Unlabeled Level Planarity

by

Joe Fowler

A Dissertation Submitted to the Faculty of the

Department Of Computer Science

In Partial Fulfillment of the Requirements
For the Degree of

Doctor of Philosophy

In the Graduate College

The University of Arizona

2009

The University of Arizona
Graduate College

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Joe Fowler
entitled Unlabeled Level Planarity
and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Date: 13 April 2009
Stephen G. Kobourov

Date: 13 April 2009
Peter Downey

Date: 13 April 2009
Alon Efrat

Date: 13 April 2009
John D. Kececioglu

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.
I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Date: 13 April 2009
Dissertation Director: Stephen G. Kobourov

Statement by Author

Acknowledgements

First, I gratefully acknowledge all the support and good advise given to me by my dissertation director Stephen Kobourov. Second, I sincerely acknowledge all the aid given to me by my fellow PhD student Alexander Estrella-Balderrama, who was the first to determine how to always draw a degree-3 spider on tracks and who has tirelessly implemented all of the ULP tree algorithms in the tool GraphSET [39], which has been an invaluable tool indeed. Finally, I wish to acknowledge Justin Cappos, who wrote the initial version of the paper regarding the "tree on tracks" problem [38]. It was this problem that initially grabbed my interest and eventually drew me to coin the term "unlabeled level planarity".

Dedication

I dedicate this work to my parents Ardeth Joye and Bruce Adelbert Fowler who were never entirely certain that they would eventually live to see the day that I would finally graduate.

Table of Contents

Table of Contents – *Continued*

List of Figures

LIST OF FIGURES – *Continued*

List of Figures – *Continued*

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 partitions V into k *levels*. In a *level drawing*, the y-coordinate of each vertex matches its label and the edges are drawn strictly y-monotone. This leads to level drawings in the xy-plane where all vertices with label j lie along the line $\ell_j = \{(x, j) : x \in \mathbb{R}\}$ and where each edge crosses any of the k horizontal lines ℓ_j for $j \in [1..k]$ at most once. A graph with such a labeling forms a *level graph* and is *level planar* if it has a level drawing without crossings.

We first consider the class of level trees that are level planar regardless of their labeling. We call such trees *unlabeled level planar* (ULP). We describe which trees are ULP and provide linear-time level planar drawing algorithms for any labeling. We characterize ULP trees in terms of two forbidden subdivisions so that any other tree must contain a subtree homeomorphic to one of these. We also provide linear-time recognition algorithms for ULP trees. We then extend this characterization to all ULP graphs with five additional forbidden subdivisions, and provide linear-time recognition and drawing algorithms for any given labeling.

Chapter 1

Introduction

The advent of workstations capable of graphical display in 1960s combined with the propensity of computer scientists and engineers to visualize and arrange their code in terms of diagrams has lead to the genesis of automated graph drawing [9, 22, 23, 79]. Here a graph is an abstract mathematical representation of a set of objects (the vertices of the graph) and their interrelationships (the edges of the graph) that connect pairs of vertices. Drawings are 2D visualizations of graphs such that the nodes in the drawing correspond to vertices and links connecting nodes correspond to the edges. The primary focus of the field graph drawing is addressing the problem of designing algorithms to automatically assign locations to vertices and route edges of graphs while appealing to some aesthetic criteria [75].

One of the primary techniques used in representing diagrams in a top-down, layered fashion are hierarchical layouts [14, 17, 76, 80]. Hierarchical relationships exist in many natural contexts including bioinformatics [55], mathematical taxonomies [56], social networks [10], software engineering [6, 25, 47], VLSI circuits [1], and computer networks [82]. The horizontal layers in the hierarchical layout partition the vertices of the graph into *levels*, where the layering of vertices into levels is also known as a *leveling*. A graph combined with a leveling that maps its vertices to a set of levels forms a *level graph*. Fixing the y-coordinate of each vertex (given by the vertical position of the layer of the drawing) and insisting that all the edges are drawn strictly downwards forms a *level drawing* of the level graph. This is equivalent to restricting the vertices of the graph to lie along sets of horizontal lines, called *tracks*, where each edge crosses a given track at most once.

When visualizing these complex interrelationships, edge crossings are often undesirable, and minimizing the number of crossings is a common aesthetic criterion [74]. A graph is considered *planar* if it can be drawn in the xy-plane without crossings, which leads to the mathematical concept of planarity. Planar graphs have been studied in terms of their relationships with their respective planar drawings [68, 66]. A *geometric graph* must be drawn with straight-line edges, whereas, a *topological graph* can be drawn with edges that curve or bend arbitrarily. One famous result by Fáry is that any topological graph that has a planar drawing also has a geometric planar drawing [42]. Another celebrated result by Pach and Wenger is that any topological planar graph has a planar drawing with fixed vertex locations [73]. Hence, planarity is independent of how the vertices are placed in the plane. Likewise, planarity is also independent of how the edges are drawn in the plane so long as they do not cross.

A much more restrictive form of planarity results when attempting to find a planar realization of a level graph. A graph is considered *level planar* if it has a level drawing without crossings. Level planarity has been well studied [58, 59, 61, 69, 72] and also has an analogous result with respect to geometric versus topological graphs. Any level graph that has a level planar drawing in which the edges can bend or curve (while remaining strictly y-monotone) also has a straight-line level planar drawing though it may require exponential space as shown by Eades *et al.* [29, 30]. In order to distinguish level planarity from standard planarly both of the restrictions imposed by a level drawing are necessary: fixing the y-coordinates of the vertices *and* drawing the edges as strictly y-monotone curves or straight-lines. Either restriction in and of itself would be insufficient to force a crossing in a graph that is planar but level non-planar for a given leveling.

Level graphs are historically defined in terms of directed graphs (graphs with oriented edges, i.e., edges drawn with arrows pointing in one of two directions) since edges in a hierarchical layout are typically drawn downwards. This is because upper layers contains objects higher in the hierarchy than lower layers. More precisely, a leveling is not just an assignment of vertices to levels, but is an assignment that respects the orientation of the edge. Vertices are assigned to levels such that no edge is directed upward. However, once a graph has had its vertices assigned to levels, the planarity of the level graph is independent of edge orientation. Vertices can only be moved back and forth in the x-direction, which does not alter orientation of edges. We can then consider the level planarity of the underlying undirected graph by removing all edge orientation. Given that edges are now without orientation, the assignment is no longer a true leveling, but rather a *labeling*, where each vertex has a label that corresponds to its level. The only restriction is that two adjacent vertices (vertices connected by an edge) cannot have the same label given that an edge between two vertices on the same track cannot be drawn strictly y-monotone.

