Characterization of Unlabeled Level Planar Trees[☆]

Alejandro Estrella-Balderrama, J. Joseph Fowler[*], Stephen G. Kobourov

Department of Computer Science, University of Arizona, 1040 E. Fourth St., Tucson, AZ 85721, USA

Abstract

Consider a graph G with vertex set V in which each of the n vertices is assigned a number from the set $\{1, \ldots, k\}$ for some positive integer k. This assignment ϕ is a *labeling* if all k numbers are used. If ϕ does not assign adjacent vertices the same label, then ϕ forms a *leveling* that partitions V into k *levels*. If G has a planar drawing in which the y-coordinate of all vertices match their labels and edges are drawn strictly y-monotone, then G is level planar. In this paper, we consider the class of level trees that are level planar regardless of their labeling. We call such trees *unlabeled level planar* (ULP). Our contributions are three-fold. First, we describe which trees are ULP and provide linear-time level planar drawing algorithms for any labeling. Second, we characterize ULP trees in terms of forbidden subtrees so that any other tree must contain a subtree homeomorphic to one of these. Third, we provide a linear-time recognition algorithm for ULP trees.

1. Introduction

When drawing a planar graph $G(V, E)$ in the xy-plane, a more restrictive form of planarity can be obtained by insisting on a predetermined y-coordinate for each vertex in V in which all the edges in E are drawn straight with a non-zero slope. This is equivalent to placing each vertex on one of k horizontal *tracks*, $\ell_j = \{(x, j) \mid x \in \mathbb{R}\}$ for $j \in [1..k]$, and connecting each pair of adjacent vertices with a line segment. The straight-edge condition can be relaxed to allow edge bends provided the edges remain strictly y-monotone. The vertices are *labeled* by their track number. This labeling ϕ is a *leveling* provided no pair of adjacent vertices are assigned to the same track. The tuple $G(V, E, \phi)$ forms a *level graph*. If a planar drawing of G can be obtained in spite of these restrictions, then G is said to be *level planar*.

Determining whether a given graph G is level planar on k levels can be difficult. The more restrictive problem LEVELED-PLANAR of deciding whether a given directed graph is level planar in which all the edges are directed downwards and are only between vertices of adjacent levels has been shown to be NP-complete [19]. However, if n levels are used, one for each vertex, a labeling in which G is level planar is easily obtained. Any straight-line planar drawing of G can be rotated until the vertices have distinct y-coordinates, the order of which gives the desired labeling.

We call a *level tree* $T(V, E, \phi)$ that is level planar over all possible such labelings ϕ an *unlabeled level planar* (ULP) tree. We characterize ULP trees in terms of forbidden subtrees and provide linear-time recognition and drawing algorithms for any labeling.

1.1. Background and Motivation

Visualizing hierarchical relationships has historically been a strong motivating factor in the study of the planarity of level graphs. Many hierarchical models such as those used in social networks [2] aim to minimize the number of levels while preserving planarity whenever possible. Sugiyama's algorithm [24] does this by favoring the use of shorter edges over longer edges in the display of a *directed acyclic graph* (DAG). Given that, often there are relatively

[☆]This work is supported in part by NSF grants CCF-0545743 and ACR-0222920.

[*]Corresponding author.

Email addresses: aestrell@cs.arizona.edu (Alejandro Estrella-Balderrama), jfowler@cs.arizona.edu (J. Joseph Fowler), kobourov@cs.arizona.edu (Stephen G. Kobourov)

few levels on which one wants to place vertices. As a consequence, researching properties of level graphs with $O(|V|)$ levels has not been actively pursued in the area of graph visualization.

However, *simultaneous geometric embedding*, which is related to geometric thickness [6, 8], has led to a new application of level graphs [3] with one vertex per level. Simultaneous embedding generalizes the notion of planarity when considering multiple planar graphs. When simultaneously embedding a set of planar graphs $\mathcal{G}$, each on n vertices distinctly labeled by the numbers $[1..n]$, all the vertices with the same label must coincide. While any single planar graph can be drawn using only straight-line edges [11], maintaining planarity for each graph without edge bends may not always be possible. Determining the pairs of graphs for which this can be done is NP-hard [10]. For instance, while it is known that two trees cannot always be drawn simultaneously without crossings or edge bends [22], it is unknown which trees always share a simultaneous geometric embedding with any path.

When simultaneously embedding a path with a tree, one approach is to attempt to draw the path monotonically. This gives a labeling in which the vertices are numbered sequentially according to the order they occur along the path. If the tree is level planar for this labeling, then Eades *et al.* [7] show that any such level planar drawing with bends can be redrawn in $O(|V|)$ time without bends. This allows a simultaneous geometric embedding with a path in which the path zig-zags downward through all vertices of the tree. This has the consequence that the set of ULP trees with one vertex per level is precisely the set of trees that have a simultaneous geometric embedding with every monotone path.

1.2. Previous Work

Jünger *et al.* [20] provide linear-time recognition and embedding algorithms for level planar graphs. Here the embedding is the left-to-right ordering of edge intersections with each track. This corrects a PQ-tree algorithm to test level planarity by Heath and Pemmaraju [17, 18]. Di Battista and Nardelli [4] gave the first PQ-tree test for *hierarchies*—level graphs in which there exists a y-monotone path to each vertex from a source vertex on the uppermost track. Eades *et al.* [7] show how to obtain a straight-line level planar drawing in $O(|V|)$ time given a level planar embedding, though it may require exponential area. If the number of levels is constant, then Dujmović *et al.* [5] provide a linear-time level planarity testing algorithm using fixed parameter tractability. Healy and Kuusik [15] give $O(|V|^2)$ recognition and $O(|V|^4)$ embedding algorithms for *proper level planar graphs* (in which all edges are between adjacent levels) using vertex exchange graphs. Harrigan and Healy [14] improve the embedding algorithm to $O(|V|^2)$ time making this a practical alternative to graph-drawing algorithms using PQ-trees that are difficult to implement and have been shown to be error-prone [21].

Further, Di Battista and Nardelli provide a set of *level non-planar* (LNP) patterns [4] that fully characterize level planar hierarchies. However, the level non-planar subgraphs these patterns match are not necessarily edge minimal. Healy *et al.* [16] extend the LNP patterns for hierarchies to provide a set of *minimum level non-planar* (MLNP) patterns in order to characterize all level planar graphs. These subgraph patterns are analogous to Kuratowski's result that any minimal non-planar graph is either a subdivided K_5 or $K_{3,3}$ [23]. However, these patterns are specific to a given labeling and are not based solely upon the underlying graph. This is unlike the ULP characterization for trees that is independent of any labeling and only relies on the structure of the tree in question. The set of MLNP patterns have been shown to be incomplete. Two new MLNP tree patterns were given in [13] based upon T_9, a forbidden tree for the set of ULP trees; see Fig. 1. This has reopened the problem of determining all such MLNP patterns.

1.3. Our Contribution

We characterize ULP trees first for the case of one vertex per level and then for the case of more vertices than levels. Our contributions are three-fold.

1. First, we describe the set of unlabeled level planar (ULP) trees as either (i) a *caterpillar* (a tree in which the removal of all the leaf vertices yields a path), (ii) a *radius*-2 *star* (any number of paths of length 1 or 2 with a common endpoint), or (iii) a *degree*-3 *spider* (three paths with a common endpoint); see Fig. 1. We note that (ii) and (iii) are only ULP with one vertex per level. For each ULP tree, we provide $O(|V|)$-time level planar drawing algorithms on integer grids for any labeling.
2. Second, we characterize ULP trees with one vertex per level in terms of two minimal forbidden trees, T_8 and T_9; see Fig. 1. If multiple vertices per level are permitted, the forbidden tree T_7 characterizes ULP trees.
3. We also provide a $O(|V|)$-time recognition algorithm for ULP trees. If a tree is not ULP, we search for a subtree homeomorphic to one of the forbidden trees, which serves as a certificate for the tree not being ULP.

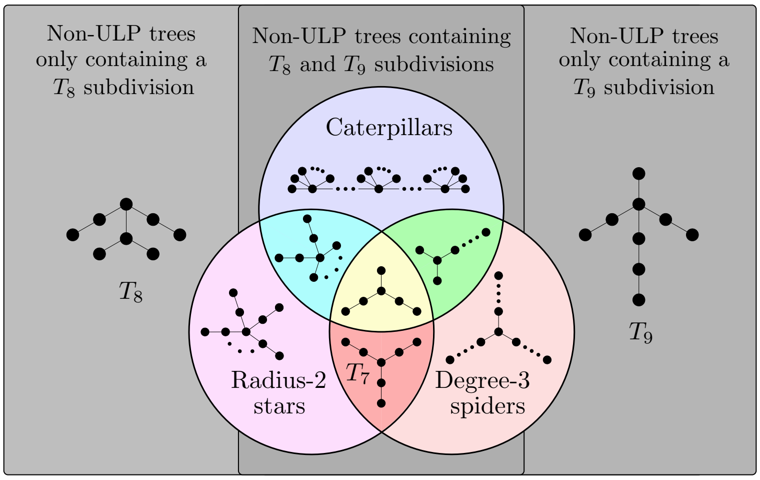


Figure 1: A Venn diagram of the universe of trees partitioned into the trees containing a subdivision of T_8 and/or T_9 (the gray rectangles minus the circles) and the trees not containing either T_8 or T_9, which are caterpillars, radius-2 stars, and degree-3 spiders. These three classes of trees comprise the set of unlabeled level planar (ULP) trees with one vertex per level.

2. Preliminaries

Historically, a level graph is defined as a directed graph with a partitioning of vertices into levels in which the edges are oriented to connect vertices of lower levels to vertices of higher levels. Since we are only concerned with the underlying undirected graph, we define a level graph without edge orientation. A *k-level graph* $G(V, E, \phi)$ on n vertices has a leveling $\phi : V \rightarrow [1..k]$ such that $\phi(u) \neq \phi(v)$ (rather than $\phi(u) < \phi(v)$ in the case of directed graphs) for every edge $(u, v) \in E$.

The leveling ϕ partitions V into k independent sets $V_1, V_2, \ldots, V_k$, which form the *k levels* of G. A *level-j* vertex v is on the j^{th} *level* V_j of G if $\phi(v) = j$ such that $V_j = \phi^{-1}(j)$. If ϕ is an injection, each level contains at most one vertex. If there is one vertex per level, this implies $k = n$ in which case ϕ is a leveling with *distinct labels*. Otherwise, $k < n$ and ϕ has *duplicate labels*.

