_G has $|V(C_G)| \in O(n^2)$ variables. Moreover, by counting $|C_i| \in O(|E(G)|^2)$, $|C_{ii}| \in O(n^2)$, and $|C_{iii}| \in O(n^3)$. Therefore the number of clauses in C_G is bounded by $O(n^3 + |E(G)|^2)$. As mentioned before, the maximal number of arcs in a level-planar graph containing $n > 2$ nodes is at most $2n - 4$. Thus in the case we use C_G for a level planarity test, a preprocessing ensures that only $O(n^2)$ 2-clauses in C_i are generated. The following result shows that the level planarity test can be formulated as a satisfiability problem.

Proposition 1 ([17]). *A level graph G with n vertices has a level-planar embedding iff $C_G - C_{iii}$ is satisfiable. The test can be done in time $O(n^2)$.*

According to [17], the transitivity formula C_{iii} is superfluous for the level planarity test. This results in a better complexity of $O(n^2)$, since SAT for 2-CNF formulas can be decided in linear time in the number of variables and clauses in the input formula [27].

Minimizing the number of crossings of G is equivalent in terms of propositional calculus to determining a truth assignment which satisfies all clauses in C_{ii} and C_{iii} and which maximizes the number of satisfied clauses in C_i. This optimization problem is known as PARTIAL MAX-SAT [28], a variant of the MAX-SAT problem, and remains NP-hard even for (unsatisfiable) 2-CNF instances. Unfortunately, it turns out that for considering crossing minimization in terms of PARTIAL MAX-SAT, formula C_{iii} cannot be dropped in general [24].

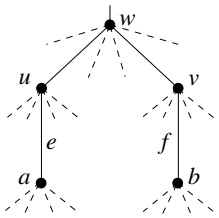

Figure 3: Part of subtree $T_i(w)$ with two non-crossing edges e and f.

Proposition 2 ([17]). *Let G be a level graph and $t : V(C_G) \to \{0,1\}$ be a truth assignment satisfying all clauses of C_{ii} and C_{iii} and minimizing the number τ_G of violated clauses in C_i. Then τ_G is the minimum number of arc crossings in a level embedding of G.*

Consider now some tree T_i from F built on a level i from L. Without loss of generality assume that T_i is a complete, k-ary tree of height d, for some integers $k, d > 1$. Note that for $d = 1$ the edges of T_i never cross in any drawing of T_i and the generation of a CNF formula C_{T_i} for T_i can be omitted. Let w be some node from $V(T_i)$ such that the height of subtree $T_i(w)$ is at least 2. Note that the edges of $T_i(w)$ connecting nodes of depth 0 and 1 never cross in any drawing of $T_i(w)$. Therefore, let $e = \{u, a\}$ and $f = \{v, b\}$ be two edges from $E(T_i(w))$ with $u \neq v$ having both depth 1 and $a \neq b$ being some children of u and v, respectively, as shown in Figure 3. In a drawing of $T_i(w)$, e and f do not cross iff

$$u <_\sigma v \quad \Leftrightarrow \quad a <_\sigma b$$

for some linear order σ on the leaves of T_i.

We describe now the construction of a Boolean formula C_{T_i} encoding the plane embedding of T_i. We proceed as follows:

1. For each level $j \in \{1, ..., d\}$ of T_i and every pair $\{u, v\}$ of distinct vertices from level j, create a Boolean variable uv that is true iff $u <_\sigma v$ for some linear order σ.
2. Create the following Boolean subformulas:
   (iv) For each level $j \in \{1, ..., d-1\}$ of T_i and every two edges $e = \{u, a\}$ and $f = \{v, b\}$ from $E(T_i)$ such that $u \neq v$ have depth j and a and b have depth $j + 1$ in T_i, form the non-crossing preserving expression:

$$(uv \to ab) \wedge (vu \to ba)$$

   (v) For each level $j \in \{1, ..., d\}$ and each pair $\{u, v\}$ of distinct vertices of depth j in T_i, form the antisymmetry expression:

$$uv \leftrightarrow \overline{vu}$$

Notice that the subformulas resulting from (iv) and (v) yield after some elementary transformations 2-CNF formulas $C_{iv}^{T_i}$ and $C_v^{T_i}$, respectively, for each tree

T_i. We proceed with the generation of Boolean formulas $C_{T_i} = C_{iv}^{T_i} \cup C_v^{T_i}$ for all trees from F and obtain finally a Boolean formula

$$C_F = \bigcup_{T_i \in F} C_{T_i}$$

encoding the plane embedding of F.

