





Minimum-Width Drawings of Phylogenetic Trees

Juan Jose Besa^(✉) , Michael T. Goodrich , Timothy Johnson,
and Martha C. Osegueda

University of California, Irvine, USA
{jjbesavi,goodrich,tujohnso,mosegued}@uci.edu

Abstract. We show that finding a minimum-width orthogonal upward drawing of a phylogenetic tree is NP-hard for binary trees with unconstrained combinatorial order and provide a linear-time algorithm for ordered trees. We also study several heuristic algorithms for the unconstrained case and show their effectiveness through experimentation.

Keywords: Phylogenetic · Tree drawing · Orthogonal · Upward

1 Introduction

A *phylogenetic tree* is a rooted tree that represents evolutionary relationships among a group of organisms. The branching represents how species are believed to have evolved from common ancestors and the vertical height of an edge represents the genetic difference or time estimates for when the species last diverged. In this paper, we study the algorithmic complexity of producing minimum-width drawings of phylogenetic trees, in particular we describe it as finding the minimum width drawing of upward orthogonal trees of fixed edge-length. Edge-length must equal vertical distance because edges can only extend vertically when the tree is both orthogonal and upward; to allow for more than one outgoing edge, vertices must extend horizontally.¹ See Fig. 1.

Given a phylogenetic tree, T , with a length, L_e , defined for each edge $e \in T$, the *min-width phylogenetic tree* drawing problem is to produce an upward planar orthogonal drawing of T that satisfies each edge-length constraint (so the drawn vertical length of each edge e is L_e)² and minimizes the width of the drawing of T .

The motivation for this problem is to optimize the area of the drawing of a phylogenetic tree, since the height of the drawing is fixed by the sum of lengths of the edges on a longest root-to-leaf path. From an algorithmic complexity perspective this problem is trivial in clock trees for non-extinct species, since all root-to-leaf paths are of the same length and all leaves must therefore be drawn

¹ Alternatively, an upward node-link tree with 1-bend edges and fixed node height.

² W.l.o.g., each node is embedded below its parent; other orientations, such as drawing nodes above their parents, are equivalent to this one via rotation.

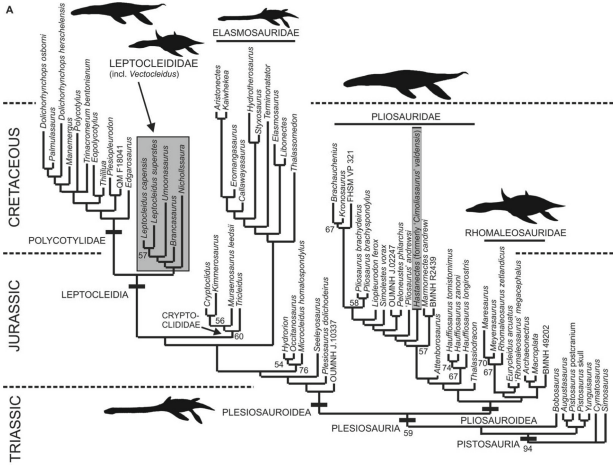


Fig. 1. An orthogonal upward drawing of a phylogenetic tree, from [6].

on the same level. Thus, we are interested in the general case, as shown in Fig. 1, where width improvement is possible by allowing subtrees of extinct species to lay above the branches of surviving species.

In this paper, we show that min-width phylogenetic tree is NP-hard if the order of the leaves can be chosen (i.e. unconstrained) and provide a linear-time algorithm for trees with a fixed leaf ordering. Also, we describe several heuristic algorithms for the former and show their effectiveness by experimentation.

Related Work. There is considerable prior work on methods for producing combinatorial representations of phylogenetic trees, that is, to determine the branching structures and edge lengths for such trees, e.g., see [17, 18, 22, 28, 30]. For this paper, we assume that a phylogenetic tree is given as input.

Existing software systems produce orthogonal drawings of phylogenetic trees, e.g., see [7, 10, 19, 23, 24, 31], but we are not familiar with any previous work on characterizing the algorithmic complexity of the minimum-width orthogonal phylogenetic tree drawing problem. Bachmaier *et al.* [2] present linear-time algorithms producing other types of drawings of ordered phylogenetic trees, including radial and circular drawings, neither of which are orthogonal drawings.

Several researchers have studied area optimization problems for planar upward tree drawing without edge-length constraints, e.g., see [1, 11–15, 20, 26, 27]. In terms of hardness, Biedl and Mondal [5] show it is NP-hard to decide whether a tree has a strictly upward straight-line drawing in a given $W \times H$ grid. Bhatt and Cosmadakis [4] show that it is NP-complete to decide whether a degree-4 tree has a straight-line (non-upward) orthogonal grid drawing where every edge has length 1. Gregori [16] extends their result to binary trees and Brunner and Matzeder [9] extend their result to ternary trees. In addition, Brandes and Pampel [8] show that several order-constrained orthogonal graph

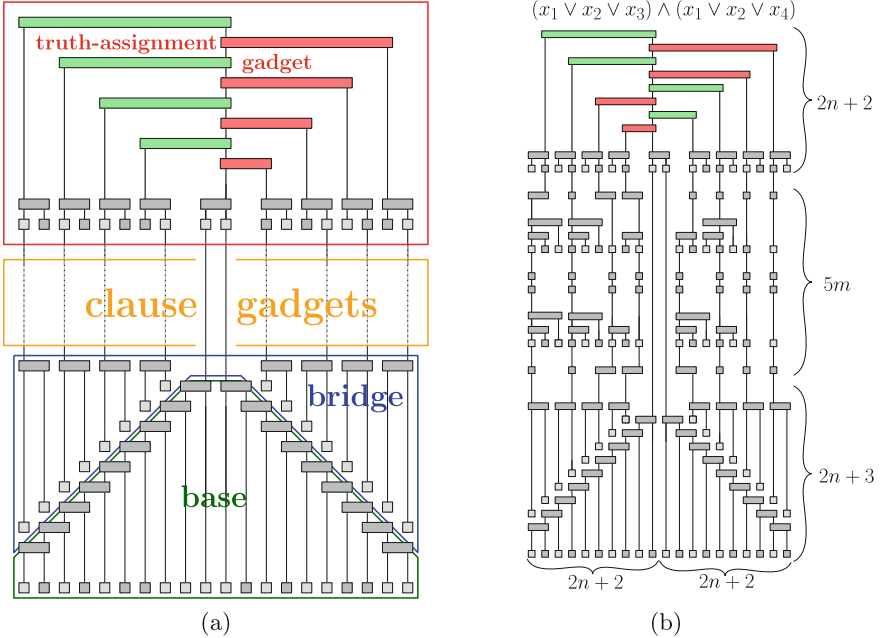


Fig. 2. Overview of Theorem 1 (a) General structure. (b) Drawing of reduction corresponding to a satisfying assignment, $\mathcal{A} = \{true, true, false, false\}$