A level graph G has a *level drawing* if (i) every vertex in V_j can be placed along the *track* $\ell_j = \{(x, j) \mid x \in \mathbb{R}\}$ and (ii) the edges can be drawn as strictly y-monotone connected sequences of line segments. Here the endpoints of each segment lie on distinct tracks so that each edge intersects any given track at most once. The order in which the edges intersect the tracks along the positive x-direction gives a *level embedding* of G. A level graph G is *level planar* if it has a level drawing without edge crossings, which corresponds to a *level planar embedding* of G. A level planar graph G is *realize*d with a level planar drawing, which forms a *realization* of G. A graph G is *unlabeled level planar* (ULP) if it is level planar over all possible labelings.

A *chain* C of G is a simple path denoted v_1–v_2– $\cdots$ –v_t. The chain u–v represents the edge (u, v). A vertex v of C is *ϕ-minimal* (or *ϕ-maximal*) if it has a minimal (or maximal) label of all the vertices of C. A vertex is *ϕ-extreme* if it is ϕ-minimal or ϕ-maximal.

Subdividing an edge (u, v) in a graph $G(V, E)$ replaces it with edges (u, w) and (w, v) in E by adding vertex w to V. A *subdivision* is the result of subdividing any number of edges. A graph $G(V, E)$ is *isomorphic* to $\tilde{G}(\tilde{V}, \tilde{E})$ if there exists a bijection $f : V \mapsto \tilde{V}$ such that $(u, v) \in E$ if and only if $(f(u), f(v)) \in \tilde{E}$. Graph $G(V, E)$ is *homeomorphic* to graph $\tilde{G}(\tilde{V}, \tilde{E})$ if there is an isomorphism between subdivisions of G and $\tilde{G}$.

The tuple $G(V, E, \phi)$ is *proper* if each edge (u, v) in E is a *short* edge such that $|\phi(u) - \phi(v)| = 1$. Any *improper* level graph can be made proper by subdividing each *long* edge ($|\phi(u) - \phi(v)| > 1$) at the points it crosses each track. These points correspond to edge bends if edges are not drawn straight.

A *caterpillar* is a tree in which the removal of all its leaf vertices yields a path, i.e., its *spine*. A *lobster* is a tree in which the removal of all its leaf vertices yields a caterpillar, but not a path. The *eccentricity* of a vertex v in a tree T is the length of the longest path with v as endpoint. The *radius* of T is the minimum eccentricity of all vertices in T. A *radius*-2 *star* (*degree*-3 *spider*) is a tree of radius 2 (arbitrary radius) in which all vertices are degree 1 or 2 except for the vertex r, the *root*, of degree greater than 2 (of degree equal to 3).

3. ULP Trees with Distinct Labels

We first present the drawing algorithms for ULP trees with one vertex per level and then their forbidden tree characterization.

3.1. Drawing ULP Trees with Distinct Labels

Many of our algorithms take a tree as an input and need to efficiently remove degree-1 vertices. This is nontrivial since each deletion can require linear-time in the worst case if standard adjacency lists are used to represent the tree. The next lemma shows how to remove all leaf vertices of a tree efficiently.

Lemma 1. *All leaves can be removed from an n-vertex tree in $O(n)$ time.*

Proof. Removing a leaf from a tree can be done in $O(1)$ time if T has a special adjacency list representation. For each vertex u in the list of vertex v, we store a pointer to the location of v in the list of u. Additionally, the adjacency lists are doubly-linked to allow for efficient deletion. The following pseudocode uses this representation to run in $O(n)$ time.

Remove–Leaves($T(V, E)$)
▷ T is a tree.
1. For each vertex u with an adjacency list with exactly one vertex v:
2. Delete the list of u retaining the pointer p to v that was in the list.
3. Use p to remove u from the doubly-linked list of v in $O(1)$ time. □

The following lemmas describe which trees are ULP and how to realize them in linear time. Brass *et al.* [3] gave an algorithm that produces a simultaneous geometric embedding of a caterpillar and a path on n vertices on an $n \times 2n$ grid. We give an algorithm for producing a more compact drawing with the next lemma.

Lemma 2. *An n-vertex caterpillar with an m-vertex spine can be realized with straight-line edges in $O(n)$ time on a $2m \times n$ grid for any distinct labeling.*

Proof. The spine v_1–v_2–$\cdots$–v_m is drawn with vertices placed at odd x-coordinates. For each spine vertex v_i, leaf vertices are placed one unit to the right at even x-coordinates. If a leaf would overlap a spine edge, then it would be placed directly above or below v_i instead; see Fig. 2. The following pseudocode takes $O(n)$ time as the location of each vertex is determined in $O(1)$ time.

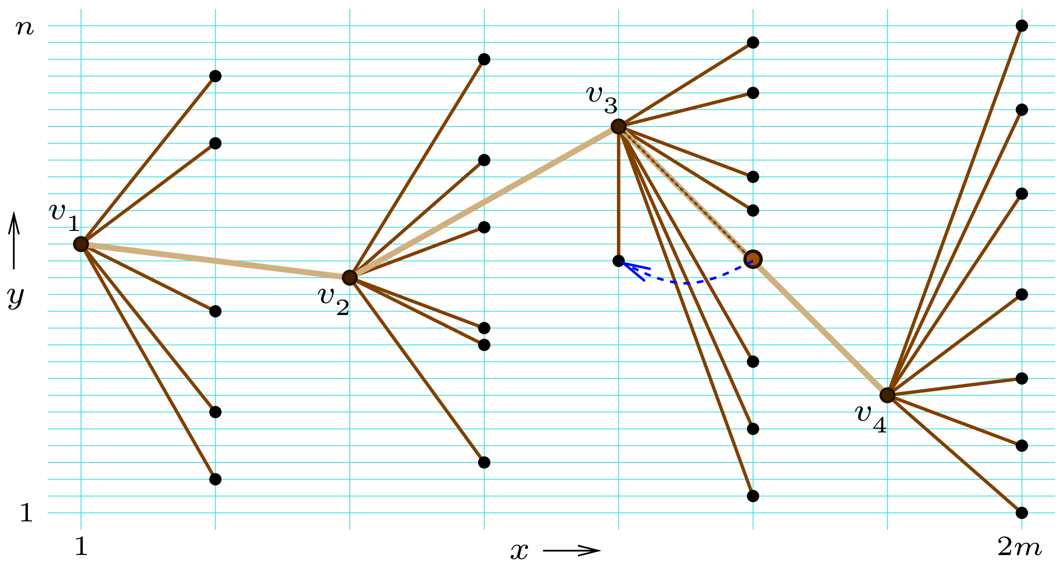


Figure 2: A realization of a caterpillar with distinct labels on a 8×30 grid.

Draw–Caterpillar($T(V, E, \phi)$)

$\triangleright$ T is a caterpillar with distinct labels.

Let S, v_1–v_2–$\cdots$–v_m, be the spine given by Remove–Leaves(T).
Draw spine S by placing v_i at $(2i-1,\ \phi(v_i))$ for $i \in [1..m]$.
Draw edge v_i–v_{i+1} for $i \in [1..(m-1)]$.
For each v_i for $i \in [1..m]$:
    For each leaf ℓ that is adjacent to v_i:
        Unless leaf ℓ would lie on v_i–v_{i+1}, place ℓ right of v_i at $(2i,\ \phi(\ell))$.
        Otherwise, place leaf ℓ at $(2i-1,\ \phi(\ell))$ above or below v_i.
        Draw edge v_i–ℓ. □

Lemma 3. *An n-vertex radius-2 star can be realized with straight-line edges in $O(n)$ time on a $(2n+1) \times n$ grid for any distinct labeling.*

Proof. The x-coordinates range from $-n$ to n with the root r having an x-coordinate of 0. Any adjacent leaf vertices of r have an x-coordinate of -1, one unit to the left of r. Any other neighbor u of r will either have an x-coordinate of 1, one unit to the right of r, if the label of the leaf ℓ at a distance 1 from u is greater than u or an x-coordinate of -1, otherwise.

Each leaf ℓ at a distance 2 from r is given an x-coordinate so that the edge ℓ–u has a slope of 1, i.e., $\Delta y = \Delta x$; see Fig. 3. The x-coordinate ℓ_x of ℓ can be found by solving the equation $\ell_x - u_x = \phi(\ell) - \phi(u)$ to get $\ell_x = \phi(\ell) - \phi(u) + u_x$.

This means that $\ell_x = \phi(\ell) - \phi(u) + 1$ if $\phi(\ell) > \phi(u)$, otherwise, $\ell_x = \phi(\ell) - \phi(u) - 1$. The following pseudocode takes $O(n)$ time since the coordinates of each vertex are determined in $O(1)$ time.

Draw–Radius–2–Star($T(V, E, \phi)$)

$\triangleright$ T is a radius-2 star with distinct labels.

Place r, the unique root vertex of maximum degree, at $(0,\ \phi(r))$.
For each vertex u that is adjacent to r:
    If u is a leaf vertex, place u at $(-1,\ \phi(u))$ and draw edge r–u.
    Otherwise, let ℓ be the leaf vertex that is adjacent to u.
        If $\phi(\ell) > \phi(u)$, place u at $(1, \phi(u))$ and ℓ at $(\phi(\ell)-\phi(u)+1, \phi(\ell))$.
        Otherwise, place u at $(-1,\ \phi(u))$ and ℓ at $(\phi(\ell) - \phi(u) - 1,\ \phi(\ell))$.
        Draw edges r–u and u–ℓ. □

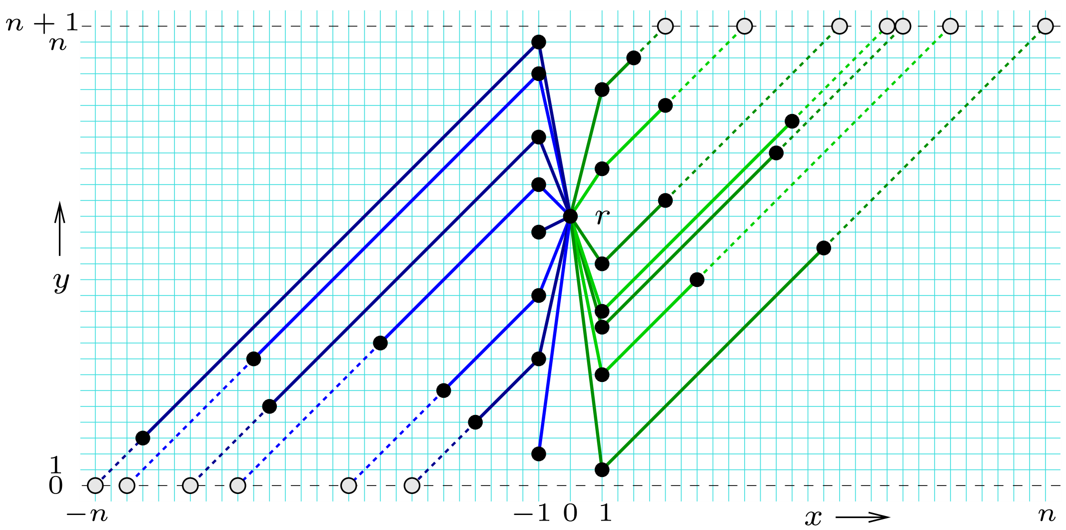


Figure 3: A realization of a radius-2 star with distinct labels on a 59×29 grid. The gray nodes indicate the intersection points of rays of slope 1 emanating from each leaf to imagined level-0 and level-$(n+1)$ that are drawn with dashed lines.