We shall now estimate the length of each formula C_{T_i}. The number of variables generated for each level $j \in \{1, ..., d\}$ of a k-ary tree T_i is equal to $\binom{k^j}{2}$ and thus bounded by $O(k^{2j})$. If $r_i \leq n$ is the number of vertices in level $i \in L$ of graph G, then the height of any k-ary complete tree T_i is at most $\lceil \log_k(r_i) \rceil$. Hence, each C_{T_i} has $O\big(\frac{r_i^2-1}{k^2-1}\big)$ variables. Furthermore, the number of 2-clauses contributed to formula $C_{iv}^{T_i}$ by a level $j \in \{1, ..., \lceil \log_k(r_i) \rceil - 1\}$ of T_i is at most $2k^2\binom{k^j}{2} \in O(k^{2+2j})$, what summed up over $\lceil \log_k(r_i) \rceil - 1$ tree levels yields $|C_{iv}^{T_i}| \in O\big(\frac{r_i^2-k^2}{k^2-1}\big)$. For the number of clauses in $C_v^{T_i}$ we proceed similar as for the number of variables above and obtain that $|C_v^{T_i}| \in O\big(\frac{r_i^2-1}{k^2-1}\big)$. Thus, the number of 2-clauses in C_{T_i} is bounded by $O(r_i^2)$ for some fixed integer $k > 1$. Notice that in case of a tree T_i with r_i leaves but of height greater than $\lceil \log_k(r_i) \rceil$, there must be an inner node in $V(T_i)$ with less than k children. That yields formulas $C_{iv}^{T_i}$ and $C_v^{T_i}$ with less variables and clauses than for the case of the k-ary complete tree with r_i leaves. Similar to Proposition 1, we obtain finally the following result for T_i:

Proposition 3. *For some fixed integer $k > 1$, a k-ary tree T_i built on a level i with r_i vertices has a planar embedding iff C_{T_i} is satisfiable. The test can be done in time $O(r_i^2)$.*

Since r_i is the number of vertices on level $i \in L$ in graph G and $r_1 + ... + r_{|L|} = n$, it follows that $|V(C_F)| \in O(n^2)$ and $|C_F| \in O(n^2)$.

Corollary 1. *For some fixed integer $k > 1$, a set of k-ary trees built on a level graph G with n vertices has a planar embedding iff C_F is satisfiable. The test can be done in time $O(n^2)$.*

Note that every satisfying truth assignment for C_F induces compatible linear orders σ_i on the leaves of each $T_i \in F$, and vice versa.

We are now ready to give a final satisfiability-based formulation for an instance (G, λ, L, F) of a generalized tanglegram on a level graph G. To this end, we simply generate CNF formulas C_G and C_F for (G, λ, L) and F, respectively, as described before, and combine them into a new CNF formula as follows

$$C_{GF} = C_G \cup C_F = (C_i \cup C_{ii} \cup C_{iii}) \cup \bigcup_{T_i \in F} \big(C_{iv}^{T_i} \cup C_v^{T_i}\big).$$

Observe that even if each T_i is planar embeddable (i.e., C_{T_i} is satisfiable) and a level graph G has a planar embedding (i.e., $C_G - C_{iii}$ is satisfiable), too, it does not imply that G plus all the T_i's together is planar embeddable. As mentioned

in the introduction, in our setting we test the existence of a planar embedding of (G, λ, L, F) on at least two planes, i.e., on one horizontal plane for the level graph G and on $|L|$ vertical planes, one for each tree T_i from F.

For a level graph G with n vertices and k-ary trees F defined on its levels L, the number of clauses in C_{GF} is bounded by $O(n^3 + |E(G)|^2)$, according to the discussion above. Furthermore, C_{GF} has $O(n^2)$ variables. Note that these estimates hold only for some fixed integer $k > 1$.

Since C_{GF} contains 3-clauses, it cannot in general be solved for SAT efficiently. However, since the transitivity formula $C_{iii} \in$ 3-CNF is superfluous for the planarity test, we can remove it from C_{GF}, thus obtaining a 2-CNF formula. Similarly as for Proposition 1, we can now solve the planarity test for (G, λ, L, F) in time $O(n^2)$ by applying the algorithm of Aspvall et al. [27]. Recall that the maximal number of arcs in a level-planar graph containing $n > 2$ nodes is at most $2n - 4$. Hence, the number of clauses $|C_{GF} - C_{iii}| \in O(n^2)$.

Proposition 4. *Let (G, λ, L, F) be an instance of a generalized tanglegram on a level graph G with n vertices and k-ary trees F, for some fixed integer $k > 1$. Then (G, λ, L, F) has a planar embedding iff $C_{GF} - C_{iii}$ is satisfiable. The test can be done in time $O(n^2)$.*

Minimizing the number of crossings of (G, λ, L, F) is equivalent to determining a truth assignment which satisfies all clauses in $C_{GF} - C_i$ and which maximizes the number of satisfied clauses in C_i, thus solving an instance of the PARTIAL MAX-SAT problem. Again, for considering crossing minimization in terms of PARTIAL MAX-SAT, formula $C_{iii} \in$ 3-CNF cannot be dropped.

Proposition 5. *Let (G, λ, L, F) be an instance of a generalized tanglegram on a level graph G with n vertices and k-ary trees F, for some fixed integer $k > 1$, and let $t : V(C_{GF}) \to \{0, 1\}$ be a truth assignment satisfying all clauses of $C_{GF} - C_i$ and minimizing the number τ of violated clauses in C_i. Then τ is the minimum number of arc crossings in an embedding of (G, λ, L, F).*

Observe that compatible linear orders σ_i for each level $i \in L$ can be extracted from a truth assignment t in time $O(n^2)$ by traversing all variables of C_{GF}.

4. Experimental results

In the following we discuss the performance and the practical utility of our satisfiability-based technique introduced in this work. To this end we present an extensive experimental comparison of our technique with several known heuristics and some ILP-based approach for the *generalized binary tanglegram* (GBT) problem introduced by Bansal et al. in [4]. Interestingly, this problem is a special case of the generalized k-ary tanglegram problem on level graphs where the number of levels is limited to 2 (i.e., $|F| = 2$) and only (not necessarily complete) binary trees are allowed (i.e., $k = 2$).