drawing problems are NP-hard, and Bannister and Eppstein [3] show that compacting certain nonplanar orthogonal drawings is difficult to approximate in polynomial time unless $P = NP$. Previous hardness proofs do not apply to the min-width phylogenetic tree drawing problem, however.

2 Min-Width Phylogenetic Tree Drawing is NP-hard

Theorem 1. *Computing the minimum width required for an upward planar orthogonal drawing of a binary tree with fixed vertical edge lengths is NP-hard.*

We prove this via a reduction from NAE-3SAT, a variant of 3SAT in which an assignment is considered satisfying when each clause’s boolean values are not all the same. An instance, ϕ , of NAE-3SAT is defined by n variables $X = \{x_1, \dots, x_n\}$ and m clauses $C = \{c_1, \dots, c_m\}$. Each clause consists of exactly 3 literals, where a *literal* is a negated or non-negated variable. Given a truth assignment, a literal is considered *satisfied* if the literal evaluates to true. A truth assignment \mathcal{A} satisfies ϕ when each clause contains only one or two satisfied literals. Each clause must therefore also have either one or two unsatisfied literals.

Given a truth assignment, \mathcal{A} , we define $\bar{\mathcal{A}}$ to be the truth assignment where each assigned truth value is the negation of the truth value assigned in \mathcal{A} . If \mathcal{A} satisfies ϕ each clause must contain a satisfied literal, l_1 , and an unsatisfied

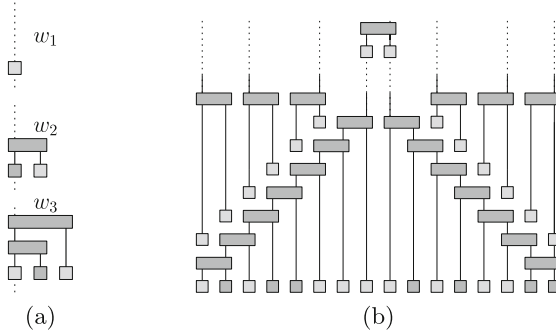


Fig. 3. Construction pieces: (a) w_k substructures (b) Pyramidal embedding of bridge and base for $n = 3$

literal, l_2 , then $\overline{\mathcal{A}}$ must also contain an unsatisfied literal, l_1 , and a satisfied literal, l_2 . Thus for any \mathcal{A} satisfying ϕ , $\overline{\mathcal{A}}$ must also satisfy ϕ .

We use this property to create a combinatorial phylogenetic tree T from an instance ϕ such that T admits an upward orthogonal drawing of width $4n + 4$ if and only if ϕ is satisfiable. Since the vertical length of each edge is fixed, each node in T has a fixed level or height at which it needs to be drawn. We say that two nodes are *horizontally aligned* when they lie at the same level. Our reduction follows the general structure shown in Fig. 2a. The top part of the tree forms a *truth-assignment gadget*, where each variable has two branches, one corresponding to a true value (shown in green) and the other corresponding to a false value (shown in red). The truth-assignment gadget's combinatorial order therefore defines truth assignments for \mathcal{A} and $\overline{\mathcal{A}}$, on the left and right, respectively. Figure 2a illustrates all true and all false truth assignments (respectively) whereas Fig. 2b illustrates a satisfying assignment for a two-clause formula. The middle part comprises a sequence of *clause gadgets*, with 3 rows for each clause gadget, plus rows of *alignment nodes*, separating the clause gadgets. The bottom parts of the tree comprise *bridge* and *base* gadgets, which ensure that the pair of branches corresponding to the true and false values for each variable must be on opposite sides (see Lemma 2).

Let $P(k)$ be a path of k nodes with unit edge-lengths, where p_1 is the root of the path and $p_i \in P$, $2 \leq i \leq k$ is the i -th node along the path. A node p_i may have two children, the node p_{i+1} (if it exists) and some other new node (or two if $i = k$). We define the substructures w_1, w_2 , and w_3 (Fig. 3a), consisting of the minimal tree with 1, 2, and 3 leaves at the same height.

We proceed from top to bottom to describe the structure of T . The tree, T , is rooted on the first node of a path $P(2n)$, called the *variable path*. We first add a w_2 component to p_{2n} a unit-length away as its first child, which we use to root the base. We then add a w_2 component as the second child of each node p_i , $1 \leq i \leq 2n$ so the $2n$ new w_2 components lay in horizontal alignment with the one rooting the base. These $2n$ w_2 components connect each node in the variable

path to a branch. Of the $2n$ branches, there are two for each of the n variables in ϕ . We label the branch originating from p_i with the truth assignment setting $x_{\lfloor(i+1)/2\rfloor}$ to ‘true’ if i is odd and ‘false’ if i is even. These branches are now called *assignment branches*.

Each assignment branch consists of m clause components connected in a path by unit-length edges, where the j -th component (for $1 \leq j \leq m$) belongs to the clause gadget of clause c_j . All clause components belong to the same clause have the same height, so each clause occupies a distinct horizontal band of space in the tree. Each clause component consists of one of three possible w_k components and two surrounding alignment nodes, one above the w_k component and one below. Each w_k component has height 3 and for clause components corresponding to the same clause the leaves of all w_k are horizontally aligned.