For simplicity, we use integer coordinates whenever possible when drawing a level graph, and use labels that match the y-coordinates of the vertices. This gives an enumeration of the levels starting from the lowermost level having a label of 1 to the uppermost level having a label of k (the number of levels), which is at most n (the number of vertices in the graph) since each level has at least one vertex.

Determining whether a given undirected graph is level planar for a fixed number of 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 [54]. If $n = k$ (i.e., one vertex per level), then a level planar labeling is easily obtained. Any straight-line planar drawing of the graph can be rotated until the vertices have distinct y-coordinates, the order of which gives the desired labeling.

We call a level graph that is level planar over all possible labelings an *unlabeled level planar* (ULP) graph. The basis of this work is to elucidate the properties of unlabeled level planarity. An ULP graph is not bound to any one particular labeling, and hence, we can consider an ULP graph purely in terms of the structure of the graph, irrespective of any labeling. This allows us to characterize ULP graphs in terms of forbidden obstructions.

We start by focusing on trees (connected acyclic graphs) and characterizing ULP trees with distinct labels in terms of two forbidden subtrees T_8 and T_9; see Fig. 1.1. Later we generalize this characterization to include all ULP graphs with distinct labels with the addition of five forbidden graphs. In both cases, we provide

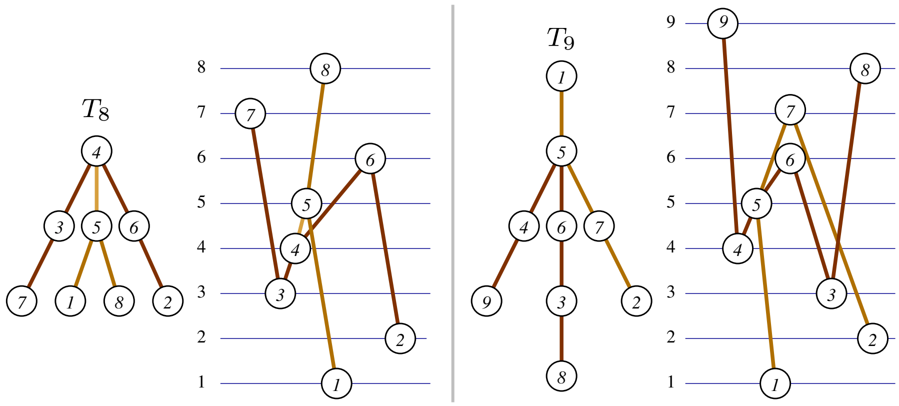


Figure 1.1: ULP forbidden trees T_8 and T_9 and their level non-planar labelings

linear-time recognition and drawing algorithms for any labeling. We also consider the case of allowing duplicate labels, i.e., more vertices than levels. This simplifies the characterizations in that there are fewer forbidden graphs, but the drawing algorithms become more complex.

The study of unlabeled level planarity has two primary motivations. The first motivation discussed in section 1.1 is how to generalize forbidden ULP graphs into level planar obstructions that can allow for a full characterization of level planarity and how to potentially use these obstructions in order to obtain better hierarchical layouts. The second motivation given in section 1.2 has to do with the simultaneous drawing of multiple graphs. Afterward, in section 1.3 we discuss related previous work, and then state our contributions in more detail in section 1.4. Section 1.5 concludes this introduction, with an overview of the remaining chapters.

1.1 Level Planarity Motivation

The ULP characterization is akin to Kuratowski's Theorem for planar graphs, which states that a graph is planar if and only if it does not contain a subdivision of K_5 or $K_{3,3}$ [68]; see Fig. 1.2. Here a subdivision of a graph is the operation of replacing single edges with chains of vertices of arbitrary length. No matter how the vertices of a K_5 or a $K_{3,3}$ subdivision are moved about in the plane, a crossing will always result. Similarly, no matter how the vertices of T_8 or T_9 are moved back and forth

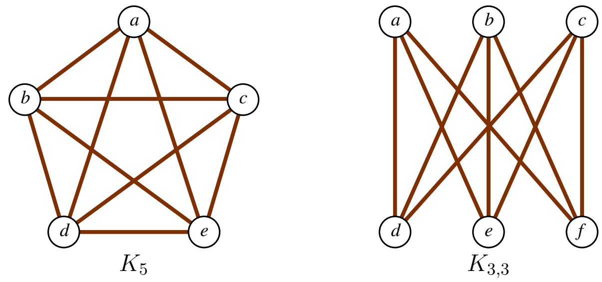


Figure 1.2: The non-planar Kuratowksi forbidden graphs K_5 and $K_{3,3}$

along each level, a crossing will also result. The Kuratowksi graphs K_5 and $K_{3,3}$ form minimal obstructions to planarity in that the removal of an edge results in a planar graph. The trees T_8 and T_9 are also edge minimal. Moreover, their forbidden labelings can be generalized into *minimal level non-planar* (MLNP) patterns, which are minimal obstructions to level planarity. One consequence of characterizing ULP trees was the discovery of additional MLNP patterns not previously unknown [44]; see Fig. 1.3.

Any directed acyclic graph (DAG) has a hierarchical representation given by a topological sort [63]. The most common method used to draw any DAG was given by Sugiyama *et al.* [80]. The first step is to assign nodes to layers, and the second step is to find an embedding (an ordering of nodes among layers) in which the number of crossings is small. While there exists good heuristics as well as exact methods based upon integer linear programs (ILPs) to find crossing minimal embeddings for the second step [57, 62, 71], most of the methods used to find layerings for the first step are based upon greedy local optimization and fail to find a global solution [19, 32, 33].