The primary goal of our experiments is to evaluate the performance of our method against some exact, ILP-based algorithm as well as against three heuristic approaches for GBT presented by Bansal et al. in [4]. According to their

evaluation, *Alternating Heuristic* (AH), *Local-Search Heuristic* (LH), and *Local-Search Alternating Heuristic* (LAH) possess very good average performance ratio and produce optimized layouts almost instantaneously even for large input instances. All three heuristics utilize fast polynomial-time exact algorithms for a restricted version of the generalized binary tanglegram problem in which the layout of one of the input trees is fixed. Their worst-case running times were already given in Section 1. Furthermore, all heuristics are guaranteed to terminate within polynomial time in the problem size and the number of iterations (AH, LAH) respectively local search steps (LH, LAH) performed, however without any guarantee for the optimality of the solution. Recall that the GBT problem itself is NP-complete. We refer the reader to [4] for a detailed description of those algorithms and heuristics.

Thus similar to the evaluations given in [4, 8], our most important criterion is the performance ratio with respect to the optimal solution in terms of the number of crossings. More specifically, for each input instance i we compute the corresponding performance ratio $\rho_i := (\tau_i + 1)/(\tau_i^* + 1)$, where τ_i denotes the crossing number obtained with the technique being tested and τ_i^* the number of crossings in an optimum layout for the input instance i. Note that we add one to both the numerator and denominator such that the ratio is defined also for crossing-free instances. Nöllenburg et al. [8] gave a simple formulation of a binary tanglegram problem as an ILP. Their ILP-based approach extends easily to exactly solve the GBT problem as well. For our evaluation, we have implemented an exact method, ILPTG, based on this approach in order to obtain crossing numbers τ_i^* of optimum layouts. Here, for solving the ILP we use the C++ API of the commercial mathematical programming software CPLEX 12.1 [23]. Furthermore, we use ILPTG (started with some positive value of the timeout parameter) to compute approximate solutions to the GBT problem, which we afterwards compare with the results of the other methods.

The number of crossings is the main criterion for assessing the quality of new algorithms for generalized tanglegrams problems. However, their computation time, our second criterion, is also an important evaluation aspect, in particular when the embeddings are to be produced interactively in some tanglegram visualization tool [29].

In order to make the comparison of our technique with the heuristics mentioned above more accurate, we have decided to use the same Python implementations of AH, LH, and LAH as the ones being tested by Bansal et al. in [4]. For the evaluation of our method, we have developed a second exact method, called in the following PMSTG, by implementing the ideas introduced in Section 3. Since in our approach, the crossing minimization problem reduces to solving some instance of the PARTAL MAX-SAT problem, PMSTG uses free PARTIAL MAX-SAT solvers from the Sixth Max-Sat Evaluation [30] in order to obtain exact but also approximate solutions for the crossing minimization problem. For our evaluation we use the following PARTAL MAX-SAT solvers:

- akmaxsat and akmaxsat_ls by Adrian Kügel, Ulm University, Germany;
- clasp 2.0.2 by Benjamin Kaufmann, Potsdam University, Germany;

- IncWMaxSat by Han Lin, Kaile Su, Chu Min Li, and Josep Argelich, Sun Yat-sen University, China, and Universitat de Lleida, Spain;
- QMaxSat0.4 and QMaxSat0.11 by Miyuki Koshimura, Xuanye An, Hiroshi Fujita, and Ryuzo Hasegawa, Kyushu University, Japan;
- SAT4J MaxSAT by Daniel Le Berre, Université d'Artois, France.

The automatic generation of the 3-CNF encodings for the input instances has been implemented in Java 1.6, and constitutes an important part of the PMSTG method. Finally, all experiments we performed on an Intel Core i7 3.2 GHz system with 6 GB RAM under Linux version 2.6.32 64 bit.

Test data. We introduce now the test instances which were used for our experimentation. Again, to make our evaluation better comparable with previous evaluation results for GBT and thus much more relevant, we tested PMSTG, ILPTG, as well as the three heuristics with the same test data of three categories as described in [4], i.e., on randomly generated input instances, on simulated gene trees/species trees, and on a real-world gene tree/species tree dataset. According to the description in [4], the random input instances were generated as follows: first they generated uniformly at random two binary trees, T_S and T_G, both with n leaves and established a random one-to-one correspondence between the leaf sets of the two trees. Then they created from those two trees an instance (T_S, T_G) of the GBT problem by adding an additional $\lfloor 15n/100 \rfloor$ uniformly at random selected inter-tree edges. We performed similar experiments for n ranging from 10 to 400 with 10 different instances for each n value.

The simulated gene trees/species trees were created by using a simplified birth-death process that mimics gene duplication and gene loss [12, 31]. To build those trees, they first generated uniformly at random a species tree T_S with n leaves. Then, they produced a simulated gene tree T_G based on T_S according to the following probabilistic procedure: At each internal node v of T_S, the subtree $T_S(v)$ either duplicates with probability d, is lost (removed) with probability r, or remains intact with probability $1-d-r$. For their experiments they chose d and r to be 0.1 and 0.12, respectively. An instance generated by this procedure is given by a pair (T_S, T_G) which set of inter-tree edges includes all edges joining leaves of T_S with leaves of T_G having the same labels. Trees of this category are of practical interest since real-world tanglegrams often consist of two related and rather similar trees. In our work we performed experiments with simulated trees for the same values of d and r, and n ranging from 10 to 400 with 10 different instances for each n. Interestingly, similar techniques for generating gene/species tree pairs resembling real ones, however, for binary tanglegram problems, can be found in [8].