Recall that each variable has two assignment branches associated to it, one corresponding to setting the variable to true and one to false. A clause component is defined by both the clause, c_j , and the branch’s labeled truth assignment, $x_i = \{true, false\}$. If x_i does not participate in clause c_j then its clause component has a w_2 substructure. If x_i appears as a literal in c_j and evaluating it with the assigned truth value satisfies the literal then the clause component has a w_3 substructure. However, if the literal is unsatisfied then it contains a w_1 substructure. For example, in Fig. 2b clause $c_1 = (x_1 \vee x_2 \vee x_3)$, the branch labeled $x_1 = true$ contains w_3 , whereas $x_1 = false$ contains w_1 and both x_4 assignment branches contain w_2 .

Top alignment nodes have incoming edges connecting from the bottom alignment nodes of the previous clause. These two consecutive rows of alignment nodes enable the clause gadgets along an assignment branch to be shifted left or right to efficiently align into the available space within that gadget (see Appendix A.1).

After the last clause component in each branch (labeled with x_i) we attach a node one unit away from the last shifter and give it two children, one $2n - 2i + 1$ units away and the other $2n - 2i + 2$. These nodes together form the bridge gadget.

To build the base, we set a path $P(2n + 2)$ as a child $5m + 2$ units away for each of the two leaves in the remaining copy of w_2 attached to p_{2n} . For each non-leaf node in the path we just connected, we set their remaining child to be a single node horizontally aligned with the leaf.

Lemma 1. *At minimal width, the base and bridge can only assume a pyramidal embedding. We define a pyramidal embedding as the embedding of the base in which nodes closer to the root lie closer to the center and nodes further from the root approach the outer sides, as shown in Fig. 3b.*

Proof. This proof is based on the fact that a pyramidal embedding fully occupies every cell, and thus occupies the least area and minimum width. Furthermore, the pyramidal embedding is the only minimum width embedding for these gadgets. We present the complete proof in Appendix A.2.

Lemma 2. *The embedding of the truth-assignment gadget defines assignments A and \bar{A} .*

Proof. As a consequence of Lemma 1, the two edges coming into the base must be centered. We refer to these two edges as ‘the split’. The base takes up width $2n + 1$ on either side of these two edges, and each branch needs width two, so at most n can fit on either side.

Furthermore, in order for the bridge to assume the pyramidal embedding, each leaf on the left side must have a corresponding leaf at the same level on the right side. Note that the height of the leaves in the bridge gadget depends on which variable branch they are attached to, so that leaves on the same level correspond to the two assignments of the same variable. Therefore, there is one assignment branch for each variable on each side, so the assignments labeling the branches on one side must all be of different variables and thus describe a truth assignment. The branches on the opposite side have the opposite assignment for each variable. Let the truth assignment on the left side be \mathcal{A} , and the one on the right side be $\overline{\mathcal{A}}$. ■

Lemma 3. *Given truth assignments \mathcal{A} and $\overline{\mathcal{A}}$, a clause in ϕ is satisfied if and only if the width used by the clause gadget is at most $2n + 1$ on both sides of the split.*

Proof. The maximum number of horizontally aligned nodes on the left side of the clause gadget is equal to the sum of the width of the w_k ’s, $k \in \{1, 2, 3\}$, in each assignment branch for assignment \mathcal{A} . Recall that within a clause gadget each assignment branch has an embedded copy of either w_1 , w_2 or w_3 . We define function $p_{i,j}$, which describes the width of an embedded copy w_k , and $S_j(\mathcal{A})$, which describes the sum of all the embedded element’s widths, as follows:

$$S_j(\mathcal{A}) = \sum_{i=1}^n p_{i,j}(\mathcal{A}), \text{ where } p_{i,j}(\mathcal{A}) = \begin{cases} 3 & \text{if } \mathcal{A} \text{ does satisfy } x_i \text{'s literal in } c_j \\ 1 & \text{if } \mathcal{A} \text{ doesn't satisfy } x_i \text{'s literal in } c_j \\ 2 & \text{if } c_j \text{ has no literal of } x_i \end{cases}$$

By definition, a clause can only be satisfied if one or two of the clause’s literals are satisfied and the remaining one or two literals must be unsatisfied. W.l.o.g. we can assume that each clause consists of two or three literals from distinct variables.³ For clauses with literals from three distinct variables, any clause c_j satisfied by a truth assignment \mathcal{A} must have only one or two satisfied literals. If \mathcal{A} satisfies only one literal in clause, c_j , then $S_j(\mathcal{A})$ evaluates to $3 + 1 + 1 + 2(n - 3) = 2n - 1$, since only 3 of the n variables participate in a clause. If it satisfies two literals, it evaluates to $3 + 3 + 1 + 2(n - 3) = 2n + 1$ instead. If \mathcal{A} satisfies c_j , then $\overline{\mathcal{A}}$ also satisfies c_j implying $S_j(\mathcal{A})$ and $S_j(\overline{\mathcal{A}})$ are both at most $2n + 1$. On the other hand, if \mathcal{A} doesn’t satisfy c_j , then it either satisfies all three literals, and $S_j(\mathcal{A})$ evaluates to $3 + 3 + 3 + 2(n - 3) = 2n + 3$, or $\overline{\mathcal{A}}$ satisfies all three

³ Degenerate cases to consider include cases when a variable contributes multiple literals to a single clause. We can safely ignore cases when all three identical literals are present (which is not satisfiable) and when positive and negated literals of the same variable are present (since the clause is always satisfied). When a literal is repeated exactly twice, we handle it as a clause of only the two distinct literals.

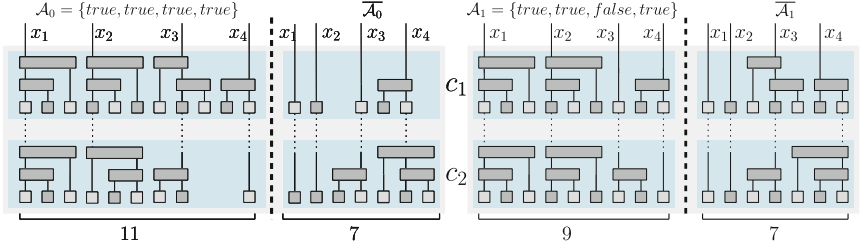


Fig. 4. Original and satisfied clause gadgets for $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_4)$

literals and $S_j(\bar{\mathcal{A}}) = 2n + 3$. Therefore if \mathcal{A} doesn't satisfy c_j , $S_j(\mathcal{A})$ or $S_j(\bar{\mathcal{A}})$ will exceed $2n + 1$.