Lemma 4. *An n-vertex degree-3 spider can be realized in $O(n)$ time on an $n \times n$ grid with one bend per edge for any distinct labeling.*

Proof. We want to greedily draw T with one bend per edge starting from the root r and proceeding outwards vertex by vertex along each chain. However, we cannot draw the chains independently. Instead, we must alternate between drawing the three chains. We need to guarantee that the next vertex v of a chain can always be placed either one unit to the left (or to the right) of the leftmost (or the rightmost) point of the subtree drawn so far without introducing a crossing.

We present this algorithm in four stages. First, we give the high-level pseudocode and the two invariants it maintains. We then show how to start drawing the degree-3 spider in order to initially achieve these invariants. Afterward, we turn to more detailed aspects of the algorithm. We determine to what extent we need to draw a given chain before switching to draw the next chain, which is dictated by the invariants we maintain. Finally, we conclude with how to draw each edge so that it does not cross any of the previously drawn edges.

A chain C is drawn one vertex at a time, which is an *expansion* of C. Each subsequent vertex is placed one unit to the left or to the right (continuing in the initial direction) of the previously placed vertex. However, we stop once the last placed vertex of C becomes ϕ-extreme. If the chain C has any vertices left to place, then the chain C' whose last placed vertex is *not* ϕ-extreme is the chain to expand next in the *opposite* direction. Otherwise, once a chain C is completely drawn, one of the remaining two chains is freely expanded to the left while the other chain is expanded to the right.

To guarantee that one chain can always be expanded to the left or to the right, two invariants need to be maintained after each vertex is placed:

(1) Two of the leaves s and t of the subtree T' drawn so far are ϕ-extreme.

(2) The track ℓ_u of the third leaf u of the subtree T' either does not intersect any other part of T' to the left or to the right of u (leaving a direction that the chain of u can continue to be expanded); see Figs. 4(e), 7(d).

These invariants allow the chain C with u to be expanded in the free direction until its last placed vertex v becomes ϕ-extreme as in going from Fig. 4(a) to (b) (here s, t, and u are vertices 9, 12, and 11 in Fig. 4(a), respectively, and v is vertex 8 in Fig. 4(b)). Then v replaces one of the ϕ-extreme vertices s or t so that invariant (1) continues to hold. W.l.o.g. assume that s is no longer ϕ-extreme (in Fig. 4(b) v, vertex 8, becomes the new ϕ-extreme vertex s).

Before placing the last vertex v of C, the track of s, ℓ_s, does not intersect any other part of T', the subtree drawn so far, since s is ϕ-extreme. The chain C can intersect ℓ_s on at most one side of s after placing v, blocking that direction.

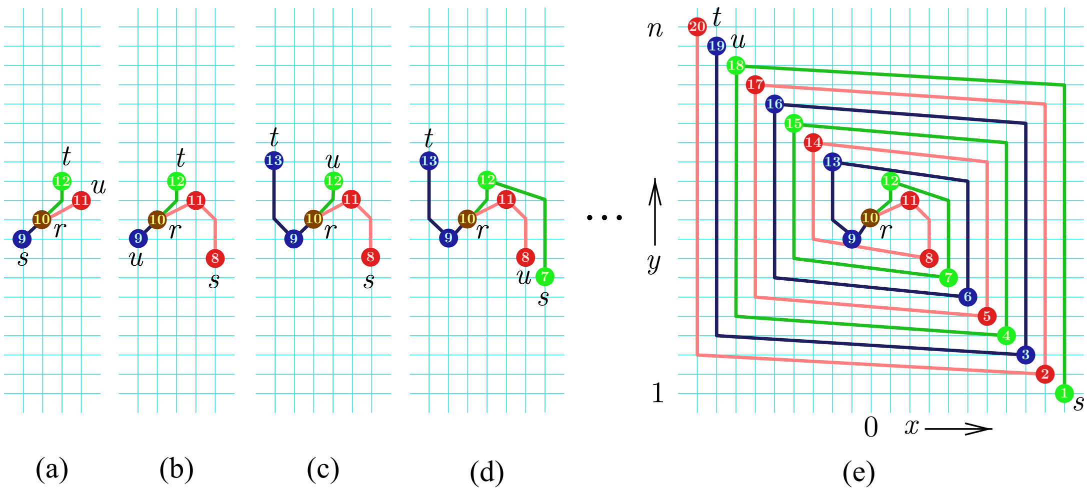


Figure 4: Step-by-step realization of a worst-case degree-3 spider with bends.

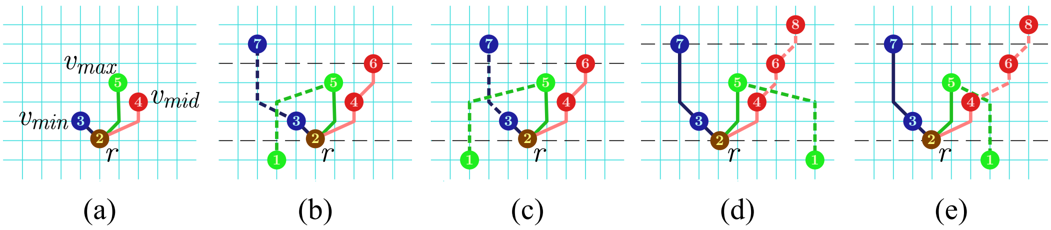


Figure 5: Four cases of expanding chains for $\phi(r) < \phi(v_{min}) < \phi(v_{mid}) < \phi(v_{max})$. All four can lead to crossings on the third expansion without taking precautions.

As a result, invariant (2) continues to hold with the old s now playing the role of the new u. The high-level algorithm Draw–Degree–3–Spider maintains these two invariants by alternating between expanding chains to the left and to the right until a vertex becomes ϕ-extreme as depicted in Figs. 4(e), 7(d).

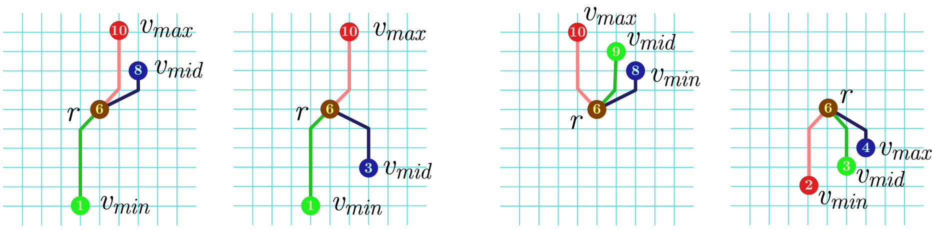
Draw–Degree–3–Spider($T(V, E, \phi)$)
▷ T is a degree-3 spider with distinct labels.
1. Let $T' \leftarrow$ Start–Drawing–Degree–3–Spider(T).
2. Let $U \leftarrow \{s, t, u\}$ be the leaves of T' such that $\phi(s) < \phi(u) < \phi(t)$.
3. Set *direction* $\leftarrow$ right
4. While u is not a leaf vertex:
5. Set $v \leftarrow$ Expand–Chain(T, T', u, *direction*).
6. If $\phi(v) < \phi(s)$ or $\phi(v) > \phi(t)$, then
7. Update $u \leftarrow s$ and $s \leftarrow v$ if $\phi(v) < \phi(s)$.
8. Update $u \leftarrow t$ and $t \leftarrow v$ if $\phi(v) > \phi(t)$.
9. Change *direction* (right to left, and vice versa).
10. Else, update $u \leftarrow v$.
11. While s is not a leaf vertex:
12. Set $s \leftarrow$ Expand–Chain(T, T', s, left).
13. While t is not a leaf vertex:
14. Set $t \leftarrow$ Expand–Chain(T, T', t, right).

Initially drawing a degree-3 spider for which the two invariants hold is non-trivial as Fig. 5 illustrates. Here v_{min}, v_{mid}, and v_{max} are the vertices that are adjacent to r such that $\phi(v_{min}) < \phi(v_{mid}) < \phi(v_{max})$. Fig. 5(a) gives an example of these three vertices with x-coordinates -1, 1, and 2, respectively.

Since v_{max} is the only ϕ-extreme leaf vertex, either the chain of v_{mid} or v_{min} can be expanded next. However, Fig. 5(b) and (c) depict two cases in which the chain of v_{mid} is first expanded to the left leaving either v_{min} or v_{max} to be expanded next to the right. This leads to a crossing on the third expansion. Fig. 5(d) and (e) depicts similar cases in which v_{min} is first expanded to the left. In all four cases a crossing is introduced. To prevent this, care must be taken while initially placing these three vertices.

If $\phi(v_{min}) < \phi(r) < \phi(v_{max})$, then both invariants hold by placing v_{min} and v_{max} one unit to the left and to the right of r, respectively, and v_{mid} to the right of v_{max}; see Fig. 6(a). Otherwise, if all three vertices have labels less than or greater than r as in Fig. 6(b), then invariant (1) does not hold. Expanding either of the other two chains in order to achieve invariant (1) may prevent invariant (2) from being achievable, which is the undesirable scenario of Fig. 5. To avoid this, the chain C that reaches the extreme point $w_{extreme}$ before it terminates or first crosses ℓ_r, i.e., the track of r, is drawn first so that it lies between the other two chains. This prevents either of those two chains from becoming trapped by an initial portion of C. Figs. 7(a)–(c) illustrate determining this extreme point $w_{extreme}$ in Fig. 7(b) among the initial portions of the three chains drawn with solid edges. The solid edges differ in Fig. 7(d) by showing the initial part of the degree-3 spider that first satisfies both invariants.