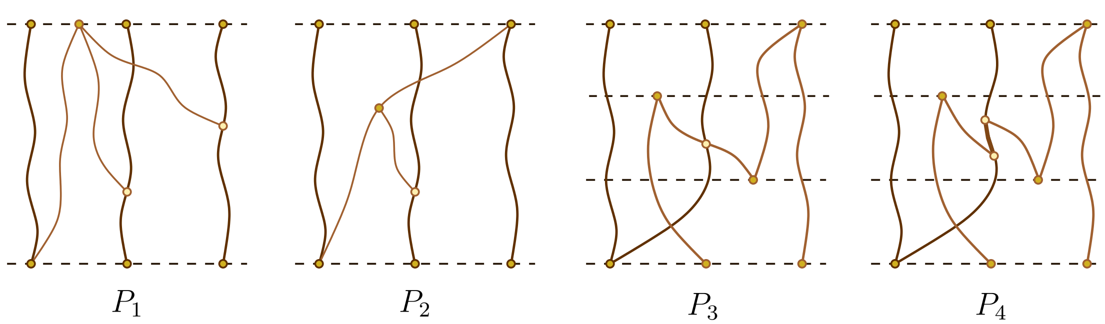


Figure 1.3: Four MLNP patterns for level non-planar trees based upon T_8 and T_9

For example, one standard method is to use a longest path layering in which the layer assigned is the distance of the longest path from a source [77]. This ensures that the minimal number of layers are used, but there is no guarantee that this will result in the fewest number of crossings after the second step is complete.

Sugiyama's algorithm is used in various commercial applications such as Rational Rose [21] that create layouts for diagrams of the Unified Modeling Language (UML) of Booth *et al.* [7]. UML diagrams have the feature that not all components are strictly hierarchical, which allows for further flexibility in assigning levels in mixed-hierarchical drawing algorithms. These generally are extensions of Sugiyama's algorithm and are also based upon simple greedy heuristics [34, 78], and can yield unreadable diagrams as a result [47]. Until recently, the only known heuristics were for crossing minimization [48]. However, exact methods have since been proposed and implemented based upon ILPs with different branch-and-cut approaches [16, 15, 18, 20].

As noted above, ILPs have been formulated both for the second step of Sugiyama's algorithm (finding a crossing minimal embedding of edges along a level) and for the standard crossing minimization problem. While the running times of these ILP algorithms can be exponential in theory, in practice they can be quite efficient for small and medium-size graphs with less than 100 vertices [18]. However, no ILP has been formulated for the first step of the Sugiyama's algorithm to allow for a level drawing in which the crossings can be minimized for a given number of levels. Instead, only relatively simple heuristics have been employed to generate layerings of nodes.

Given the success of the approach of using ILPs with branch-and-cut to find exact solutions for the other problems, it is natural to ask whether formulating an ILP is possible, and if so, is it a viable approach for the first step of Sugiyama's algorithm? In order to devise the constraints of such an ILP, having a better understanding of the underlying obstructions to level planarity is an essential precursor. The study of the unlabeled level planarity is a step in this direction as evidenced by the discovery of additional MLNP patterns based upon forbidden ULP graphs.

1.2 Simultaneous Embedding Motivation

Visualization of multiple graphs simultaneously has many real world applications such as multi-user interactive displays [26, 67], pivot partitioning in DNA probe array layouts in bioinformatics [64, 65], and social network exploration [11]. When visually examining spatially related information, viewers construct an internal model known as the mental map [31, 70]. When relating multiple diagrams, where each diagram has nodes that correspond to the same set of objects, the mental map allows the viewer to recognize the interrelationships between the diagrams, and hence, between objects that are not denoted by edges in any one diagram. Simultaneous embedding, first proposed by Brass *et al.* [12, 13], aids in this visualization by placing vertices common in each graph at the same coordinates when drawing the graphs simultaneously in the plane. Here the common vertices are given the same label which then stipulates a mapping between the graphs.

In this way, simultaneous embedding is a generalization of traditional planarity, where one looks for a common embedding in the plane for multiple graphs defined on the same vertex set such that the embedding is planar for any one graph. However, the union of the graphs may be non-planar. If there are no restrictions on how the edges are drawn, then any number of graphs share a simultaneous embedding [13, 73]. However, if only straight-line edges are allowed, as in the case of simultaneous geometric embedding (SGE), then this is no longer true. Simultaneous embedding is distinctly dissimilar from standard planarity, where geometric planarity and topological planarity are equivalent notions [42]. Simultaneous geometric embedding is related to the geometric thickness of a graph, which is the fewest number of planar geometric graphs whose union forms the graph in question [35]. This has applications in VLSI chip design [28].

Relatively little is known about which graphs share a SGE. In general, determining for a given pair of graphs whether they admit a SGE is NP-hard [41]. However, it has been shown that certain pairs of n-vertex planar graphs, such as pairs of paths, pairs of cycles, and pairs of caterpillars, always admit a SGE in which each vertex

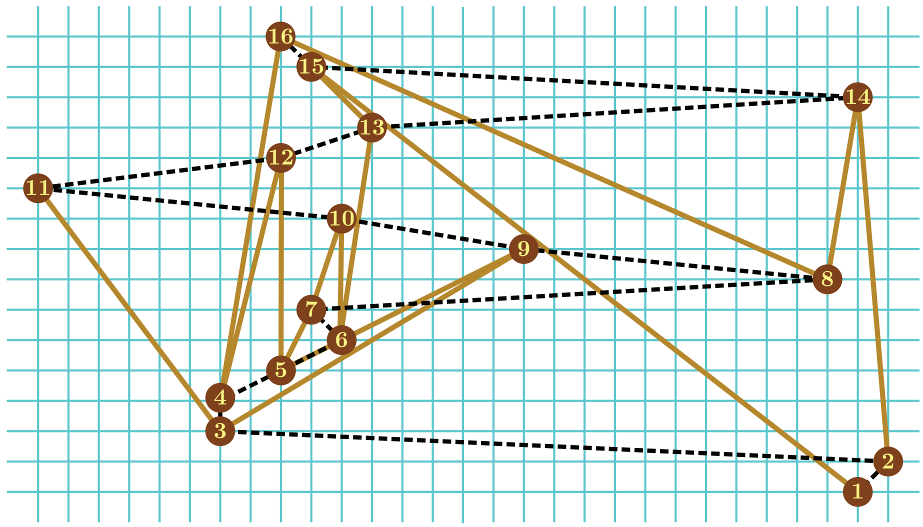


Figure 1.4: Example of simultaneously embedding a monotone (dashed) path with a tree

is given a distinct label between 1 and n [13]. Unfortunately, the pairs of classes of graphs for which this possible is fairly restrictive. There exist pairs of outerplanar graphs, triples of paths, even (planar graph, path) pairs that do not allow a SGE for particular labelings [13, 36, 45]. More recently, it was shown that there exists pairs of trees that do not always allow for a SGE [46]. However, it remains open whether a path and a tree can always be drawn simultaneously using only straight-line edges for any distinct labeling.