For clauses with literals from two distinct variables, any \mathcal{A} can only satisfy c_j with one satisfied literal and one unsatisfied literal. $S_j(\mathcal{A})$ and $S_j(\bar{\mathcal{A}})$ both evaluate to $3 + 1 + 2(n - 2) = 2n$, both remaining strictly less than $2n + 1$. However if \mathcal{A} doesn't satisfy c_j then either \mathcal{A} satisfies both literals, and $S_j(\mathcal{A}) = 3 + 3 + 2(n - 2) = 2n + 2$, or $\bar{\mathcal{A}}$ satisfies both literals. Therefore if \mathcal{A} doesn't satisfy c_j , $S_j(\mathcal{A})$ or $S_j(\bar{\mathcal{A}})$ will exceed $2n + 1$.

We have now proved that $S_j(\mathcal{A})$ and $S_j(\bar{\mathcal{A}})$ are both at most $2n + 1$ if and only if \mathcal{A} and $\bar{\mathcal{A}}$ both satisfy c_j . As long as the clause gadget is able to assume a dense embedding, the width necessary on opposite sides should be exactly equal to $S_j(\mathcal{A})$ and $S_j(\bar{\mathcal{A}})$. The alignment nodes in each clause gadget are sufficient to guarantee a dense embedding is possible, which we prove in Appendix A.1. ■

Wrapping up the main proof, if any clause is not satisfied the clause gadget will exceed the allowable space of $2n + 1$ and increase the width to at least $4n + 5$. Therefore only a satisfying assignment \mathcal{A} would retain a width of $4n + 4$, proving that if a satisfying assignment for ϕ exists then there exists an embedding of T with width $4n + 4$ (Fig. 4).

On the other hand, if a tree T has a drawing of width $4n + 4$ then every clause was satisfied (following Lemma 3), and thus \mathcal{A} must describe a satisfying assignment for ϕ . This proves that T can be embedded with width $4n + 4$ if and only if ϕ is satisfiable.

Furthermore, our reduction features a multi-linear number of nodes: the variable gadget has $8n + 3$ nodes, the clause gadgets exactly $5mn$ nodes, the bridge gadget $6n$ and the base gadget $8n + 6$ totaling exactly $22n + 5mn + 12$ nodes. This completes our proof of Theorem 1.

3 Linearity for Fixed-Order Phylogenetic Trees

Theorem 2. *A minimum width upward orthogonal drawing of a fixed-order n -node phylogenetic tree can be computed in $O(n)$ time.*

We provide an algorithm that computes a minimum width drawing. The key idea is to construct a directed acyclic graph (DAG) of the positional constraints

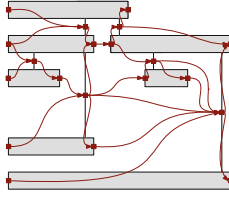


Fig. 5. The constraint graph of Lemma 4

between nodes and edges. The DAG can then be processed efficiently to determine a positioning of each node and edge that ensures the minimum width. Let S be a set of non-intersecting orthogonal objects (e.g., rectangles and segments) in the plane. Two objects s and s' are horizontally visible if there exists a horizontal segment that intersects s and s' but no other object of S . Since the height of each object of our drawing is fully determined by the edge lengths of the tree, determining which objects are horizontally visible is essential to construct a minimum width drawing. For a fixed order combinatorial phylogenetic tree $T = (V, E)$, the *Constraint Graph* $D = (U, A)$ of T is a directed graph with a vertex for each left and right side (of the rectangle representing the node in the drawing) of each node of T and one for each edge of T . (See Fig. 5.) An arc $e = uv \in A$ if the objects corresponding to u and v are horizontally visible and u precedes v as determined by the fixed order.

Lemma 4. *The constraint graph D of the fixed order n -node phylogenetic tree $T = (V, E)$ is a DAG with $3n - 1$ vertices and $O(n)$ edges, where $n = |V|$.*

Proof. The key is that D must be planar, for the full proof see Appendix A.3.

The algorithm has two main steps. First, we construct the constraint graph D , and then we process the constraint graph to find a minimum-width drawing. As we have mentioned, the vertices of D can be constructed directly from the vertices and edges of T . We now show that the arcs in D can be created using a single pre-order (node, then children left to right) traversal of the tree, while growing a frontier indicating the rightmost object seen at each height. We maintain the frontier efficiently as an ordered list of height ranges. Whenever we update the frontier we have found a new rightmost object. If we are not extending the frontier (i.e. adding to the end of the list), then we have covered/partially covered some object. The two objects must be horizontally visible so we add a new directed arc from the left object to the right.

For a linear algorithm, we must avoid searching in the frontier for the position of each object. The key observation is that, while processing a node v of T , the edges and nodes of the subtree rooted at v only affect the frontier below v . In other words, the position of v in the frontier doesn't change while processing its subtree. When a child is completely processed we can find the next sibling's position in the frontier by looking at the position of their parent.

Once the constraint graph is constructed, it must be processed to find the positions at which to draw each object. We process the vertices of D in topological order. Vertices that have no incoming arcs, which are the sources of the constraint graph, must be the left side of vertices of T and can be positioned at x-coordinate 0. At each remaining vertex, we check its incoming arcs and assign it the leftmost position that is to the right of every vertex in its in-neighborhood. Because the arcs represent the necessary order at each height and the sources of the DAG are positioned as far left as possible, a simple inductive argument proves that the resulting drawing has minimum width.

Traversing the tree to construct our DAG requires us to update the frontier once for every arc, source, and sink of the DAG. Each update takes constant time, so by Lemma 4 determining the arcs of the constraint DAG takes a total of $O(n)$ time. The time taken for the processing step includes the topological sort and the time to check each incoming arc at each vertex. Both of these are bounded by the number of arcs, so by Lemma 4 processing the DAG also takes $O(n)$ time. In conclusion both steps take $O(n)$ time so the algorithm takes $O(n)$ in total. This completes the proof of Theorem 2.