Let $v_{extreme} \in \{v_{min}, v_{mid}, v_{max}\}$ be the initial vertex of chain C with the most extreme vertex $w_{extreme}$. We first expand chain C to the right of r until C reaches $w_{extreme}$. After expanding either of the other two chains to the left so



(a) (b)

Figure 6: Four initial cases for a degree-3 spider. Invariant (1) holds for (a) but not for (b).

that its last placed vertex w_{left} becomes the other ϕ-extreme after crossing ℓ_r, invariant (1) holds (with $w_{extreme}$ and w_{left} playing the roles of vertices s and t in invariant (1)). Placing the third initial vertex v_{right} one unit to the right of r then achieves invariant (2) (with v_{right} playing the role of vertex u in invariant (2)). Afterward, both invariants are satisfied and the next expansion starts from the right. This is done with the following pseudocode.

Start–Drawing–Degree–3–Spider($T(V, E, \phi)$)
$\triangleright$ Initially draw degree-3 spider T until both invariants hold.
1. Place r, the root vertex of degree 3, at $(0, \phi(r))$.
2. Let $U \leftarrow \{v_{min}, v_{mid}, v_{max}\}$ be the vertices that are adjacent to r such that $\phi(v_{min}) < \phi(v_{mid}) < \phi(v_{max})$.
3. Let $T'(V', E')$ be the tree drawn so far where $V' \leftarrow \{r\}$ and $E' \leftarrow \varnothing$.
4. If $\phi(v_{min}) < \phi(r) < \phi(v_{max})$, then
   Draw–Bent–Edge(T, T', r, v_{min}, left),
   Draw–Bent–Edge(T, T', r, v_{max}, right), and
   Draw–Bent–Edge(T, T', r, v_{mid}, right).
5. Otherwise,
6. Set $w_{extreme} \leftarrow r$, the current vertex that is the most ϕ-extreme:
7. For each v in U:
8. Set $w'_{extreme} \leftarrow$ Get–Extreme(T, r, v).
9. If $\phi(r) \leq \phi(w_{extreme}) < \phi(w'_{extreme})$ or
   $\phi(r) \geq \phi(w_{extreme}) > \phi(w'_{extreme})$,
   then set $w_{extreme} \leftarrow w'_{extreme}$ and $v_{extreme} \leftarrow v$.
10. Let v_{left} and v_{right} be the two vertices in U other than $v_{extreme}$.
11. Draw $v_{extreme}$ with Draw–Bent–Edge(T, T', r, $v_{extreme}$, right) and expand with Expand–Chain(T, T', $v_{extreme}$, right, $w_{extreme}$).
12. Draw v_{right} with Draw–Bent–Edge(T, T', r, v_{right}, right).
13. Draw v_{left} with Draw–Bent–Edge(T, T', r, v_{left}, left) and expand with Expand–Chain(T, T', w_{left}, left, null).

Here Start–Drawing–Degree–3–Spider determines the initial extreme of a chain with the following procedure.

Get–Extreme($T(V, E, \phi)$, r, u)
$\triangleright$ Find extreme of chain with vertex u in a degree-3 spider T with root r.
1. Set *extreme* $\leftarrow \phi(u)$, the current extreme to the label of u.
2. While there is another next vertex v along the chain starting with u that does not cause the chain to cross ℓ_r, the track of r:
3. If $\phi(u) > \phi(r)$, increase *extreme* to $\phi(v)$ if $\phi(v) >$ *extreme*.
4. Otherwise, decrease *extreme* to $\phi(v)$ if $\phi(v) <$ *extreme*.
5. Return $w_{extreme}$, the vertex with the label of *extreme*.

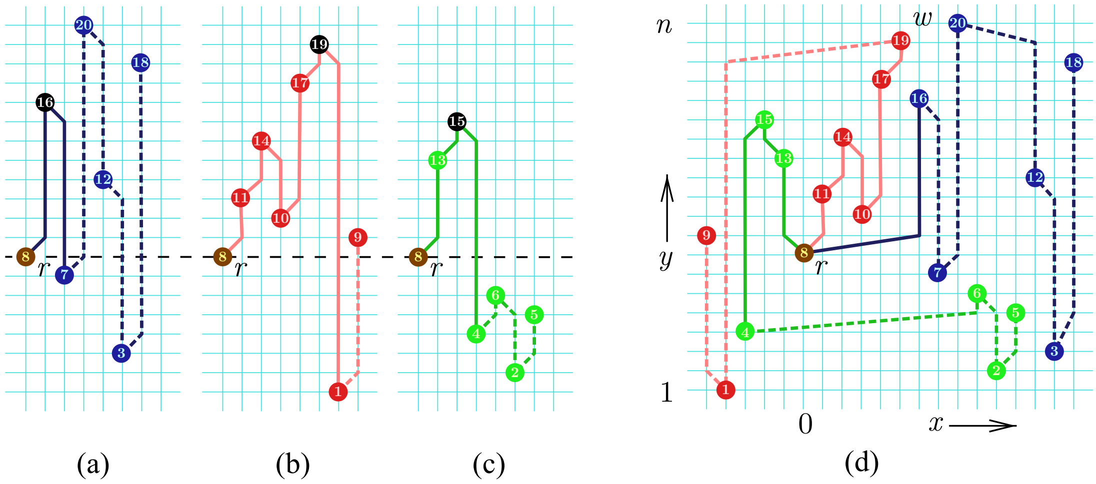


Figure 7: Determining the initial extreme used to start drawing a degree-3 spider.

Expansion of a chain is then accomplished by the next procedure.

Expand–Chain($T(V, E, \phi)$, $T'(V', E')$, u, *direction*, w)
▷ T' is the subtree of T drawn so far. Expands the chain starting at u to the right if *direction* is right, and to the left otherwise. Specifying the optional vertex w forces the expansion to go at least to w even after a vertex of the chain becomes ϕ-extreme.

1. Let ϕ_{max} and ϕ_{min} be the maximum and minimum labels of $V' \cup \{w\}$.
2. For the next vertex v that is adjacent to u in T that is not in V':
3. Draw–Bent–Edge(T, T', u, v, *direction*) and set $u \leftarrow v$.
4. Until $\phi(v) > \phi_{max}$ or $\phi(v) < \phi_{min}$ or v is w or v is a leaf vertex.
5. Return v, the last vertex that was added to T'.

Finally, we consider how to draw each edge with a bend. If no part of T' lies directly to the left (or to the right) of the last vertex u of the chain, then u could reach vertex v. Route the edge to the left (or to the right) from u that is above (or below) all the other vertices to a bend directly above (or below) v. From the bend, the edge proceeds directly downwards (or upwards) to v.

Bend b has the same x-coordinate as v. The y-coordinate of b is determined by whether the previous vertex u of v is above or below v. If $\phi(u) > \phi(v)$, we place b one unit below u, otherwise, we place b one unit above u. This is so that if u is ϕ-extreme, the line segment u–b will not cross any of the edges of the subtree T' drawn so far.

The x-coordinate of v is one greater (or one less) than the maximum (or minimum) x-coordinate of the tree T' drawn so far if the edge is to be drawn to the right (or to the left); see Fig. 8(a) and (b). This procedure of drawing edge u–v with bend b is given by the following pseudocode.

Draw–Bent–Edge($T(V, E, \phi)$, $T'(V', E')$, u, v, *direction*)
▷ T' is the subtree of T drawn so far. Vertex u has been placed and vertex v is to be drawn to the right of u if *direction* is right, and to the left of u, otherwise.

1. Let x_{max} and x_{min} be maximum and minimum x-coordinates of T'.
2. If *direction* is right, set $v_x \leftarrow x_{max} + 1$, otherwise set $v_x \leftarrow x_{min} - 1$.
3. If $\phi(u) < \phi(v)$, set $b_y \leftarrow \phi(u) + 1$. Otherwise, set $b_y \leftarrow \phi(u) - 1$.
4. Place v at $(v_x, \phi(v))$, bend b at (v_x, b_y), and draw edges u–b and b–v.
5. Update T' by adding v to V' and (u, v) to E'.

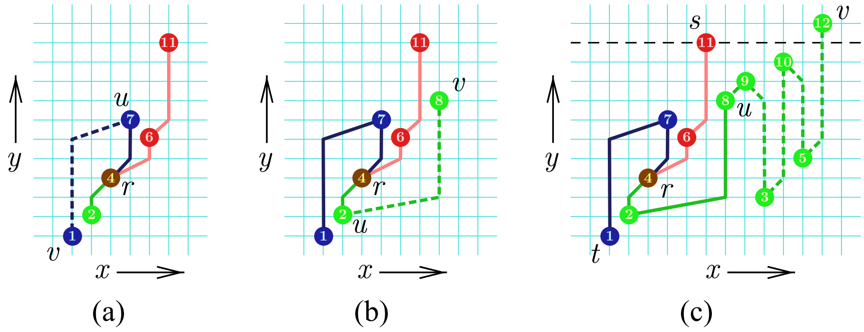


Figure 8: In (a) and (b), edge u–v is drawn to the left and to the right of u using Draw–Bent–Edge. In (c), the right chain is expanded to the right from u using Expand–Chain until v replaces s as the ϕ-maximum.

Here Draw–Bent–Edge draws the edge u–v with bend b so that b is either one unit above or below u depending on whether v is above or below u. This avoids any crossings since invariant (2) ensures no part of T' lies along the track of u in the direction of the expansion.

Start–Drawing–Degree–3–Spider takes $O(n)$ time since each vertex is placed in $O(1)$ time and each of the three calls to Get–Extreme take $O(n)$ time. Afterward, each vertex is placed in $O(1)$ time in Draw–Degree–3–Spider, leading to an overall $O(n)$ running time. Since the drawing is widened one unit per vertex, the drawing uses $n \times n$ space. □

Lemma 5. *An n-vertex degree-3 spider can be realized with no bends in $O(n)$ time though it may require up to $O(n!) \times n$ area for some distinct labelings.*

Proof. The algorithm of Lemma 4 can be modified to use straight-line edges with the following edge drawing algorithm in lieu of Draw–Bent–Edge.

Draw–Straight–Edge($T(V, E, \phi)$, $T'(V', E')$, u, v, *direction*)
▷ T' is the subtree of T drawn so far. Vertex u has been placed and vertex v is to be placed next.

1. Let x_{max} and x_{min} be maximum and minimum x-coordinates of T'.
2. If *direction* is right, let $x_v > x_{max}$ be the least such integer in which edge u–v would not intersect T'.
3. Otherwise, let $x_v < x_{min}$ be the greatest such integer in which edge u–v would not intersect T'.
4. Place v at $(x_v, \phi(v))$ and draw edge u–v.
5. Update T' by adding v to V' and (u, v) to E'.

Figure 9 gives a degree-3 spider that requires exponential area when drawn using this modified algorithm. At each step in the algorithm, there is only one choice when placing the next vertex so that the three chains spiral about each other.