When simultaneously embedding a path with an arbitrary graph, 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; see Fig. 1.4. If the graph is level planar for this labeling, then any such level planar drawing with bends can be redrawn in $O(n)$ time without bends [30]. This allows for a SGE 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 graphs with one vertex per level is precisely the set of graphs that have a SGE with every monotone path.

1.3 Related Previous Work

Jünger, Leipert, and Mutzel [58, 59, 61, 69] provide linear-time recognition and embedding algorithms for level planar graphs. Here the embedding is the left-to-right ordering of the edge intersections with each track. These algorithms use the PQ-tree data structure of Booth and Lueker [8]. The JLM algorithm corrects a previous PQ-tree algorithm to test level planarity by Heath and Pemmaraju [52, 53]. Di Battista and Nardelli [24] 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.* [30] 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.* [27] provide a linear-time level planarity testing algorithm using fixed parameter tractability. Healy and Kuusik [50] 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 [49] 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 [60].

Further, Di Battista and Nardelli provide a set of *level non-planar* (LNP) patterns [24] that fully characterize level planar hierarchies. However, the level non-planar subgraphs these patterns match are not necessarily edge minimal. Healy *et al.* [51] extend the LNP patterns for hierarchies to provide a set of *minimal level non-planar* (MLNP) patterns in order to characterize all level planar graphs. However, these patterns are specific to a given labeling and are not based solely upon the underlying graph. This is unlike the ULP characterization that is independent of any labeling and only relies on the structure of the graph in question. The set of MLNP patterns have been shown to be incomplete. Two new MLNP tree patterns were given in [44] based upon the forbidden ULP tree T_9; see Figs. 1.1 and 1.3. This has reopened the problem of determining all MLNP patterns.

1.4 Our Contribution

We first characterize ULP trees for the case of one vertex per level (distinct labels) and then for the case of more vertices than levels (duplicate labels). Here our contributions are four-fold.

1. First, we describe the set of unlabeled level planar (ULP) trees as consisting of (i) *caterpillars* (trees in which the removal of all leaves yields a path), (ii) *radius*-2 *stars* (any number of paths of length 1 or 2 with a common endpoint), or (iii) *degree*-3 *spiders* (three paths with a common endpoint); see Fig. 1.5. We note that (ii) and (iii) are only ULP with distinct labels.

2. Second, for each ULP tree, we provide $O(|V|)$-time level planar drawing algorithms on integer grids for any labeling.

3. Third, we characterize ULP trees with one vertex per level in terms of two minimal forbidden graphs, T_8 and T_9; see Fig. 1.5. If multiple vertices per level are permitted, the forbidden graph T_7 characterizes ULP trees.

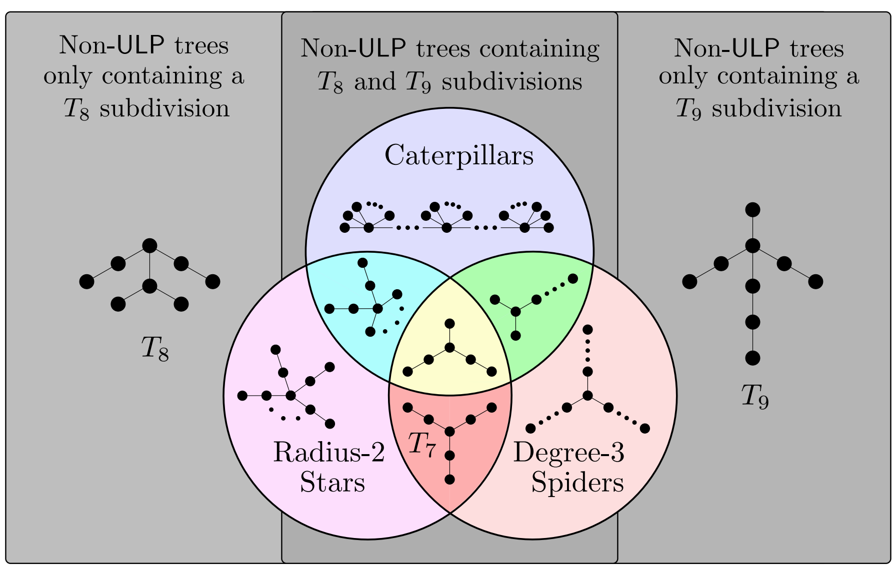


Figure 1.5: Venn diagram of the universe of trees partitioned into the trees containing a subtree homeomorphic to T_8 or T_9 (the gray rectangles minus the circles) and the set of unlabeled level planar (ULP) trees with one vertex per level, which are caterpillars, radius-2 stars, and degree-3 spiders.

4. Finally, we also provide $O(|V|)$-time recognition algorithms 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.

We next characterize ULP graphs for the case of one vertex per level, and then for the case of more vertices than levels. Here our contributions are three-fold.

1. First, we describe the set of ULP graphs with distinct labels as consisting of (i) *generalized caterpillars*, (ii) *radius*-2 *stars*, (iii) *extended* 3-*spiders*, and (iv) *extended* K_4 *subgraphs*; see Fig. 1.6. If duplicate labels are allowed, then only K_3-*caterpillar*s and graphs isomorphic to G_ω are ULP.

2. Second, for each type of ULP graph, we provide $O(|V|)$-time level planar drawing algorithms on integer grids for any labeling; see Fig. 1.7.