4 Heuristics and Experiments

Let T be a combinatorial phylogenetic tree. Once the order of the children of each vertex is determined, we can use Theorem 2 to find a minimum width drawing that respects the edge lengths. Consequently, a heuristic only needs to define the ordering of the children in each vertex. We define the *flip* of a tree (or subtree) rooted at v as the operation of reversing the order of the children of v and every descendant of v . Flipping a tree corresponds to flipping its drawing and does not affect its minimum width.

The *greedy* heuristic proceeds bottom up from the leaves to the tree's root. For each vertex v with children c_0, \dots, c_k this heuristic assumes that the order of the subtrees rooted at its children are fixed and finds the way to arrange its children to minimize width. To do so it considers every possible permutation and combination of flipped children. In general, for a degree d vertex, the greedy heuristic checks $O(d!3^{d-1})$ possible orderings, bounded degree trees therefore take $O(1)$ time per vertex. Because the algorithm calculates the minimum width drawing using the $O(n)$ algorithm from Theorem 2, and runs it $O(1)$ times per vertex, the total running time of the heuristic is $O(n^2)$ for bounded degree trees.

Theorem 3. *The greedy heuristic has an approximation ratio of at least $\Omega(\sqrt{n})$, even for binary phylogenetic trees.*

Proof. Recall the structures for w_k as described in Fig. 3a and consider equivalent structures for larger values of k where all k leaves lie in horizontal alignment. Using this definition of w_k , Fig. 6 shows a tree structure where a minimum width embedding of the subtree in yellow makes it impossible for the entire drawing to admit minimum width. For a minimum width subtree, the greedy heuristic must choose the smallest width possible ($k+2$) thus placing the long edges on opposite

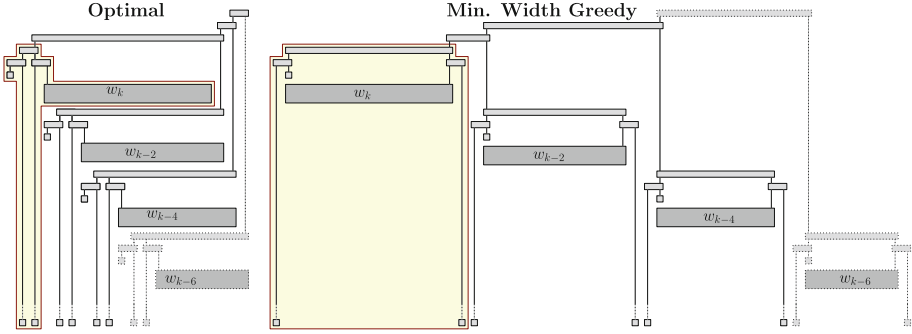


Fig. 6. Tree structures causing worst case performance for the greedy heuristic.

sides. In an optimal ordering the subtree’s embedding would need to be a unit wider ($k + 3$) and place both of long edges adjacent to each other, making the space below w_k available for other subtrees. Label each subtree with the size of the w_k structure inside it, and consider the tree structure with $k/2 - 1$ subtrees $w_k, w_{k-2} \dots w_2$. This structure will have an optimal width of $(k + 3) + (k/2 - 2)$ where the first term accounts for the top-most subtree for w_k (in yellow) and the second from the number of edges connecting to remaining subtrees underneath. The greedy heuristic must instead place each subtree, enclosed by the long edges, side-by-side forcing most leaves into distinct columns. Only one pair of leaves per subtree share their column, therefore the width is equal to the number of leaves $(n + 1)/2$ minus the $k/2 - 1$ overlapping leaves (where n is the total number of nodes). In total, there are $n = \sum_{i=0}^{k/2-1} (7 + 2(k - 2i) + 1) - 1 = k^2/2 + 5k - 1$ nodes, from which we find k . We find that $k \approx \sqrt{2n}$, and therefore the approximation ratio achieved by the greedy heuristic for this tree is $\frac{(n+1)/2 - k/2 - 1}{3k/2 + 1} \approx \frac{n - \sqrt{2n}}{3\sqrt{2n}} = \Theta(\sqrt{n})$, which proves that greedy can have an approximation at least as bad as our tree, thus proving the ratio is at least $\Omega(\sqrt{n})$. ■

Similar to the greedy heuristic, the *minimum area heuristic* proceeds bottom up from leaves to root and finds the best way of arranging its children assuming their sub-trees have a fixed order. While the greedy heuristic minimizes the area of the tree’s bounding rectangle, the minimum area heuristic minimizes the area of the orthogonal y-monotone bounding polygon at the expense of a potential larger total width. The running time is the same as the greedy heuristic and the approximation ratio is also at least $\Omega(\sqrt{n})$ (see Theorem 4 in Appendix A.4).

The *hill climbing* algorithm is a standard black-box optimization approach. Beginning from an initial configuration, it repeatedly tests small changes and keeps them if they do not hurt the quality of the solution. The quality of the solution is exactly equal to the width of the resulting drawing of the tree, and each change tested corresponds to reordering one node’s children.

Our *simulated annealing* algorithm is another black-box optimization approach [21], with the same procedure as hill climbing. The main difference is that changes hurting the solution’s quality are kept with probability inversely related

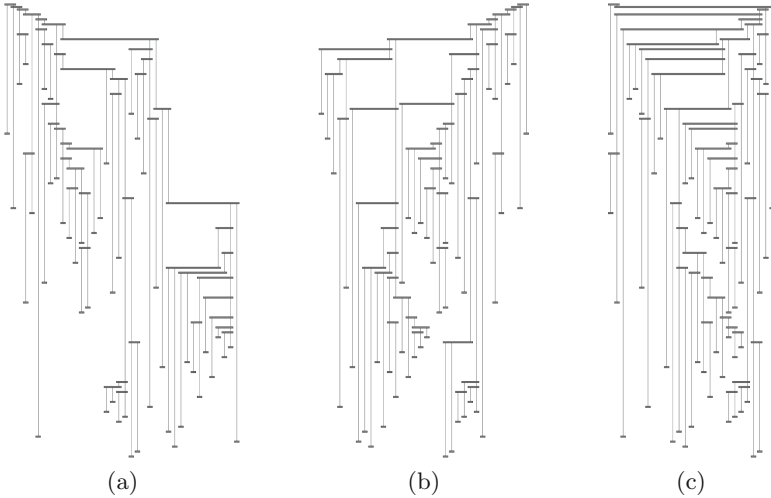


Fig. 7. Drawings of a tree with 93 nodes (a) Input order, width = 37 (b) Greedy order, width = 33 (c) Simulated annealing order, width = 26

to both the difference in quality and the number of steps taken so far. Once the number of steps is large enough, simulated annealing mimics the behavior of the hill climbing algorithm. Compared to hill climbing, simulated annealing has the advantage of not being trapped in a local minimum when a poor starting point was chosen.