Finally, our real-world input instances comprise an empirical dataset on Angiosperms [32]. It contains 1301 gene trees T_G with number of leaves ranging from 4 to 94 and one species tree T_S with 7 leaves representing different taxa. Thus there are 1301 phylogenetic tree pairs (T_S, T_G) in our real-world input. An empirical dataset of similar size has also been used by Nöllenburg et al. in [8].

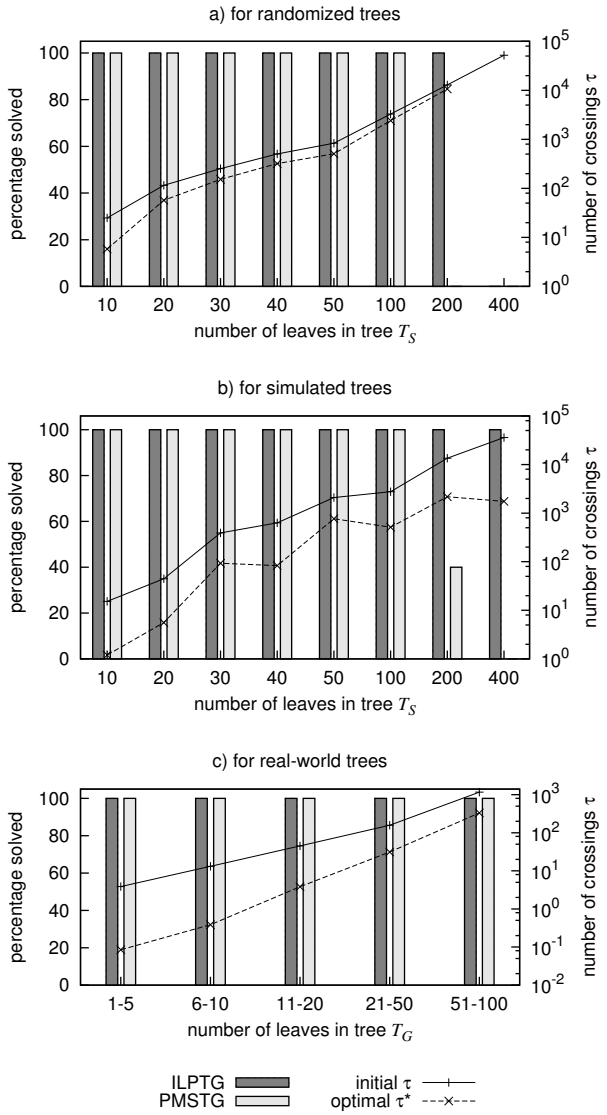

Figure 4: Percentage of optimally solved instances by the exact methods ILPTG and PMSTG (left axes), and average numbers of crossings of initial and optimal layouts (right axes) for random, simulated, and real-world generalized binary tanglegrams. Note the different scales on the axes.

Computation of optimal layouts. In Figure 4 and 5 we present some statistics collected during the experimentation with optimal solutions to the input GBTs. The initial as well as the optimum crossing numbers for all categories of the input instances are depicted in Figure 4a, 4b, and 4c, respectively. Figure 4a

indicates a polynomial growth of the average crossing numbers. Moreover, it could be observed that the average ratio between the crossing numbers of initial and optimal layouts is much smaller for random instances (about 2.1) than for the simulated and real-world instances (about 31.7 and 10.8, respectively). Furthermore, note that for the random instances the gap between the initial and optimal crossing numbers decreases with the growing number of leaves in tree T_S. This cannot be observed for other categories of input instances.

Figure 4 presents also the percentage of optimally solved instances by the exact methods ILPTG and PMSTG. According to Figure 4a and 4b for instances with at least 200 leaves in the species trees T_S, ILPTG performed more robustly than PMSTG applying diverse PARTAL MAX-SAT solvers. This can be explained partly by the fact that CPLEX used by ILPTG took advantage of the parallel architecture of our test system, in contrast to the sequential implementations of the PARTAL MAX-SAT solvers which did not. However, for test instances with up to 100 leaves in trees T_S the average computation times of ILPTG and PMSTG were comparable (see further Figure 7). For test instances containing at least 200 nodes in tree T_S both exact methods appeared to be very time and resource consuming, whereby the random instances turned to be harder to solve than the simulated ones. On the other hand, the real-world instances due to their relatively small sizes could be solved to optimality very fast by both exact solvers (see Figure 7c).

Finally, in Figure 5 the average numbers of Boolean variables and clauses needed for the SAT encodings of the random, simulated, and real-world test instances, respectively, are depicted. The polynomial growth of the number of variables and clauses, proved theoretically in Section 3, can be best observed in Figure 5a. For simulated and real-world instances this behavior cannot be seen so clear because the trees T_S and T_G of each test instance contain typically different numbers of leaves. The number of variables and clauses needed to encode GBT instances with 400 and more leaves in T_S reached 10^8, what in most cases exceeded the possibilities of the PARTIAL MAX-SAT solvers used and of the test system, constituting the main cause of failure.