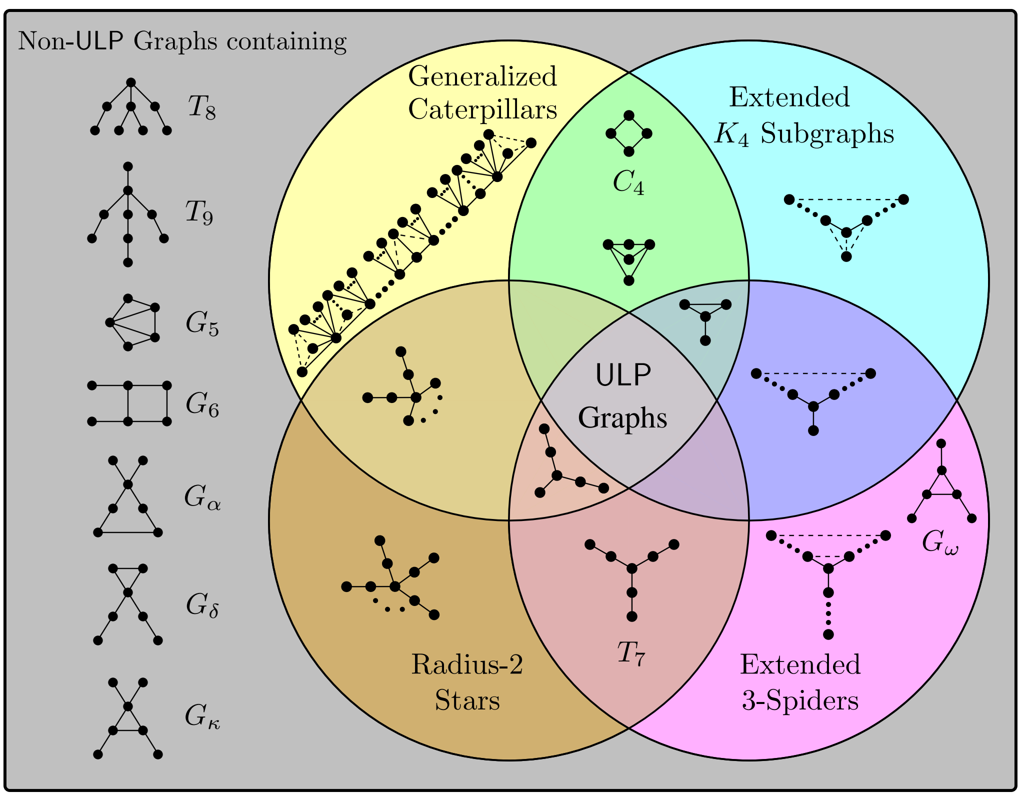
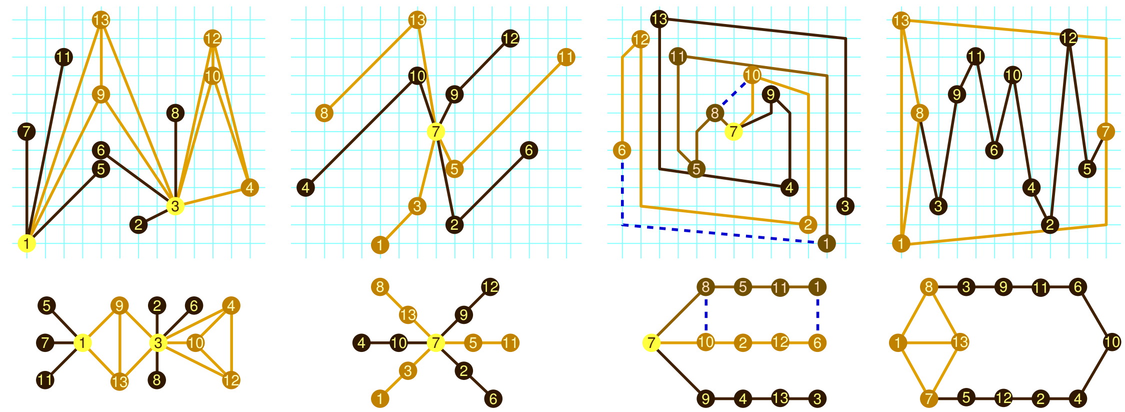


Figure 1.6: Venn diagram of the set of ULP graphs with distinct labels characterized by seven forbidden graphs. Graphs that do not contain any of these forbidden graphs are radius-2 stars, generalized caterpillars, extended 3-spiders, and extended K_4 subgraphs.



(a) (b) (c) (d)

Figure 1.7: Level planar drawings of a generalized caterpillar in (a), a radius-2 star in (b), an extended 3-spider in (c), and an extended K_4 in (d).

3. Third, we characterize ULP graphs with distinct labels in terms of the two forbidden ULP trees T_8 and T_9 and five additional forbidden ULP graphs, G_5, G_6, G_α, G_δ, and G_κ; see Fig. 1.6. For the case of duplicate labels, we show that T_7, C_4, and G_κ form the set of forbidden ULP graphs.

4. Finally, we provide $O(|V|)$-time recognition algorithms for ULP graphs.

1.5 Chapter Overviews

Chapter 2 lays the groundwork for subsequent chapters. The chapter begins with a formal introduction to simultaneous embedding and level planarity, follows with the graph terminology used throughout this work, and ends by carefully defining the various classes of graphs related to unlabeled level planarity.

Chapter 3 shows how to efficiently determine in linear-time whether a given graph is ULP. While recognizing ULP trees is easily done, recognizing ULP blocks, the "building blocks" for ULP graphs, require more detailed algorithms. This leads to a comprehensive set of recognition algorithms that can determine whether or not a graph is ULP with distinct or duplicate labels in linear time.

Chapter 4 proves that the proposed classes of ULP trees each have a level planar drawing for any given labeling. The chapter starts by showing how to draw caterpillars for distinct labels, which is the easiest of all the tree drawing algorithms. Level drawings of radius-2 stars are more involved, but still easily done. However, finding nice level drawings for degree-3 spiders requires more effort, and several algorithms are needed in order to produce the final level planar drawings. The chapter concludes with the drawing algorithm for caterpillars with duplicate labels, which is significantly more challenging than drawing caterpillars with distinct labels.

Chapter 5 gives the full characterization for ULP trees. First, the labelings of T_8 and T_9 given in Fig. 1.1 are shown to be level non-planar and minimal. The universe of trees are then proved to be partitioned between those that are ULP and those that contain one of the forbidden ULP trees. The more difficult case of distinct labels involving the two forbidden trees T_8 and T_9 is handled first, followed by the easier case of duplicate labels where there is only the one forbidden tree T_7.

Chapter 6 uses results from chapter 5 to show how to certify whether a tree is ULP, by finding a forbidden subdivision (if it exists) in the given tree in linear time.

Chapter 7 shows how to draw an ULP graph for any labeling. For the case of distinct labels, there are three new classes of graphs not previously covered in chapter 4. Drawing the class of generalized caterpillars requires the recognition algorithms for ULP blocks given in chapter 3 and a careful examination of their definitions from chapter 2. Drawing the class of extended 3-spiders involves extensive modification of the degree-3 spider drawing algorithms from chapter 4 in order to accommodate additional edges. Drawing the class of biconnected extended K_4 subgraphs is relatively straightforward. For the case of duplicate labels, there are two distinct classes of graphs. Drawing the class of K_3-caterpillars depends heavily on the caterpillar drawing algorithms with duplicate labels from chapter 4, though there are several non-trivial differences. Drawing the class of graphs isomorphic to G_ω (see Fig. 1.6) is the simplest of all the drawing algorithms.