We bound the value of x_v at step j of the algorithm. Let h_j and w_j denote the height and width of the subtree drawn up and to step j. Let a–b–c, o–p–q, and u–v–w be the last two edges of the three chains as shown in Fig. 9 in which edges a–b, o–p, u–v, b–c, p–q, and v–w are drawn in steps $i, i+1, \ldots, i+5$, respectively. For the last edge v–w in step $i+5$ not to have a bend, it cannot intersect any part of the tree drawn so far. This implies that v–w must lie below the ϕ-minimal vertex b from step i. Since v was placed in step $i+2$, the difference between the x-coordinates of v and b (subtracting the extra width $w_{i+1} - w_i$ from drawing o–p in step $i+1$) is

$$x_v - x_b = (w_{i+2} - w_i) - (w_{i+1} - w_i) = w_{i+2} - w_{i+1}.$$

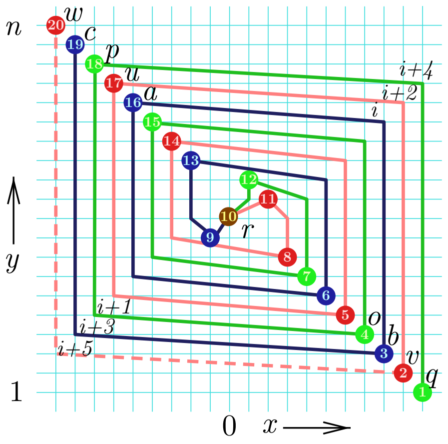


Figure 9: A degree-3 spider that can require $O(n!) \times n$ space when realized with no bends.

Similarly, the difference between the x-coordinates of w and v (subtracting the extra width $w_{i+4} - w_{i+2}$ from drawing p–q in step $i + 4$) is

$$x_w - x_v = (w_{i+5} - w_{i+2}) - (w_{i+4} - w_{i+2}) = w_{i+5} - w_{i+4}.$$

The slope of the edge v–w is strictly greater than $-1/(x_b - x_x)$, which gives the most compact drawing. The height difference between w and v is the height at step $i + 5$ minus the extra height of 1 from placing q in step $i + 4$. Hence,

$$x_w - x_v = (y_w - y_v)/(\text{slope of } v\text{–}w) = (h_{i+5} - 1) \cdot (x_v - x_b).$$

Since $h_j = j$, we combine the previous three equations to solve for w_{i+5} as

$$\begin{aligned} w_{i+5} &= (i+4)(w_{i+2} - w_{i+1}) + w_{i+4} \\ &= (i+4)(w_{i+2} - w_{i+1}) + (i+3)(w_{i+1} - w_i) + w_{i+3} \\ &\vdots \\ &= \sum_{k=4}^{i+4} k(w_{k-2} - w_{k-3}). \end{aligned}$$

Substituting for $j = i + 5$, we determine the recurrence for w_j to be

$$\begin{aligned} w_j &= \sum_{k=4}^{j-1} k(w_{k-2} - w_{k-3}) \\ &= (j-1)w_{j-3} - (j-1)w_{j-4} + (j-2)w_{j-4} - (j-2)w_{j-5} + \ldots - w_1 \\ &= (j-1)w_{j-3} - w_{j-4} - w_{j-5} - \ldots - w_1 \\ &= (j-1)w_{j-3} - \sum_{k=1}^{j-4} w_k. \end{aligned}$$

Finally, we solve the recurrence for the increase in width Δ_j at step j as

$$
\begin{aligned}
w_j - w_{j-1} &= \Big((j-1)w_{j-3} - w_{j-4} - \sum_{k=1}^{j-5} w_k\Big) - \Big((j-2)w_{j-4} - \sum_{k=1}^{j-5} w_k\Big) \\
&= (j-1)(w_{j-3} - w_{j-4}) \\
\Delta_j &= (j-1)\Delta_{j-3} = (j-1)(j-4)\cdots 1.
\end{aligned}
$$

Hence, we have $(\frac{j-1}{3})! < (j-4)(j-7)\cdots 1 < (j-4)! < \Delta_j < (j-1)!$ as bounds. This shows that this tree requires exponential area using this modified algorithm. The width of the degree-3 spider at step j can then be bounded as

$$w_j = \sum_{k=1}^{j-1} k! = (j-1)(j-1)! < j!$$

This tree is a worst-case for our algorithm in terms of the amount of area used in each step. We observe that by placing the j^{th} vertex of any degree-3 spider T at a distance of $|j!|$ from r in the appropriate positive or negative x-direction, which is more than strictly necessary, we are guaranteed to avoid a crossing in T. Hence, the algorithm uses at most $2n! \times n$ area. □

Combining Lemmas 2, 3, 4, and 5, we have our first theorem.

Theorem 6. *Caterpillars, radius-2 stars, and degree-3 spiders are all ULP with one vertex per level. Each can be realized in $O(n)$ time.*

3.2. Forbidden Trees for ULP Trees with Distinct Labels

We next introduce the forbidden subtrees T_8 and T_9 shown in Fig. 10.

Lemma 7. *There exist labelings preventing T_8 and T_9 from being level planar when distinct labels are used.*

Proof. First, we consider T_8 in Fig. 10(a) with a distinct labeling satisfying $\{\phi(a), \phi(f)\} > \phi(d) > \{\phi(g), \phi(c)\} > \phi(b) > \{\phi(e), \phi(h)\}$ (or its reverse). We contend that these labelings are level non-planar. To prevent the chain a–b–c–d–e from

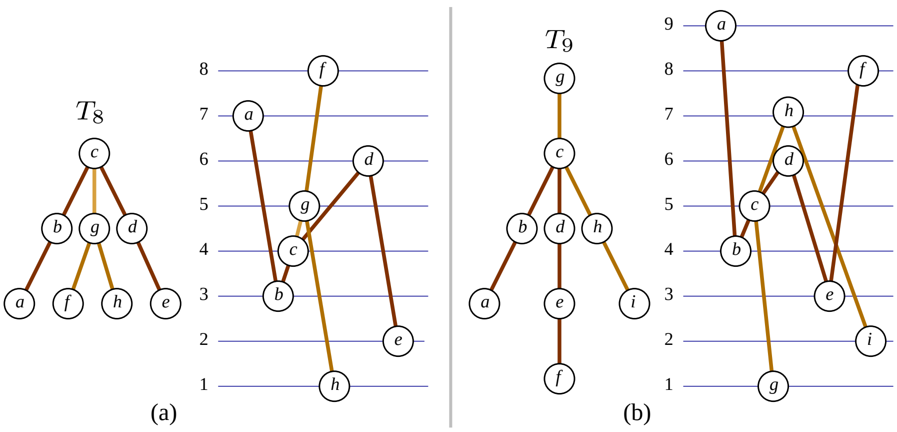


Figure 10: Distinct labelings preventing T_8 and T_9 from being ULP.

self intersecting, c must lie between the intersections of a–b and d–e with the track ℓ_c of c. The edge c–g forces g to also lie between the intersections of a–b and d–e with the track ℓ_g. There are two cases: either (i) g lies between the intersection of a–b with ℓ_g and the intersection of c–d (if $\phi(c) < \phi(g)$) or b–c (if $\phi(c) > \phi(g)$) with ℓ_g, in which case g–h must cross an edge of the chain a–b–c–d, or (ii) g lies between the intersections of c–d (if $\phi(c) < \phi(g)$) or b–c (if $\phi(c) > \phi(g)$) with ℓ_g and the intersection of d–e with ℓ_g, in which case g–f must cross an edge of chain b–c–d–e.

Next, we consider T_9 in Fig. 10(b) with a distinct labeling satisfying the partial order $\{\phi(a), \phi(f)\} > \phi(h) > \phi(d) > \phi(c) > \phi(b) > \phi(e) > \{\phi(g), \phi(i)\}$ (or its reverse). Such a labeling can also be shown to level non-planar. Again to prevent the chain a–b–c–d–e from self intersecting, c must lie between the intersections of a–b and d–e with track ℓ_c. W.l.o.g. assume that a–b intersects ℓ_c to the right of where d–e intersects ℓ_c. To prevent the chain a–b–c–d–e–f from self intersecting, there are two cases to consider: either (i) e–f intersects ℓ_c to the left of where a–b intersects ℓ_c or (ii) e–f intersects ℓ_c to the right of where d–e intersects ℓ_c. For case (i), c–g must either intersect ℓ_e to the left of e, in which case it must cross e–f, or to the right of e, in which case it must cross d–e. For case (ii), either h lies to left of where a–b intersects ℓ_h in which case c–h must cross a–b, h lies to right of where e–f intersects ℓ_h in which case c–h must cross e–f, or h lies between where a–b and e–f intersect ℓ_h in which case h–i must cross an edge of chain a–b–c–d–e–f as in Fig. 10(b). □

This leads to the following corollary:

Corollary 8. *If a tree contains a subtree homeomorphic to T_8 or T_9, then it cannot be ULP with distinct labels.*

Proof. We provide a labeling ϕ of a tree T containing a subtree homeomorphic to a level non-planar tree $\tilde{T}$, which can be either T_8 or T_9 by Lemma 7. Let h be the homeomorphism that maps an edge in $\tilde{T}$ to the path in T and a vertex in $\tilde{T}$ to the endpoint of the path in T. Label the vertices of $\tilde{T}$ using an appropriate labeling ϕ' from Lemma 7 that forces a crossing in $\tilde{T}$.

We maintain the same relative ordering of the labels in T as in $\tilde{T}$. In particular, we want $\phi(h(u)) < \phi(h(v))$ if and only if $\phi'(u) < \phi'(v)$ for each edge (u, v) in $\tilde{T}$. For each path $h((u, v)) = p_{(u,v)} = v_1$–$v_2$–$\cdots$–$v_k$ in T that corresponds to an edge (u, v) in $\tilde{T}$, we want $\phi(v_1) < \phi(v_2) < \cdots < \phi(v_k)$ if $\phi'(u) < \phi'(v)$. We can assign the other vertices of T not in the image of h arbitrary labels. Then every edge (u, v) in $\tilde{T}$ corresponds to a strictly monotone path $p_{(u,v)}$ in T preserving the non-planarity of the realization of $\tilde{T}$. □

We next show that T_8 and T_9 are minimal level non-planar trees.