Experiments. We evaluate five data sets of real phylogenetic trees obtained from the online phylogenetic tree database TreeBase [25]. The size and compositions of the datasets can be seen in Fig. 8. Each tree in TreeBase originates from a scientific publication, which unfortunately means there are too many for us to list on this paper; we instead provide a complete list of the studies associated with phylogenetic trees used in the data sets and complete experiment source code at github.com/UC-Irvine-Theory/MinWidthPhylogeneticTrees (along with some interesting drawings).

Each dataset is read using Dendropy [29], an open source Python library for phylogenetic computing. Each tree is read with an induced order from the source file, which we will serve as the initial configuration. The datasets are filtered to contain only trees with existing edge-lengths and maximum degree 3. Edge-lengths are normalized into discrete values that preserve nodes' original vertical ordering. For trees with few missing edge-lengths we assume missing edges are of unit length. These normalized datasets are used to evaluate the heuristics and produce easily comparable drawings.

Results. The first thing that stands out from our results is that all of our proposed approaches improve on the original input order. A typical example is shown in Fig. 7, where the input width is improved by the greedy heuristic and

Name	Data Set				Avg. Width Difference from Anneal.				
	#Trees	Smallest	Largest	Average	Input	Greedy	MinArea	Hill	Anneal.
Small	363	85	100	92	+26%	+4%	+13%	±0	±0
Medium	1026	188	399	271	+26%	+4%	+12%	-1%	±0
Large	28	2151	3305	2541	+40%	+5%	+9%	-6%	±0
Plant	80	195	3305	754	+57%	+11%	+19%	-3%	±0
Preferred	175	21	2387	192	+21%	+5%	+10%	-1%	±0

Fig. 8. Results. The left side shows the composition of the data sets, while the right side compares the width obtained versus the simulated annealing.

simulated annealing. Secondly, although the greedy and minimum area heuristics have a bad approximation ratio guarantee, this does not translate to real world trees. As can be seen in Fig. 8, both heuristics performed well for trees regardless of size. For example, in the Preferred data set the greedy heuristic achieved the same width as the Simulated Annealing in 50% of the trees (Fig. 9). However, a few cases exist where the greedy heuristic significantly under-performs. This is notable for two reasons: the first is that the greedy heuristic produces a drawing 60% wider than hill climbing, and the second is that the greedy heuristic is outperformed by the minimum area heuristic.

Finally, it is clear that black-box approaches are useful to find small-width drawings as they rarely produce drawings wider than those from the heuristics. However the width decrease achieved by the black box algorithms comes at a cost in running time, since in our implementation they took around 40 times longer to converge on average.

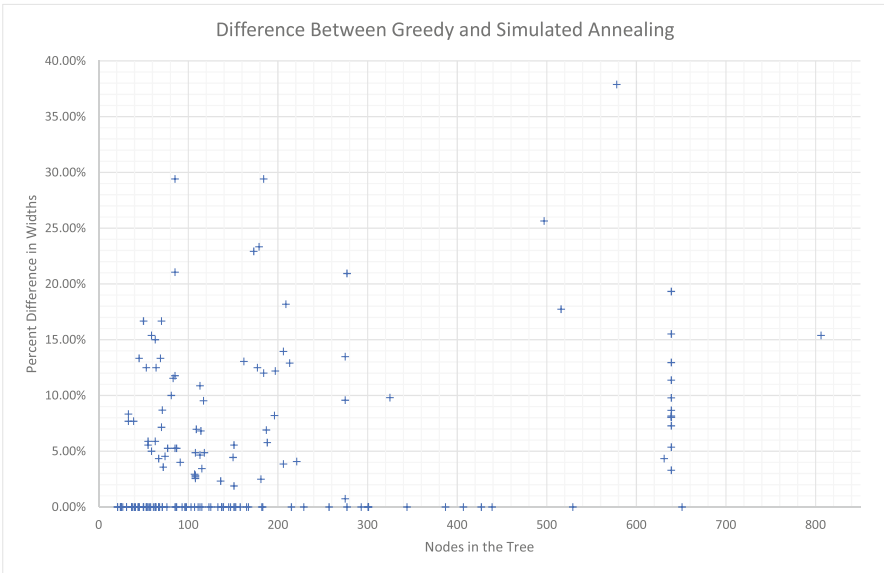


Fig. 9. Percent width difference between greedy and simulated annealing depending on tree size for Preferred dataset.

A Additional Proofs

A.1 Alignment Nodes

Alignment nodes only need to be able to fully realign satisfied clauses, therefore within the three structures at least one must be of width three and at least one of width one. Therefore if we consider the periodicity of the column at which the edge drops, the maximum possible phase difference in these periods is shown in Fig. 10. Considering this order as an extreme case is sufficient because after using both satisfied literal structures (after x_3 in c_1) the only remaining widths could be two (which maintains the phase difference) or one (which reduces the phase difference). The same argument can be made after using both unsatisfied literal structures after x_2 .

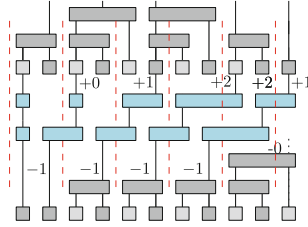


Fig. 10. Set of consecutive clauses, $c_1 = x_2 \vee x_3 \vee x_6$ and $c_2 = x_1 \vee \overline{x_2} \vee x_6$ requiring the largest realignment, with satisfying assignment $x = \{false, true, true, \dots\}$ (Color figure online)

Without alignment nodes it would be impossible to connect x_2 and x_3 to the next clause, but adding the two row of alignment nodes (shown in blue) between clauses enable them to remain connected. This allows each clause to remain tight and assume width of at most $2n + 1$ whenever they are satisfied, regardless of the previous clause.