Chapter 8 describes the complete characterization of ULP graphs. All of the forbidden ULP graphs are first shown to possess labelings that force a crossing when drawn as level graphs, and are then shown to be minimal in that the removal of an edge results in one or more ULP graphs. Generalized caterpillars have their own forbidden characterization, which forms a basis in proving the complete characterization for all ULP graphs with distinct labels. The chapter ends by providing the corresponding characterization for duplicate labels, which depends first on characterizing K_3-caterpillars before including graphs that are isomorphic to G_ω.

Chapter 9 concludes with an assessment of our results and describes open problems that remain for unlabeled level planarity and other related future work.

Chapter 2

Preliminaries

We begin this chapter with a formal introduction to simultaneous embedding and level planarity. Afterward, we follow with graph theoretical terminology and notation. We end with definitions of graph classes related to unlabeled level planarity.

2.1 Simultaneous Embedding and Level Planarity

Two planar n-vertex graphs $G_1(V, E_1)$ and $G_2(V, E_2)$ have a *simultaneous embedding with mapping* if they can be drawn in the xy-plane with a bijection $f : V \mapsto V$ such that v and $f(v)$ have the same xy-coordinates and the planarity of each graph is maintained. If this can be done for some bijection f, then G_1 and G_2 are *simultaneously embeddable*. If edges of both E_1 and E_2 are drawn with straight-line edges, then G_1 and G_2 have a *simultaneous geometric embedding* (SGE).

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 numbered levels to vertices of higher numbered 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 *labeling* $\phi : V \rightarrow [1..k]$ such that $\phi(u) \neq \phi(v)$ (rather than a *leveling* in the case of directed graphs where $\phi(u) < \phi(v)$) for every edge $(u, v) \in E$.

The labeling 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 $V_j = \phi^{-1}(j)$. Hence, $\phi(v)$, the label of v, determines its corresponding level. When ϕ is an injection, each level contains at most one vertex. Moreover, when there is one vertex per level, $k = n$ in which case ϕ has *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* ℓ_j, the horizontal line $\{(x, j) \mid x \in \mathbb{R}\}$, and (ii) the edges can be drawn as strictly y-monotone polylines, where a polyline is a connected sequence of line segments. *Edge bends* are the endpoints of segments that do not correspond to the endpoints of any edge. Edge bends naturally occur at any point an edge intersects a track, where 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 *realized* with a level planar drawing, which forms a *realization* of G. Any realization with bends can be "stretched out" in the x-direction to form a straight-line realization in $O(n)$ time as shown by Eades *et al.*. [30]. However, the area of the realization may become exponential.

If G is level planar for any labeling, then G is considered to be *unlabeled level planar* (ULP). When simultaneously embedding an ULP graph with a path, the order of the vertices along the path determine a labeling ϕ. The ULP graph can then be drawn simultaneously with the path zig-zagging downward in strict y-monotone fashion; see Fig. 1.4.

A *chain* C is a simple path denoted v_1–v_2– $\cdots$ –v_t. A vertex v of C is *ϕ-minimal* (or *ϕ-maximal*) if it has a minimal (or maximal) track number of all the vertices of C. Such a vertex is *ϕ-extreme* if it is ϕ-minimal or ϕ-maximal.

2.2 Graph Notation and Terminology

For graph $G(V, E)$, the vertex set V and edge set E can also be written $V(G)$ and $E(G)$, respectively. The *order* $n(G)$ of a graph G is number of vertices. The *length* of a path p or a cycle C, written $|p|$ and $|C|$ respectively, is the number of edges in p or C. The path and the cycle of order n are denoted P_n and C_n, and have lengths $|P_n| = n - 1$ and $|C_n| = n$, respectively. The *path length* and *cycle length* of a graph are the lengths of the longest path and cycle, respectively.

The *neighborhood* $N(v)$ of vertex v is the set of adjacent vertices, the *neighbors*, of v. The *closed neighborhood* $N[v]$ of v includes v, i.e., $N[v] = N(v) \cup \{v\}$. The *degree* of v, written $deg(v)$, is given by $deg(v) = |N(v)|$. For a graph $G(V, E)$, the maximum and minimum degrees of G are written $\Delta(G)$ and $\delta(G)$, respectively.

In a graph $G(V, E)$, *subdividing* an edge $(u, v) \in E$ replaces edge (u, v) with the pair of edges (u, w) and (w, v) in E so that w is added V. A *subdivision* of G is obtained through a series of edge subdivisions. A graph $G(V, E)$ is *isomorphic* to a graph $\tilde{G}(\tilde{V}, \tilde{E})$ if there exists a bijection $f : V \to \tilde{V}$ such that $(u, v) \in E$ if and only if $\big(f(u), f(v)\big) \in \tilde{E}$. A graph $G(V, E)$ is *homeomorphic* to a graph $\tilde{G}(\tilde{V}, \tilde{E})$ if the subdivisions of G and $\tilde{G}$ are isomorphic. A graph H is a *forbidden subdivision*, or more simply, a *forbidden graph* of a class of graphs $\mathcal{G}$ if for every graph $G \in \mathcal{G}$, G does not contain a subgraph homeomorphic to H.

The union and the intersection of $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, are denoted by $G_1 \cup G_2$ and $G_1 \cap G_2$, respectively. The union $G_1 \cup G_2$ is *disjoint* provided $V_1 \cap V_2 = \varnothing$. The *graph join* of G_1 and G_2, written $G_1 \vee G_2$, is their disjoint union with all the edges that connect V_1 to V_2, i.e., the edges $\{(v_1, v_2) | v_1 \in V_1, v_2 \in V_2\}$. The *induced subgraph* of G on subset $V' \subseteq V$, written $G[V']$, is the subgraph $G'\big(V', E \cap (V' \times V')\big)$. The *complement* of a graph G is $\overline{G}$ such that $G \cup \overline{G} = K_n$ and $G \cap \overline{G} = \overline{K_n}$, which is the empty graph on n vertices. The *complete k-partite graph* $K_{n_1, n_2, \ldots, n_k}$ is $\overline{K_{n_1}} \vee \overline{K_{n_2}} \vee \cdots \vee \overline{K_{n_k}}$.