Lemma 9. *Removing any edge from T_8 or T_9 yields a forest of ULP trees.*

Proof. If removing an edge from T_8 of Fig. 10(a) decreases the degree of the vertices c and/or g, then the resulting graph is either a forest of (i) a caterpillar and a lone edge (after removing b–c or c–d), (ii) two paths (after removing c–g), or (iii) a degree-3 spider (after removing either f–g or g–h). Otherwise, removing either a–b or d–e, which maintains the degree of both c and g, yields (iv) a caterpillar with a spine of length 5. Moving onto T_9 of Fig. 10(b), if removing an edge maintains the degree of vertex c, then the resulting graph must be a forest of either (i) a caterpillar (after removing a–b, d–e or h–i) and the possible lone edge e–f (if d–e was removed) or (ii) a radius-2 star (after removing e–f). On the other hand, if the degree of c decreases to 3, then the resulting graph is a (iii) degree-3 spider and, possibly, a path. □

We can now complete our characterization of ULP trees with distinct labels.

Lemma 10. *Every tree either contains a subtree homeomorphic to T_8 or T_9 or it is a caterpillar, a radius-2 star, or a degree-3 spider.*

Proof. Any tree T that is not a caterpillar must contain a lobster. One can simply remove leaf vertices of T until a lobster remains. Every lobster must contain a subtree isomorphic to a minimal lobster T_7 (a $K_{1,3}$ with each edge subdivided once) since any lobster has at least one vertex r of degree 3 and the three vertices a, b, and c that are at distance 2 from r; see Fig. 11(a). Both T_8 and T_9 each contain a subtree isomorphic to T_7; hence, they cannot be caterpillars. T_8 cannot be a radius-2 star or a degree-3 spider because it has two vertices of degree 3. Since T_9 has radius 3 and a vertex of degree 4, it also cannot be a radius-2 star or a degree-3 spider. By Lemma 9, both T_8 and T_9 are minimal examples of trees that are not caterpillars, radius-2 stars, or degree-3 spiders. We next show that trees

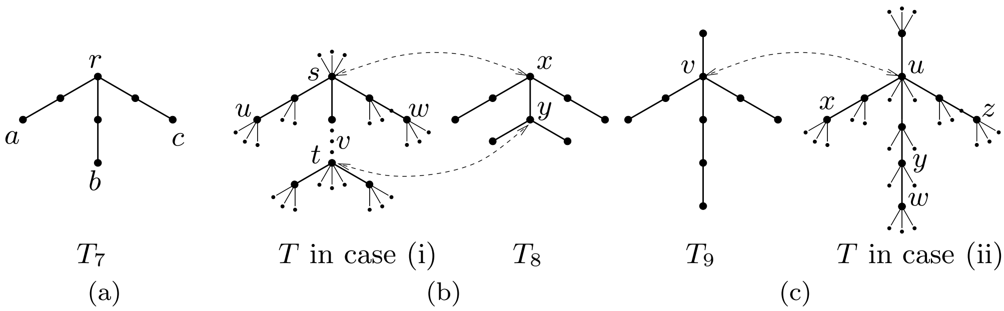


Figure 11: Homeomorphic copies of T_8 and T_9 in trees for Lemma 10.

without a subtree homeomorphic to T_8 or T_9 are one of the three classes of ULP trees with distinct labels given by Theorem 6.

Assume then that T is not in any of these three classes of trees. Since T is not a degree-3 spider, there are two cases: either T has (i) two vertices s and t with degree of at least 3 or (ii) one vertex u with degree k greater than 3. In case (i), we find a subtree of T homeomorphic to T_8. Let x and y denote the two vertices of degree 3 in T_8, where y is the one with adjacent leaf vertices; see Fig. 11(b). Since T is not a caterpillar it must have a subtree isomorphic to T_7. W.l.o.g. let s be the vertex in T corresponding to the root vertex r of T_7, and let t be any other vertex with degree of at least 3 in T.

We map the vertices s and t from T to the vertices x and y from T_8, respectively. Since t has degree of at least 3 in T, there exist two neighbors of t not along the path from s to t in T, which we map to the two vertices that correspond to the leaf vertices adjacent to y in T_8. Only one of the three vertices u, v, and w in T, corresponding to the leaf vertices a, b, and c of T_7, can be along the path s to t in T. Suppose w.l.o.g. it is the vertex v that corresponds to b. Then the vertices u and w in T that correspond to a and c in T_7 can be mapped to the two remaining leaf vertices in T_8. This completes the mapping of vertices of T_8, showing that T contains a subtree homeomorphic to T_8. The only subdivided edge of T_8 is x–y that maps to the path from s to t in T.

Next we consider case (ii) in which we find the subtree in T homeomorphic to T_9. The one vertex u in T of degree k greater than 3 must be the vertex corresponding to the root vertex of the subtree in T isomorphic to T_7; see Fig. 11(c). Otherwise, if there were separate vertices of degree greater than 3, case (i) would apply. Let u be mapped to the degree-4 vertex v of T_9. Since T is not a radius-2 star, there exists a vertex w at a distance 3 from u, which can be mapped to the leaf vertex in T_9 at a distance 3 from v.

Only one of the three vertices x, y, and z in T, corresponding to the leaf vertices a, b, and c of T_7, can be along the path from u to w. W.l.o.g. suppose b corresponds to the vertex y along the path from u to w. The other two vertices x and z in T that correspond to a and c in T_7 can be mapped to the other two leaf vertices in T_9. The remaining leaf vertex of T_9 adjacent to v can be mapped to the fourth vertex adjacent to u in T since u has degree greater than 3. Hence, T has a subtree that is homeomorphic to T_9. □

Combining Theorem 6 and Corollary 8 with Lemma 10 gives our main theorem characterizing ULP trees with distinct labels.

Theorem 11. *The following three statements are equivalent:*

1. *T does not contain a subtree homeomorphic to T_8 or T_9.*
2. *T is a caterpillar, a radius-2 star, or a degree-3 spider.*
3. *T is ULP trees with distinct labels.*

4. Unlabeled Level Planar Trees with Duplicate Labels

First, we show that caterpillars are the only ULP trees with duplicate labels and then show that T_7 is the only minimal forbidden subtree.

4.1. Drawing ULP Trees with Duplicate Labels

We extend Lemma 2 to compute a linear-time realization of a caterpillar for any labeling ϕ by showing that it is also ULP with duplicate labels. For any nonempty subset U of V, we define $Dup(U)$ to be the number of vertices in U with "duplicate" labels, i.e., $Dup(U) = 0$ if all of U have distinct labels, whereas, $Dup(U) = |U| - 1$ if all of U have the same label. For a tree, we also define $L_{above}(v)$ and $L_{below}(v)$ to be the sets of leaf vertices that are adjacent to v in T with labels less than and greater than $\phi(v)$, respectively. The distance between adjacent spine vertices v_i and v_{i+1} is then a function of the number of duplicate labels of the leaf vertices of v_i as given by the following lemma.

Lemma 12. *An n-vertex caterpillar on k levels with an m-vertex spine can be realized with straight-line edges in $O(n)$ time on a $(m + b) \times k$ grid for any labeling where $b = \sum_{i=1}^{m} \max\{Dup(L_{above}(v_i)),\ Dup(L_{below}(v_i))\}$.*

Proof. We draw the spine v_1–v_2–$\cdots$–v_m from the left to the right so that the leaf vertices of $L_{above}(v_i)$ and $L_{below}(v_i)$ lie to the right of v_i for $i \in [1..m]$. With a clockwise (or counterclockwise) radial sweep, we draw each vertex in $L_{above}(v_i)$ (or $L_{below}(v_i)$) at the next available grid point. Drawing the spine edge v_i–v_{i+1} with the leaf vertices of v_i takes a total of $\left(1 + \max\{Dup(L_{above}(v_i)),\ Dup(L_{below}(v_i))\}\right) \times k$ space.

Place v_{i+1} at the next x-coordinate to the right of the leaf vertices of v_i; see Fig. 12. Since each of the edges ℓ–v_i incident to v_i have unique slopes, at most one leaf ℓ might lie along the edge v_i–v_{i+1}. In this case, ℓ is moved to the left so as to have the same x-coordinate as v_i.

This drawing is then a realization with straight-line edges since each leaf incident to v_i is either drawn above or below v_i or to the right of v_i in order to avoid all crossings. The pseudocode for this algorithm is given next.

Draw–Caterpillar($T(V, E, \phi)$)
▷ T is a caterpillar with distinct or duplicate labels.
1. Let S, v_1–v_2–$\cdots$–v_m, be the spine of T, and L be the leaves of T.
2. Perform a counting sort on L with key_1 and then with key_2 such that $key_1(\ell) = \phi(\ell)$ and $key_2(\ell) = i$ for each leaf ℓ in L adjacent to v_i.
3. Let $L_1, L_2, \ldots, L_m$ be sublists where each ℓ in L_i is adjacent to v_i.
4. Initialize x, the x-coordinate of the current spine vertex, to 1.
5. For each v_i for $i \in [1..m]$:
6. Place v_i at $(x, \phi(v_i))$ and draw spine edge v_i–v_{i+1} if $i > 1$.

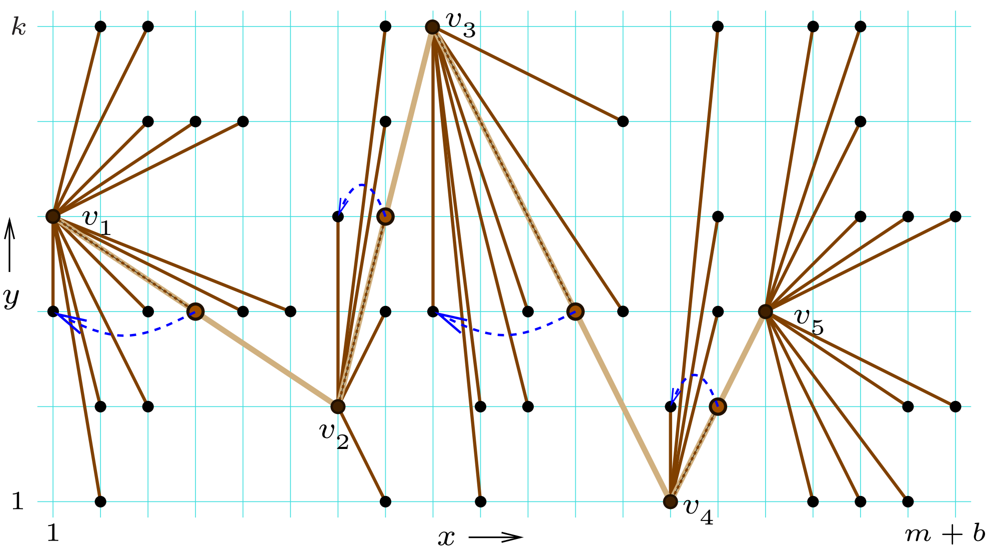


Figure 12: A realization of a 45-vertex caterpillar with duplicate labels on a 20×6 grid. Arrows indicate how vertices initially placed on spine edges are moved in order to avoid any edge overlaps.