Performance. In the subsequent discussion we refer to the performance ratios shown in Figure 6. In our experiments concerning the performance ratio but also the computation time (see next paragraph) there was a timeout after 100 seconds wall clock time for all methods tested. Note that the performance ratios and the computation times summarize regularly terminated runs and those aborted after 100 seconds.

A first inspection of the plots immediately reveals that for random and simulated instances with up to 50 leaves in T_S as well as for all real-world instances there are two clear methods of choice that not only outperform the other heuristics but even achieve average performance ratios (within the computation time far bellow the timeout of 100 seconds) hardly deviating from the optimum: ILPTG and PMSTG. The boxplots in Figure 6 show that the heuristics AH and LH performed worst and spread over a relatively large range of values, in particular for medium-sized simulated GBTs (see Figure 6b and Table 1, too). The

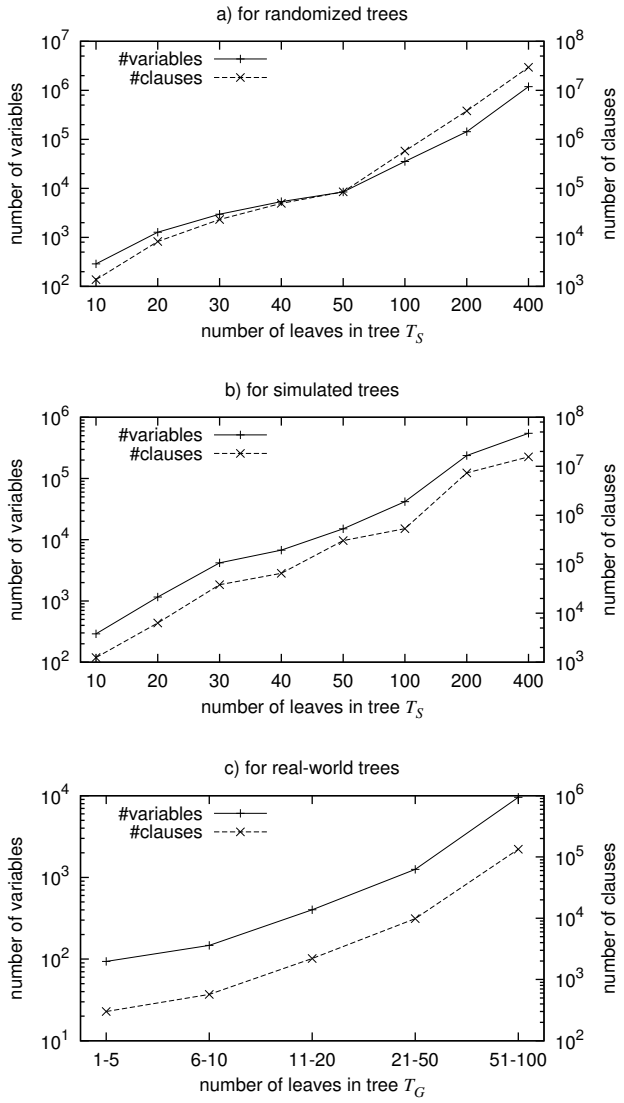

Figure 5: Average numbers of Boolean variables and clauses of the SAT encodings for random, simulated, and real-world generalized binary tanglegrams.

results indicate also that across all instances AH performed much worse than LH. The best performance among the heuristics tested showed LAH, which for random and simulated instances with 100 and more leaves in T_s achieved also the best average performance among all methods tested (see Table 1). Again, comparing the boxplots in Figure 6a and 6b, it is noteworthy that for small- and medium-sized instances ILPTG and PMSTG performed equally well on random