A.2 Pyramidal Structure

Recall **Lemma 1**: *At minimal width, the base and bridge can only assume a pyramidal embedding. We define a pyramidal embedding as the embedding of the base in which nodes closer to the root lie closer to the center and nodes further from the root approach the outer sides, as shown in Fig. 3b.*

Proof. We define the filled area of a drawing as the sum of the space used by each node (equal to its width), and the space used by each edge (equal to its length). The length of the edges is fixed, but we can change the filled area by changing the layout to minimize the width of the non-leaf nodes. When the base is in the pyramidal embedding, it fills the least possible area, since every non-leaf

node must have width equal to its number of children. We now show that this is the only configuration with minimal area.

We begin from the parents of the base gadget, which belong to a copy of w_2 (Fig. 3a). The two incoming edges from this structure must be next to one another, with no gap between them. The two nodes at the top level of the base each then have both a leaf and a large subtree attached.

The width of each of these top-level nodes must be two, and the only way to achieve this is that each node's leaf must lie on the inside and its subtree on the outside. Similarly, for each subsequent node along the path, the same argument shows that its leaf must lie on the inside. This proves by induction that the base needs to be in a pyramidal embedding. The bridge then must fit against the base. The bridge nodes with the lowest leaves must be on the outside, with the next lowest leaves next to them, and so on by induction back to the center. This shows that the pyramidal embedding is the unique embedding that minimizes the filled area. Since the levels containing the base and bridge nodes are completely packed with no gaps, this also implies that the pyramidal embedding is the unique embedding that minimizes the width of these levels. ■

A.3 Constraint Graph is a DAG

Recall **Lemma 4**: *The Constraint Graph D of the fixed order n -node phylogenetic tree $T = (V, E)$ is a DAG with $3n - 1$ vertices and $O(n)$ edges, where $n = |V|$.*

Proof. Our objects are the left and right sides of each vertex in T , and the edges in T . This gives us two vertices in D for each vertex in T , and one for each edge. Since T is a tree, it must have $n - 1$ edges, so D has $3n - 1$ vertices. If two objects are horizontally visible, then there is a segment between them that crosses only those two objects. We will use these segments to build a planar embedding of D , which will imply that D has $O(n)$ edges.

Let the collection of segments connecting our objects be S . We first construct a larger planar graph D' , in which the vertices are the endpoints of S . The edges of D' include all of the segments in S . We also add edges connecting each vertex to the vertices immediately above and below it that represent the same object. By the definition of horizontally visible, each segment in S can only intersect two objects, so none of the segments in S can intersect. The additional edges also cannot intersect, since they are ordered by the height of the vertices. Therefore, D' is planar.

We then contract all of the edges that connect two vertices in D' corresponding to the same object. This produces the DAG D . Since D is a contraction of a planar graph, it must also be planar. ■

A.4 Approximation Guarantee for Minimum Area Heuristic

We first describe the running time of the heuristic. For each ordering of the children, the minimum width drawing is calculated using the algorithm from Theorem 2 and the bounding polygon is calculated by traversing the tree once

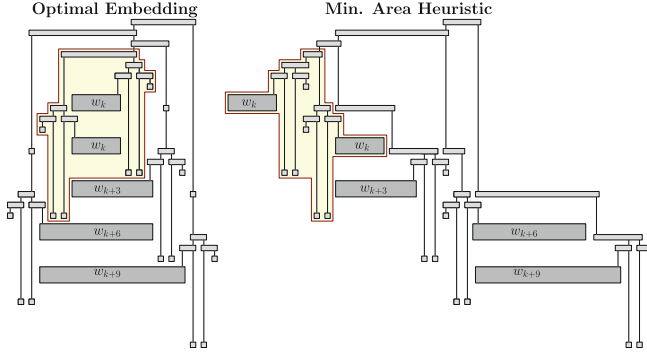


Fig. 11. Tree structures causing worst case performance for minimum area heuristic.

to find the extreme-most branches, running in $O(n)$. We repeat this $O(1)$ times per vertex for a total running time of $O(n^2)$ for bounded degree trees.

Theorem 4. *The minimum area heuristic has an approximation ratio of at least $\Omega(\sqrt{n})$.*

Proof. Recall, the structures for w_k as defined in Theorem 3 and further constrain it to be a complete binary tree with all its k leaves in horizontal alignment. Recall the subtrees used in Theorem 3 and note the subtrees used in this tree instead increase the size of their w_k by 3 each time (with the exception of the first two which have the same w_k). Furthermore each subtrees nodes end immediately before the first node in the subtree two subtrees away, the latter subtree also has a node aligned with the former's w_k leaves. The leaves in w_k are horizontally aligned with the top node in the next subtree.

Using these definitions Fig. 11 demonstrates a tree structure where a minimum area embedding of the two subtrees in yellow makes it impossible for the entire drawing to admit minimum width and minimum area. The heuristic achieves the right embedding for the subtree but fails to choose the right embedding for the two sibling subtrees. Although the optimal's embedding uses a larger area for the combination of both siblings with w_k , it occupies an almost rectangular space resulting in a really small area (and width) increase when adding the next subtree. Define each subtree by the size of the w_k structure inside it, consider the tree with $2k/3$ subtrees $w_k, w_{k+3} \dots w_{3k}$. This structure will have an optimal width of $3k + 6$. The minimum area heuristic would instead have two subtrees with their w_k on opposite sides and the w_{k+3} overlapping the bottom-most w_k and every next pair of subtrees overlapping in the same way. The total width achieved by the minimum area heuristic would therefore be $\sum_{i=0}^{2k/6} (k + 6i + 5) = 2k^2/3 + 11k/3 + 5$. The total number of nodes is $n = \sum_{i=0}^{2k/3} (7 + 2(k - 2i) + 1) + 6 + k = 4k^2/9 + 7k + 14$, which we can use to find k in terms of n . We find that $k \approx \sqrt{n/2}$, and therefore the approximation ratio achieved by the greedy heuristic for this tree is $\frac{2k^2/3 + 11k/3 + 5}{3k + 6} \approx \frac{2k}{9} = \Omega(\sqrt{n})$. ■