7. Set x_a and x_b, the x-coordinates of L_i above and below v_i, to $x + 1$.
8. For next leaf ℓ in L_i starting at the beginning so that $\phi(\ell) > \phi(v_i)$:
9. Increment x_a if last leaf had the same label. Place ℓ at $(x_a, \phi(\ell))$.
10. For next leaf ℓ in L_i starting at the end so that $\phi(\ell) < \phi(v_i)$:
11. Increment x_b if last leaf had the same label. Place ℓ at $(x_b, \phi(\ell))$.
12. Update $x = \max\{x_a, x_b\} + 1$.
13. For each leaf ℓ adjacent to spine vertex v with x-coordinate v_x:
14. If ℓ lies on spine edge v–w, move ℓ to $(v_x, \phi(\ell))$.
15. Draw edge ℓ–v.

Step 2 forms a linear-time radix sort on the leaf vertices that allows processing in clockwise and counterclockwise directions. The two calls made to counting sort each take $\Theta(n + k)$ time, sorting the adjacency lists of all the leaf vertices simultaneously. Otherwise, it would take $\Theta(m(n + k))$ time if the lists were sorted separately for each of the m spine vertices. As a result, Draw–Caterpillar runs in $O(n)$ time since each vertex is placed in $O(1)$ time. □

Corollary 13. *Caterpillars on k levels are ULP for any $0 \leq k \leq n$. Each can be straight-line realized in $O(n)$ time on an $O(n) \times n$ grid for any labeling.*

4.2. Forbidden Tree for ULP Trees with Duplicate Labels

The forbidden tree T_7 in Fig. 13 is not ULP with duplicate labels for the given labelings that force a self intersection.

Lemma 14. *There exists a duplicate labeling that prevents T_7 from being level planar on k levels for any $2 \leq k < n$.*

Proof. Let C and C' denote chains a–b–c–d–e and a–b–c–g–f, respectively. For T_7, if $k = 2$, let ϕ obey $\phi(a) = \phi(c) = \phi(f) = \phi(e) > \phi(b) = \phi(d) = \phi(g)$. W.l.o.g. assume that both C and C' each proceed left to right in order to avoid self intersections. This means that a–b intersects track ℓ_a to the left of where c and f intersect ℓ_a and track ℓ_b to the left of where d and g intersect ℓ_b, whereas, d–e and f–g intersects ℓ_a to the right of where c intersects ℓ_a. In order for c–d not to cross d–e, c–d must intersect ℓ_b to the left of where d intersects ℓ_b. However, f–g must then cross c–d.

For T_7, if $2 < k < n$, let ϕ obey $\phi(a) \geq \phi(d) = \phi(g) > \phi(c) > \phi(b) \geq \phi(\{e, f\})$. Assume w.l.o.g. that C proceeds left to right. For C to avoid a self intersection, a–b intersects ℓ_c to the left of c and ℓ_d to the left of d, whereas, d–e intersects ℓ_c to the right of c and ℓ_b to the right of b. For a–b to avoid crossing c–g, a–b must intersect ℓ_g to the left of g while d–e must intersect ℓ_g to the right of g since $\ell_g = \ell_d$. However, this implies f–g must cross the chain a–b–c–d. □

Corollary 15. *A tree $T(V, E)$ cannot be ULP with duplicate labels if T contains a subtree isomorphic to T_7.*

Proof. We give a non-planar labeling ϕ if T contains a subtree homeomorphic to T_7. Any such homeomorphic subtree must contain a subtree $T'(V', E')$ isomorphic to T_7. This is because the homeomorphic subtree only has one vertex of degree 3 that would be mapped to the corresponding root of T_7. Assign the vertices of V' using a labeling from Lemma 14 preventing T' from being ULP with duplicate labels. Since this is an isomorphism, we can assign the other vertices of T to any of the remaining levels. Given that the subtree T' has a self-intersection with ϕ, so must T. □

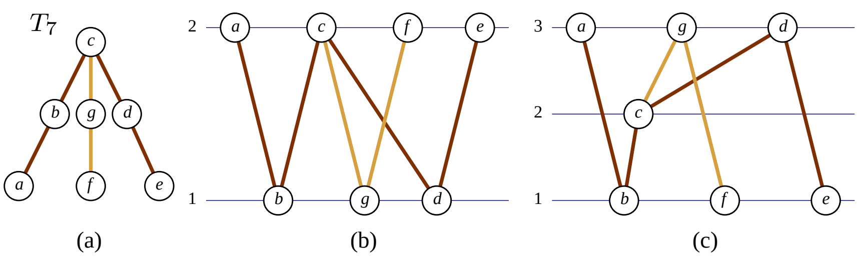


Figure 13: Level assignments that prevent T_7 from being ULP with duplicate labels.

Next, we show that T_7 is minimal with the following lemma.

Lemma 16. *Removing any edge from T_7 yields a forest of caterpillars.*

Proof. Removing an edge from T_7 incident to its root c; see Fig. 13(a), leaves a path and a lone edge. Otherwise, removing an edge leaves a caterpillar. □

Next, we prove that if a tree does not have a subtree isomorphic to T_7 then it must be a caterpillar.

Lemma 17. *An n-vertex tree T either contains a subtree isomorphic to T_7 or T is a caterpillar.*

Proof. One can repeatedly remove leaf vertices from any tree that is not a caterpillar until one has a lobster. One can continue removing leaf vertices from any lobster until one has the lobster T_7. The lobster T_7 is minimal since it cannot have any more leaf vertices removed without becoming a caterpillar by Lemma 16. Hence, every lobster must contain a subtree isomorphic to T_7.

By definition, a caterpillar cannot contain a subtree that is isomorphic to a lobster such as T_7. Hence, the set of all trees is clearly partitioned between those with a subtree isomorphic to T_7, which are not ULP, and those without such a subtree, which are caterpillars. □

Combining Corollaries 13 and 15 with Lemma 17 gives our main theorem characterizing ULP trees with duplicate labels.

Theorem 18. *The following three statements are equivalent:*

1. *T does not contain a subtree isomorphic to T_7.*
2. *T is a caterpillar.*
3. *T is ULP with duplicate labels.*

5. Linear Time Recognition of ULP Trees

While any ULP tree can be drawn in linear-time, the question remains how to determine if a tree is ULP before doing so. The next theorem gives our linear-time recognition algorithm.

Theorem 19. *Any ULP n-vertex tree $T(V,E)$ can be recognized in $O(n)$ time.*

Proof. If the number of levels is less than n, this implies that there are duplicate labels in which case we only need to determine if T is a caterpillar. Otherwise, we also need to determine whether T is a radius-2 star or a degree-3 spider. This is done with the following pseudocode.

```
Is−Caterpillar(T(V, E))                                  ▷ T is a tree.
  1. Let T′ be the subtree of T given by Remove−Leaves(T).
  2. Return true if T′ is a path; return false otherwise.

Is−Radius−2−Star(T(V, E))                                ▷ T is a tree.
  1. Let T′ be the subtree of T given by Remove−Leaves(T).
  2. Let T′′ be the subtree of T′ given by Remove−Leaves(T′).
  3. Return true if T′′ has only one vertex r and all the other vertices
     in T′ have degree 2 in T; return false otherwise.

Is−Degree−3−Spider(T(V, E))                              ▷ T is a tree.
  1. Return true if the maximum degree of T is 3 and if T has only one
     vertex of degree 3; return false otherwise.
```

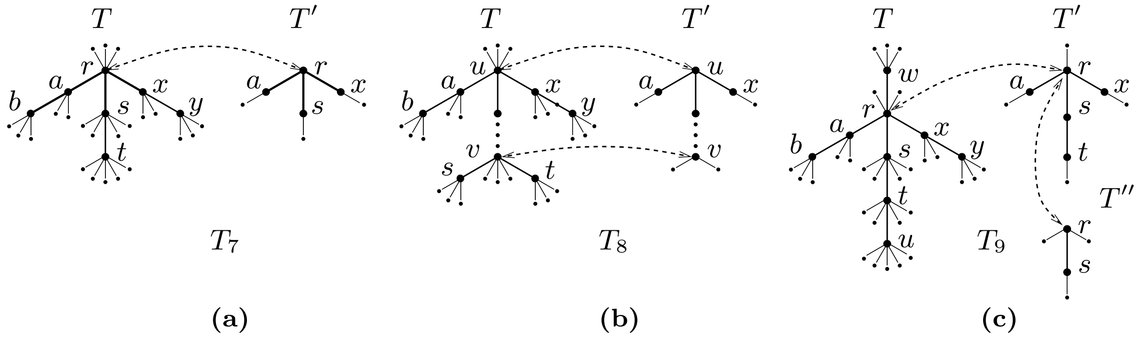


Figure 14: Finding T_7, T_8 and T_9 in T by removing the leaf vertices in T to get the subtree T' in (a), (b) and (c), and repeating this process with T' to get the subtree T'' in (c).

Is–ULP–Tree($T(V, E)$, k) ▷ T is a graph with k labels.

1. Return false if T is not a tree.
2. If $k < |V|$ return Is–Caterpillar(T).
3. Otherwise, return Is–Caterpillar(T) **or** Is–Radius–2–Star(T) **or** Is–Degree–3–Spider(T). □

If a tree is not ULP, then we know by Theorems 11 and 18 that the tree must contain a subtree homeomorphic to one of the forbidden trees. The next two theorems show how this can also be done in linear time.

Theorem 20. *A subtree isomorphic to T_7 can be found in any n-vertex tree $T(V, E)$ that is not ULP with duplicate labels in $O(n)$ time.*

Proof. By Lemma 17, if T is not ULP with duplicate labels, then T must contain a subtree isomorphic to T_7. By removing all leaf vertices from T, we obtain T'. We look for any vertex in T' of degree 3 or more, which then corresponds to the root r of the lobster T_7 in T; see Fig. 14(a). This allows us to find a subtree isomorphic to T_7 in $O(n)$ time as follows:

Find–T_7–Subtree($T(V, E)$)
▷ T is a tree that is not ULP with duplicate labels.

1. Let T' be the subtree of T given by Remove–Leaves(T).
2. Let r be a vertex of degree at least 3 in T' and let a, s, and x be any three neighbors of r.
3. Let b, t, and y be any neighbors (other than r) of a, s, and x in T.
4. Return the induced subtree of T on the vertices $\{r, a, b, s, t, x, y\}$. □

Theorem 21. *A subtree homeomorphic to T_8 or isomorphic to T_9 can be found in any n-vertex tree $T(V, E)$ that is not ULP with distinct labels in $O(n)$ time.*

Proof. By Lemma 10, if T is not ULP with distinct labels, we may assume that it either contains a subtree homeomorphic to T_8 or to T_9. If there exists a homeomorphic copy of T_8 in T, then the edge u–v between the vertices of degree at least 3 is the only subdivided edge.