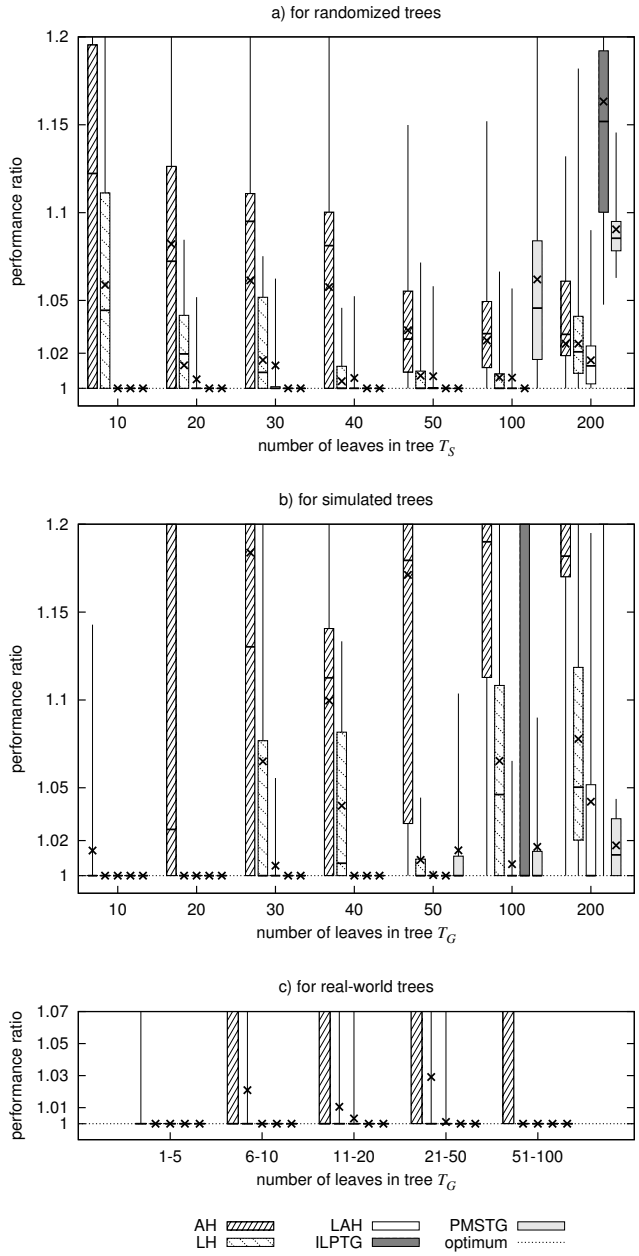

Figure 6: Performance ratios of the five methods AH, LH, LAH, ILPTG, and PMSTG for random, simulated, and real-world generalized binary tanglegrams. The boxplots show median (−), arithmetic mean (×), first and third quartile, minimum, and maximum.

Table 1: Average performance ratios ρ of the five methods AH, LH, LAH, ILPTG, and PMSTG for random, simulated, and real-world generalized binary tanglegrams. Here the actual performance ratios for random and simulated instances were first grouped by the instance size into those having up to 50 leaves in T_S and those with at least 100 leaves, and averaged afterwards. Results marked with an asterisk mean that all test instances of a given category and size were solved to optimality by the corresponding method.

Instance		Method				
Category	#leaves in T_S	AH	LH	LAH	IPLTG	PMSTG
random	≤ 50	1.109	1.020	1.006	1*	1*
random	≥ 100	1.026	1.016	1.011	1.082	1.076
simulated	≤ 50	1.269	1.023	1.001	1*	1,003
simulated	≥ 100	1.265	1.072	1.024	5.533	1.017
real-world	1-100	1.668	1.012	1.001	1*	1*

and simulated instances, whereas the other methods, in particular AH and LH, were susceptible to the similarity and consequently to the crossing number of the two trees (see Figure 4a and 4b, too). This can be regarded as a clear advantage of ILPTG and PMSTG whose performances, according to the results, do not seem to depend on the grade of similarity between T_G and T_S. Recall that in contrast to the random test instances, the trees T_G and T_S of every simulated test instance (T_G, T_S) can considerably differ.

Out of the 70 random input instances for which the optimum solution was known, LAH found an optimum layout in 54 cases, whereas ILPTG and PMSTG in 60 and 52 cases, respectively. Further, out of the 70 simulated test instances solved optimally by some exact method, LAH found an optimum layout in 53 cases, whereas ILPTG and PMSTG in 57 and 58 cases, respectively. Finally, all 1301 real-world test instances were solved optimally by ILPTG and PMSTG. Here, LAH delivered always an optimal layout except for 2 cases.

For input instances with 100 and more leaves in tree T_S the ILPTG method achieved not only very poor performance ratios (see Figure 6b) but also long computation times (see Figure 7c) and thus cannot be regarded as a practical method for large GBTs.

Computation time. The main question here is whether the PMSTG method, whose performance ratio for instances of up to 50 leaves in T_S is far better than the three heuristics and comparable with the exact method ILPTG and which even finds optimal solutions in most of the cases, is fast enough to be used in practice. It is of interest if we can compute optimal solutions for typical input sizes efficiently. Under the assumption that the leaf sets of typical real-world GBTs do not exceed 50 nodes (thus comparable with the sizes of the real-world instances tested here), PMSTG can be regarded as a practical technique for solving the generalized binary tanglegram problem efficiently and, most important, with a guarantee for the optimum.

AH, in spite of its rather poor performance, was able to produce an optimized layout almost instantaneously even for large input instances, outclassing

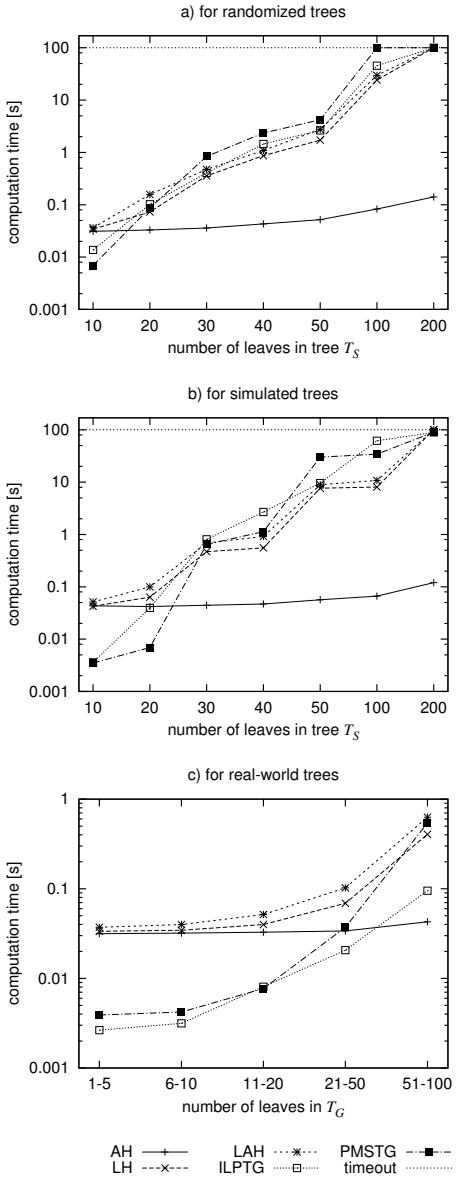

Figure 7: Average computation times of the five methods AH, LH, LAH, ILPTG, and PMSTG for random, simulated, and real-world generalized binary tanglegrams. The computation times of PMSTG include times for the generation of CNF formulas.