References

1. Alam, M.J., Dillencourt, M., Goodrich, M.T.: Capturing Lombardi flow in orthogonal drawings by minimizing the number of segments. In: Hu, Y., Nöllenburg, M. (eds.) GD 2016. LNCS, vol. 9801, pp. 608–610. Springer, Cham (2016). <https://doi.org/10.1007/978-3-319-50106-2>
2. Bachmaier, C., Brandes, U., Schlieper, B.: Drawing phylogenetic trees. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 1110–1121. Springer, Heidelberg (2005). https://doi.org/10.1007/11602613_110
3. Bannister, M.J., Eppstein, D.: Hardness of approximate compaction for nonplanar orthogonal graph drawings. In: van Kreveld, M., Speckmann, B. (eds.) GD 2011. LNCS, vol. 7034, pp. 367–378. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-25878-7_35
4. Bhatt, S.N., Cosmadakis, S.S.: The complexity of minimizing wire lengths in VLSI layouts. *Inf. Process. Lett.* **25**(4), 263–267 (1987)
5. Biedl, T., Mondal, D.: On upward drawings of trees on a given grid. In: Frati, F., Ma, K.-L. (eds.) GD 2017. LNCS, vol. 10692, pp. 318–325. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73915-1_25
6. Benson, R.B., Ketchum, H., Naish, D., Turner, L.E.: A new leptocleidid (sauropterygia, plesiosauria) from the vectis formation (early barremian-early aptian; early cretaceous) of the isle of wight and the evolution of leptocleididae, a controversial clade. *J. Syst. Palaeontol.* **11**, 233–250 (2013)
7. Boc, A., Diallo, A.B., Makarenkov, V.: T-REX: a web server for inferring, validating and visualizing phylogenetic trees and networks. *Nucleic Acids Res.* **40**(W1), W573–W579 (2012)
8. Brandes, U., Pampel, B.: Orthogonal-ordering constraints are tough. *J. Graph Algorithms Appl.* **17**(1), 1–10 (2013)
9. Brunner, W., Matzeder, M.: Drawing ordered $(k - 1)$ -ary trees on k -grids. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 105–116. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18469-7_10
10. Carrizo, S.F.: Phylogenetic trees: an information visualisation perspective. In: Proceedings of the 2nd Conference on Asia-Pacific Bioinformatics, pp. 315–320 (2004)
11. Chan, T.M.: Tree drawings revisited. *Discret. Comput. Geom.* 1–22 (2018)
12. Chan, T.M., Goodrich, M.T., Kosaraju, S.R., Tamassia, R.: Optimizing area and aspect ratio in straight-line orthogonal tree drawings. *Comput. Geom.* **23**(2), 153–162 (2002)
13. Di Battista, G., Didimo, W., Patrignani, M., Pizzonia, M.: Orthogonal and quasi-upward drawings with vertices of prescribed size. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 297–310. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-46648-7_31
14. Frati, F.: Straight-line orthogonal drawings of binary and ternary trees. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 76–87. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77537-9_11
15. Garg, A., Goodrich, M.T., Tamassia, R.: Planar upward tree drawings with optimal area. *Int. J. Comput. Geom. Appl.* **06**(03), 333–356 (1996)
16. Gregori, A.: Unit-length embedding of binary trees on a square grid. *Inf. Process. Lett.* **31**(4), 167–173 (1989)
17. Gusfield, D.: Efficient algorithms for inferring evolutionary trees. *Networks* **21**(1), 19–28 (1991)

18. Huelsenbeck, J.P., Ronquist, F., Nielsen, R., Bollback, J.P.: Bayesian inference of phylogeny and its impact on evolutionary biology. *Science* **294**(5550), 2310–2314 (2001)
19. Huson, D.H., Scornavacca, C.: Dendroscope 3: an interactive tool for rooted phylogenetic trees and networks. *Syst. Biol.* **61**(6), 1061–1067 (2012)
20. Kim, S.K.: Simple algorithms for orthogonal upward drawings of binary and ternary trees. In: Canadian Conference on Computational Geometry (CCCG), pp. 115–120 (1995)
21. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. *J. Chem. Phys.* **21**(6), 1087–1092 (1953)
22. Miller, M.A., Pfeiffer, W., Schwartz, T.: Creating the CIPRES science gateway for inference of large phylogenetic trees. In: Gateway Computing Environments Workshop (GCE), pp. 1–8, November 2010
23. Müller, J., Müller, K.: TREEGRAPH: automated drawing of complex tree figures using an extensible tree description format. *Mol. Ecol. Notes* **4**(4), 786–788 (2004)
24. Page, R.D.: Visualizing phylogenetic trees using treeview. *Curr. Protoc. Bioinform.* (1), 6.2.1–6.2.15 (2003)
25. Piel, W.H., Chan, L., Dominus, M.J., Ruan, J., Vos, R.A., Tannen, V.: Treebase v. 2: a database of phylogenetic knowledge. *e-BioSphere* (2009)
26. Rusu, A., Fabian, A.: A straight-line order-preserving binary tree drawing algorithm with linear area and arbitrary aspect ratio. *Comput. Geom.* **48**(3), 268–294 (2015)
27. Shin, C.S., Kim, S.K., Chwa, K.Y.: Area-efficient algorithms for straight-line tree drawings. *Comput. Geom.* **15**(4), 175–202 (2000)
28. Stamatakis, A., Meier, H., Ludwig, T.: RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics* **21**(4), 456–463 (2004)
29. Sukumaran, J., Holder, M.T.: Dendropy: a python library for phylogenetic computing. *Bioinformatics* **26**(12), 1569–1571 (2010)
30. Warnow, T.: Tree compatibility and inferring evolutionary history. *J. Algorithms* **16**(3), 388–407 (1994)
31. Zainon, W.N.W., Calder, P.: Visualising phylogenetic trees. In: Proceedings of 7th Australasian User Interface Conference, pp. 145–152 (2006)