A *vertex cut* is a vertex subset that disconnects a graph into more than one component when removed. The *connectivity* of a graph of order n is either the size of the smallest vertex cut, if one exists, or $n - 1$, if not. A graph G is *k-connected* if the connectivity of G is at least k. A graph with connectivity 1 has a *cut-vertex* whose removal disconnects the graph.

A *component* of G is a maximal 1-connected subgraph. A *biconnected component* or a *block* of G is a maximal 2-connected subgraph. A *trivial block* is an isolated edge (a P_2), while a *non-trivial block* contains a cycle.

A *cyclic graph* contains a cycle, whereas, a *tree* is connected acyclic graph. A *pendant vertex* v is an endpoint of G, i.e., has degree 1, which has a single incident *pendant edge*. A pendant vertex of a tree is a *leaf*, whose incident pendent edge is a *leaf edge*. Non-pendant vertices and edges are *internal*.

For graph G with subgraph H, $G - H$ is the graph where all the edges of H and all the vertices of H *that only have neighbors in* H have been removed from G. For graphs G_1 and G_2, $G_1 + G_2$ denotes $G_1 \cup G_2$ and *implies that* $|V(G_1) \cap V(G_2)| = 1$. For edge e and graph H where $E(H) = \{e\}$, $G - e$ denotes $G - H$ where $e \in E(G)$, and $G + e$ denotes $G + H$ where $e \notin E(G)$ and *implies that* e is a pendant edge in the graph $G + e$.

The *distance* between vertices u and v is the length of the shortest path from u to v. The *eccentricity* of a vertex v, denoted $ecc(v)$, is the greatest distance to any other vertex. The *radius* of a graph is the minimum eccentricity of any vertex.

2.3 Classes of Graphs

Removal of all leaves from tree T gives *spine* S. A *caterpillar* is a tree T such that its spine is a path S; see Fig. 2.1(a), whereas, a *lobster* is a tree T such that its spine is a caterpillar S that is not also a path. A *claw* is a $K_{1,3}$, and a *star* is a $K_{1,k}$ for some $k \geq 1$; see Fig. 2.1(c), (d). A *spider* is an arbitrarily subdivided star, whose

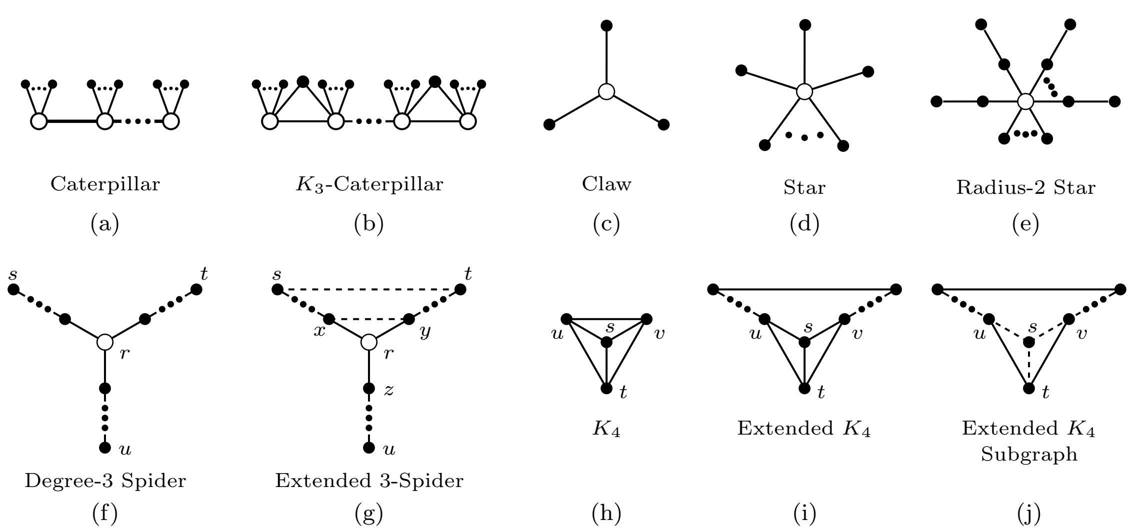


Figure 2.1: Various classes of graphs related to unlabeled level planarity in which the dashed edges are optional and the white vertices are cut vertices.

root is a vertex of maximum degree. A trivial root has degree of 1 or 2. A *leg* is a path from a root to an endpoint. A *radius*-2 *star* (R2S) is a spider with a non-trivial root that has radius 2; see Fig. 2.1(e). The *degree* of a spider is the degree of its root, e.g., a *degree*-3 *spider*, which is an arbitrarily subdivided claw; see Fig. 2.1(f).

Definition 2.3.1. *An* extended 3-spider *(E3S) is formed from a degree-3 spider by adding two optional edges: one edge connecting two of the leaves s and t, and the other edge connecting two of the neighbors x and y of the root r; see Fig. 2.1(g).*

Definition 2.3.2. *An* extended K_4 *is formed by arbitrarily subdividing the edge (u, v) of the complete graph K_4 on the vertices $\{s, t, u, v\}$; see Fig. 2.1(h), (i). An* extended K_4 subgraph *(EK4) is a connected subgraph of an extended K_4 in which the edges (u, s), (v, s), and (s, t) are optional; see Fig. 2.1(j).*

A *connector* of a block H of G is any vertex in H that is a cut-vertex in G. A k-*block* is a block H with k connectors. Hence, a 0-block is a biconnected graph.

We wish to "generalize" a class of trees into a corresponding superclass by substituting edges for 1-blocks and 2-blocks. Let G be a connected graph, and let H be a 2-connected graph. *Block substitution* of H for an internal edge (u, v) in G where $V(G) \cap V(H) = \{u, v\}$ gives G', defined to be $G' := (G - (u, v)) + H$, so that H forms a 2-block in G' where u and v are connectors of H to $G - (u, v)$. Block substitution of H for a leaf edge (u, ℓ) in $E(G)$ results in the graph $(G - \ell) + H$ where $V(G) \cap V(H) = \{u\}$ so that H is a 1-block in $(G - \ell) + H$ and u is the connector of H to $G - \ell$.