other competitors, see Figure 7a and 7b. Observe also that AH was the only method whose computations had not to be interrupted after the elapse of the timeout of 100 seconds. For small-sized random and simulated instances PM-

STG was among the fastest methods with computation times between 0.001 and 0.1 seconds. For medium-sized random and simulated instances the times grew to values between 0.1 and 10 seconds, placing PMSTG in the mid-range of the other methods. For random instances with at least 50 leaves in tree T_S, PMSTG was clearly beaten by other methods, whereas for simulated instances with at least 50 leaves, PMSTG was on average as fast as the second ranked method, ILPTG. Here, the relatively short computation times of ILPTG (especially when compared with the other exact method, PMSTG) can be explained by the fact that in contrast to the other methods, ILPTG using CPLEX took advantage of the parallel architecture of the test system allowing for running up to four parallel processing threads. Moreover, it is remarkable that the computation times of PMSTG do not differ from the computation times of the heuristics LAH and LAH as much as we had expected it. This indicates a clear potential of our method, which with more efficient PARTIAL MAX-SAT solvers can be still improved.

Furthermore, comparing the computation times depicted in Figure 7a and 7b, one can see that they grew much quicker for simulated test instances than for random ones. This can again be explained by the fact that the trees T_G and T_S in the simulated instances are less similar than those of the random instances, resulting in a greater difference between the initial and the optimal crossing numbers (see Figure 4a and 4b). Interestingly, the computation times for random instances with up to 50 leaves in tree T_s show an almost polynomial growth, see Figure 7a. This cannot be observed for other categories of input instances.

According to the results from Figure 7c for small- and medium-sized real-world instances, PMSTG turned to be, together with ILPTG, the fastest method, well ahead of the remaining three heuristics. For large real-world test instances ILPTG is slightly preferable to PMSTG.

Finally, we want to make some remarks regarding the efficiency of the PARTAL MAX-SAT solvers used in the evaluations. During our evaluation we have observed that clasp 2.0.2, IncWMaxSat, and QMaxSat0.4 were the fastest solvers for computing optimal solutions for the small random and simulated instances. For medium-sized test instances of those categories QMaxSat0.4, QMaxSat0.11, akmaxsat, and akmaxsat_ls were the most efficient, whereas large instances with at least 100 leaves in T_S could be solved only with clasp 2.0.2 and SAT4J MaxSAT. For small real-world GBTs QMaxSat0.4 appeared to be the fastest solver, beaten only by clasp 2.0.2 for instances with 50 and more leaves in T_S.

5. Concluding remarks

We have presented a satisfiability-based formulation of the planarity test and the crossing minimization problem on generalized tanglegrams defined on level graphs. Here, the first problem essentially reduces to testing satisfiability of a 2-CNF formula and can be solved in $O(n^2)$ time for instances with n level vertices and k-ary trees defined on each level, for some fixed integer $k > 1$. Moreover, we have shown that the latter problem has a formulation as a PARTIAL MAX-

SAT problem. Here, the question arises whether one could derive bounds on the approximation ratio for generalized tanglegram instances.

The evaluation study has proved that our new method, although designed for a much more general case of generalized k-ary tanglegrams on level graphs, performs also very well on generalized binary tanglegrams, a more specific case of generalized tanglegrams on level graphs. When compared with other techniques, PMSTG emerged as a very competitive technique for computing optimal layouts of small and medium-sized test instances. In this range its performance was at least as good as the performance of the very efficient heuristics LH and LAH of Bansal et al. [4]. For real-world datasets our technique was even able to produce optimal solutions much faster than the three heuristics tested, what clearly qualifies it for use in interactive visualization tools.

From the practical point of view, the advantage of our approach is threefold. First, it can be applied to a much more general case of tanglegram problem, it possesses very competitive performance ratios and computation times, and it does not require the proprietary CPLEX software. Finally, since it is applying PARIAL MAX-SAT solvers to find optimized solutions, any further improvement of those solvers, including parallel ones, will imply a similar improvement in the performance and computation time of our method.

Acknowledgments

The authors would like to thank Mukul S. Bansal, Wen-Chieh Chang, Oliver Eulenstein, and David Fernández-Baca for providing them with an implementation of their methods and realistic instances of generalized binary tanglegrams, and Alexander van der Grinten for implementing the CNF generator for the PMSTG method.

References

[1] E. Speckenmeyer, A. Wotzlaw, S. Porschen, A satisfiability-based approach for embedding generalized tanglegrams on level graphs, in: Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT'11), Vol. 6695 of Lecture Notes in Computer Science, 2011, pp. 134–144.

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

[3] M. A. Charleston, S. L. Parkins, Lizards, malaria, and jungles in the Caribbean, in: Tangled Trees: Phylogeny, Cospeciation and Coevolution, University of Chicago Press, 2003, pp. 65–92.

[4] M. S. Bansal, W. Chang, O. Eulenstein, D. Fernández-Baca, Generalized binary tanglegrams: Algorithms and applications, in: Proceedings of the 1st International Conference on Bioinformatics and Computational Biology, Vol. 5462 of Lecture Notes in Computer Science, 2009, pp. 114–125.