To find this subdivided edge of T_8, we first take any vertex u of degree 3 or more in T' (the subtree of T obtained by removing all of its leaf vertices); see Fig. 14(b). This corresponds to the root of the lobster T_7 in T_8. Any remaining vertex of degree 3 or more in T can then play the role of v. Comparing T and T' in this way allows us to find a subtree homeomorphic to T_8 if one exists in $O(n)$ time as follows:

Find–T_8–Subdivision($T(V, E)$)
▷ T is a tree that is not ULP with distinct labels.

1. Let T' be the subtree of T given by Remove−Leaves(T).
2. Let u be any vertex of degree at least 3 in T'.
3. Let v be any other vertex of degree at least 3 in T. If one does not exist, return the empty tree.
4. Let p be the unique path u to v in T, and let V_p be the vertices of p.
5. Let s and t be any neighbors of v in T that are not in V_p.
6. Let a and x be any neighbors of u in T' that are not in V_p.
7. Let b and y be any neighbors (other than u) of a and x, resp., in T.
8. Return the induced subtree of T on the vertices $\{a, b, s, t, x, y\} \cup V_p$.

Finding a path p in step 4 can be done in $O(n)$ using depth-first search starting from vertex u. Following the predecessor tree from v to u gives the path p.

To find a T_9 subdivision, it suffices to find a subtree isomorphic to T_9 since T_9 only contains one vertex of degree greater than 2. Any subdivided edges only introduce vertices of degree 2, hence, if T contains a subtree homeomorphic to T_9, it must also contain a subtree isomorphic to T_9.

We begin by removing all leaf vertices from T in order to obtain T', and repeat this procedure on T' in order to obtain T''; see Fig. 14(c). Since vertex r of degree 4 in T_9 has one leaf u at a distance of 3, two other leaf vertices b and y at a distance 2, and one other leaf w at a distance 1, T has a subtree isomorphic to T_9 if and only if (i) r is in T'', (ii) r has degree at least 3 in T', and (ii) r has degree at least 4 in T. Once we have r, we can find a subtree isomorphic to T_9 in $O(n)$ time as follows:

Find−T_9−Subdivision($T(V, E)$)
▷ T is a tree that is not ULP with distinct labels.
1. Let T' be the subtree of T given by Remove−Leaves(T).
2. Let T'' be the subtree of T' given by Remove−Leaves(T').
3. Let r be any vertex in T'' with degree at least 3 in T' and with degree at least 4 in T. If one does not exist, return the empty tree.
4. Let s be any neighbor of r in T''.
5. Let t be any neighbor of s (other than r) in T', and u be some neighbor of t (other than s) in T.
6. Let a and x be any neighbors (other than s) of r in T'.
7. Let b and y be any neighbors (other than r) of a and x, resp., in T.
8. Let w be any neighbor of r (other than a, s, and x) in T.
9. Return induced subtree of T on the vertices $\{a, b, r, s, t, u, w, x, y\}$. □

6. Conclusion and Future Work

Level planarity adds two constraints to standard planarity: First, vertices are each labeled with an integer between 1 and k, assigning it to one of k levels, where the y-coordinate of a vertex is determined by its label. Second, edges connect vertices of distinct levels and are composed of strictly y-monotone line segments.

We added the restriction that the underlying graph be level planar over all possible labelings. We termed level planar graphs that meet this final restriction unlabeled level planar (ULP). We considered two cases: distinct labels with one vertex per level, and duplicate labels with fewer levels than vertices.

This led us to consider the following questions that we have answered for trees:

(1) Which graphs are ULP with distinct labels and which are not, and why?

(2) How can these graphs always be drawn for any labeling?

(3) Can these graphs be easily recognized?

(4) Are there graphs that are also ULP for the case of duplicate labels?

We briefly summarize our results and their significance.

(1) ULP trees with distinct labels consist of caterpillars, radius-2 stars, and degree-3 spiders. Every other tree contains one of the two forbidden trees T_8 and T_9. This is akin to Kuratowski's K_5 and $K_{3,3}$ forbidden subdivisions of planar graphs.

(2) Each type of ULP tree can be drawn in linear-time and space on an integer grid for any labeling. Our algorithms produce consistent drawings in which the same graph is drawn in a similar manner for any labeling. This has the added benefit of allowing dynamic visualization in which the labelings can be permuted arbitrarily.

(3) ULP trees can be recognized by determining in linear-time if the tree contains a subtree homeomorphic to one of the forbidden trees. We have an efficient implementation of all these algorithms that dynamically determines whether a given tree is ULP, and if so, provides a compact level planar drawing. If not, an instance of one of the forbidden subtrees is highlighted. A fully functional implementation, along with movies, screen shots, and downloadable example graphs highlighting each algorithm can be found at `http://ulp.cs.arizona.edu`.

(4) Caterpillars are the only trees that are also ULP when multiple vertices can have the same label. This implies that level caterpillars are the only trees that are always level planar.

In the conference version of this paper [9], only the first question was fully addressed, while the second and third questions were only partially addressed, and the fourth question was not considered. Recently, the first two questions have been answered for general planar graphs [12], which is an extension of this work here, although the remaining two questions have yet to be addressed in the general case. The set of forbidden ULP graphs given in [12] includes the forbidden trees T_8 and T_9. The corresponding characterization for general ULP graphs relies on the correctness of the results given here for ULP trees. Moreover, the fact that neither T_8 nor T_9 is ULP is fundamental in proving the completeness of that characterization and the proofs in [12], which does not repeat the arguments given here.

In addition to generalizing all of the ULP tree results to ULP graphs, future work includes extending these results for other types of planarity, such as radial level planarity and cyclic level planarity. As ULP trees were useful for finding new MLNP tree patterns [13], ULP graphs should be useful for finding other missing level non-planar patterns [1].

References

[1] C. Bachmaier and W. Brunner. Linear time planarity testing and embedding of strongly connected cyclic level graphs. In *16th European Symposium on Algorithms, ESA 2008*. To appear in 2008.

[2] U. Brandes, P. Kenis, and D. Wagner. Communicating centrality in policy network drawings. *IEEE Transactions on Visualization and Computer Graphics*, 09(2):241–253, 2003.

[3] P. Brass, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. B. Mitchell. On simultaneous graph embedding. *Computational Geometry: Theory and Applications*, 36(2):117–130, 2007.

[4] G. Di Battista and E. Nardelli. Hierarchies and planarity theory. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(6):1035–1046, 1988.

[5] V. Dujmović, M. R. Fellows, M. T. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. A. Rosamond, M. Suderman, S. Whitesides, and D. R. Wood. On the parameterized complexity of layered graph drawing. In *9th European Symposium on Algorithms, ESA 2001*, volume 2161 of *LNCS*, pages 488–499. Springer, 2001.

[6] C. Duncan, D. Eppstein, and S. G. Kobourov. The geometric thickness of low degree graphs. In *20th ACM Symposium on Computational Geometry, SoCG 2004*, pages 340–346, 2004.

[7] P. Eades, Q. Feng, X. Lin, and H. Nagamochi. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. *Algorithmica*, 44(1):1–32, 2006.

[8] D. Eppstein. Separating thickness from geometric thickness. In *10th Symposium on Graph Drawing, GD 2002*, volume 2528 of *LNCS*, pages 150–161. Springer, 2003.

[9] A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. Characterization of unlabeled level planar trees. In *14th Symposium on Graph Drawing, GD 2006*, volume 4372 of *LNCS*, pages 367–379. Springer, 2007.

[10] A. Estrella-Balderrama, E. Gassner, M. Jünger, M. Percan, M. Schaefer, and M. Schulz. Simultaneous geometric graph embeddings. In *15th Symposium on Graph Drawing, GD 2007*, volume 4875 of *LNCS*, pages 280–290. Springer, 2008.

[11] I. Fáry. On straight line representation of planar graphs. *Acta Scientiarum Mathematicarum*, 11:229–233, 1948.

[12] J. J. Fowler and S. G. Kobourov. Characterization of unlabeled planar graphs. In *15th Symposium on Graph Drawing, GD 2007*, volume 4875 of *LNCS*, pages 37–49. Springer, 2008.

[13] J. J. Fowler and S. G. Kobourov. Minimum level nonplanar patterns for trees. In *15th Symposium on Graph Drawing, GD 2007*, volume 4875 of *LNCS*, pages 69–75. Springer, 2008.

[14] M. Harrigan and P. Healy. Practical level planarity testing and layout with embedding constraints. In *15th Symposium on Graph Drawing, GD 2007*, volume 4875 of *LNCS*, pages 62–68. Springer, 2008.

[15] P. Healy and A. Kuusik. The vertex-exchange graph: A new concept for multi-level crossing minimisation. In *7th Symposium on Graph Drawing, GD 1999*, volume 1731 of *LNCS*, pages 205–216. Springer, 2000.
[16] P. Healy, A. Kuusik, and S. Leipert. A characterization of level planar graphs. *Discrete Mathematics*, 280(1-3):51–63, 2004.
[17] L. S. Heath and S. V. Pemmaraju. Recognizing leveled-planar dags in linear time. In *3rd Symposium on Graph Drawing, GD 1995*, volume 1027 of *LNCS*, pages 300–311. Springer, 1996.
[18] L. S. Heath and S. V. Pemmaraju. Stack and queue layouts of directed acyclic graphs. II. *SIAM Journal on Computing*, 28(5):1588–1626 (electronic), 1999.
[19] L. S. Heath and A. L. Rosenberg. Laying out graphs using queues. *SIAM Journal on Computing*, 21(5):927–958, 1992.
[20] M. Jünger and S. Leipert. Level planar embedding in linear time. *Journal of Graph Algorithms and Applications*, 6(1):67–113, 2002.
[21] M. Jünger, S. Leipert, and P. Mutzel. Pitfalls of using PQ-trees in automatic graph drawing. In *4th Symposium on Graph Drawing, GD 1996*, volume 1190 of *LNCS*, pages 193–204. Springer, 1997.
[22] M. Kaufmann, I. Vrťo, and M. Geyer. Two trees which are self-intersecting when drawn simultaneously. In *13th Symposium on Graph Drawing, GD 2005*, volume 3843 of *LNCS*, pages 201–210. Springer, 2006.
[23] C. Kuratowski. Sur les problèmes des courbes gauches en Topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.
[24] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.