[5] K. Buchin, M. Buchin, J. Byrka, M. Nöllenburg, Y. Okamoto, R. I. Silveira, A. Wolff, Drawing (complete) binary tanglegrams: Hardness, approximation, fixed-parameter tractability, in: Proceedings of the 16th International Symposium on Graph Drawing, Vol. 5417 of Lecture Notes in Computer Science, 2008, pp. 324–335.

[6] B. Venkatachalam, J. Apple, K. St. John, D. Gusfield, Untangling tanglegrams: Comparing trees by their drawings, IEEE/ACM Transactions on Computational Biology and Bioinformatics 7 (4) (2010) 588–597.

[7] G. Di Battista, P. Eades, R. Tamassia, I. G. Tollis, Graph Drawing: Algorithms for Geometric Representations of Graphs, Prentice Hall, 1998.

[8] M. Nöllenburg, D. Holten, M. Völker, A. Wolff, Drawing binary tanglegrams: An experimental evaluation, in: Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments (ALENEX), 2009, pp. 106–119.

[9] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, On the linear-cost subtree-transfer distance between phylogenetic trees, Algorithmica 25 (2-3) (1999) 176–195.

[10] J. Dufayard, L. Duret, S. Penel, M. Gouy, F. Rechenmann, G. Perriere, Tree pattern matching in phylogenetic trees: automatic search for orthologs or paralogs in homologous gene sequence databases, Bioinformatics 21 (2005) 2596–2603.

[11] W. P. Maddison, Gene trees in species trees, Gene Trees in Species Trees 46 (3) (1997) 523–536.

[12] M. Syvanen, Cross-species gene transfer; implications for a new theory of evolution, Journal of Theoretical Biology 112 (1985) 333–343.

[13] M. S. Hafner, P. D. Sudman, F. X. Villablanca, T. A. Spradling, J. W. Demastes, S. A. Nadler, Disparate rates of molecular evolution in cospeciating hosts and parasites, Science 265 (5175) (1994) 1087–1090.

[14] M. R. Garey, D. S. Johnson, Crossing number is NP-complete, SIAM Journal on Algebraic and Discrete Methods 4 (3) (1983) 312–316.

[15] J. Hopcroft, R. E. Tarjan, Efficient planarity testing, J. ACM 21 (4) (1974) 549–568.

[16] S. Leipert, Level planarity testing and embedding in linear time, Ph.D. thesis, Institut für Informatik, Universität zu Köln (1998).

[17] B. Randerath, E. Speckenmeyer, E. Boros, P. L. Hammer, A. Kogan, K. Makino, B. Simeone, O. Cepek, A satisfiability formulation of problems on level graphs, Electronic Notes in Discrete Mathematics 9 (2001) 269–277.

[18] K. Kawarabayashi, B. A. Reed, Computing crossing number in linear time, in: Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC), 2007, pp. 382–390.

[19] H. Fernau, M. Kaufmann, M. Poths, Comparing trees via crossing minimization, Journal of Computer and System Sciences 76 (7) (2010) 593–608.

[20] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1979.

[21] S. Khot, N. K. Vishnoi, The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into l_1, in: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2005, pp. 53–62.

[22] F. Baumann, C. Buchheim, F. Liers, Exact bipartite crossing minimization under tree constraints, in: Proceedings of the 9th International Symposium on Experimental Algorithms (SEA), Vol. 6049 of Lecture Notes in Computer Science, 2010, pp. 118–128.

[23] IBM ILOG CPLEX Optimization Studio (2009).
URL `http://www.ibm.com/software/integration/optimization/cplex-optimization-studio/`

[24] E. Speckenmeyer, S. Porschen, PARTIAL MAX-SAT of level graph (mixed-Horn)formulas, Studies in Logic 3 (3) (2010) 24–43.

[25] P. Eades, N. C. Wormald, Edge crossings in drawings of bipartite graphs, Algorithmica 11 (4) (1994) 379–403.

[26] S. Porschen, E. Speckenmeyer, Satisfiability of mixed Horn formulas, Discrete Applied Mathematics 155 (11) (2007) 1408–1419.

[27] B. Aspvall, M. F. Plass, R. E. Tarjan, A linear-time algorithm for testing the truth of certain quantified Boolean formulas, Information Processing Letters 8 (3) (1979) 121–123.

[28] B. Cha, K. Iwama, Y. Kambayashi, S. Miyazaki, Local search algorithms for partial MAXSAT, in: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI/IAAI), 1997, pp. 263–268.

[29] D. Holten, J. J. Van Wijk, Visual comparison of hierarchically organized data, Computer Graphics Forum 27 (3) (2008) 759–766.

[30] Sixth Max-SAT Evaluation (2011).
URL `http://maxsat.ia.udl.cat/introduction/`

[31] L. Arvestad, A.-C. Berglund, J. Lagergren, B. Sennblad, Gene tree reconstruction and orthology analysis based on an integrated model for duplications and sequence evolution, in: Proceedings of the 8th Annual International Conference on Computational Molecular Biology (RECOMB), 2004, pp. 326–335.

[32] M. J. Sanderson, M. M. McMahon, Inferring angiosperm phylogeny from EST data with widespread gene duplication, BMC Evolutionary Biology 7 (Suppl 1) (2007) S3+.