For a tree T with spine S, a *joining block* can be substituted an edge (u, v) of S and an *ending block* can be substituted for a pendant edge (u, ℓ) incident to an endpoint u of S. If T is a star, then an ending block also can be substituted for a pendant edge incident to the root u of T. A joining block is a 2-block with connectors u and v; an ending block is a 1-block with connector u. *Adding ending block* H to u, which consists of substituting H for some leaf edge (u, ℓ) incident to u, is the term used when the leaf edge being replaced is not explicitly specified.

We define the six types of ULP joining and ending blocks in Fig. 2.2 as follows:

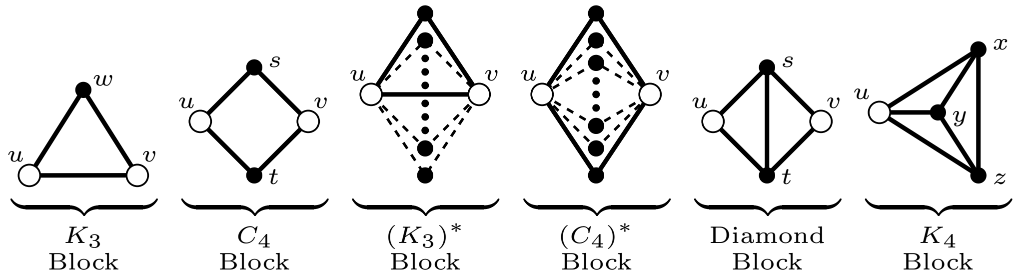


Figure 2.2: Six types of ULP blocks: K_3, C_4, $(K_3)^*$, $(C_4)^*$, diamond, and K_4 blocks. The white vertices are the possible connectors for each block.

Definition 2.3.3.

(a) A $(K_3)^m$ block *is a* $\overline{K_m} \vee P_2$ *join where* P_2 *is the edge* (u, v)*. A* $(K_3)^*$ block *denotes a* $(K_3)^m$ *block for an arbitrary* $m \geq 0$*, where* $(K_3)^0$ *is* P_2*.*

(b) A K_3 block *is a* $(K_3)^1$ *block, i.e., a 3-cycle* $u\text{–}v\text{–}w\text{–}u$ *on* $\{u, v, w\}$*.*

(c) A $(C_4)^m$ block *is a* $\overline{K_{m+1}} \vee \overline{P_2}$ *join where* $\overline{P_2}$ *is on* $\{u, v\}$*. A* $(C_4)^*$ block *denotes a* $(C_4)^m$ *block for an arbitrary* $m \geq 1$*.*

(d) A C_4 block *is a* $(C_4)^1$ *block, i.e., a 4-cycle* $u\text{–}s\text{–}v\text{–}t\text{–}u$ *on* $\{u, v, s, t\}$*.*

(e) A diamond block *is a* $\overline{K_2} \vee P_2$ *join where* $\overline{K_2}$ *is on* $\{u, v\}$ *and* P_2 *is the edge* (s, t)*, i.e., a* $(K_3)^2$ *on* $\{u, v, s, t\}$ *where* $deg(u) = deg(v) = 2$ *and* $deg(l) = deg(t) = 3$*.*

(f) A K_4 block *is the complete graph on* $\{u, x, y, z\}$*.*

Each of the six blocks either can be a 0-block, an ending block with connector u that *ends* on u, or a joining block with connectors u and v that *joins* u to v, with the one exception that a K_4 block cannot be a joining block.

Definition 2.3.4. *For a given set of ULP joining and ending blocks* $\mathcal{B}$*: A tree* T *(that is not a star) with spine* S *is* generalized *by substituting any edge* (u, v) *of* S *for a joining block in* $\mathcal{B}$ *and by adding at most one ending block in* $\mathcal{B}$ *to each endpoint* u *of* S*. If* T *is a star, then* T *is generalized by adding at most two ending blocks in* $\mathcal{B}$ *to the root* u*. An edge also has the trivial generalization that consists of replacing the edge with a single block in* $\mathcal{B}$*.*

Definition 2.3.5. *A* K_3-caterpillar *(K3C) is a caterpillar generalized with* K_3 *blocks; see Figs. 2.1(b) and 2.2.*

Definition 2.3.6. *A* generalized caterpillar *(GC) is a caterpillar generalized with* $(K_3)^*$, $(C_4)^*$, *diamond, and* K_4 *blocks; see Fig. 2.2.*

Chapter 3

Recognizing ULP Trees and Graphs

In this chapter, we show how to recognize the following four classes in linear time:

Definition 3.0.1.

1. $\mathcal{T}^*_{\mathsf{ULP}} = \{G : G \textit{ is a caterpillar}\}$,
2. $\mathcal{T}_{\mathsf{ULP}} = \mathcal{T}^*_{\mathsf{ULP}} \cup \{G : G \textit{ is a radius-}2 \textit{ star or a degree-}3 \textit{ spider}\}$,
3. $\mathcal{G}^*_{\mathsf{ULP}} = \{G : G \textit{ is a } K_3\textit{-caterpillar or is isomorphic to } G_\omega\}$, *and*
4. $\mathcal{G}_{\mathsf{ULP}} = \{G : G$ *is a radius-*2 *star, a generalized caterpillar, an extended* 3*-spider, or an extended* K_4 *subgraph*$\}$,

which we shall show in chapters 4 and 7 are the classes of ULP trees and graphs with duplicate and distinct labels, respectively.

First, we show how to remove pendant vertices efficiently from a graph, which is an operation we use repeatedly. Then, we give the recognition algorithms for ULP trees. Next, we show how to recognize the ULP blocks of Definition 2.3.3, which form the building blocks for generalized caterpillars and K_3-caterpillars, and are also used to recognize biconnected extended K_4 subgraphs. We conclude this chapter with the full recognition algorithms for all ULP graphs.

3.1 Removing Leaves Efficiently

Many of our algorithms take a graph 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 pendant vertices of a graph efficiently.

Algorithm: Remove-Leaves

▷ `Remove all pendant vertices from a graph` G `in` $O(n)$ `time`

Input: Adjacency list L representation of graph G
Output: Modified adjacency list L with pendant vertices removed from G

foreach *pendant vertex* ℓ *in* G *;* ▷ `where` $|L[\ell]| = 1$
do
  $\{v\} \leftarrow L[\ell]$, $p \leftarrow$ pointer to ℓ in $L[v]$
  Delete list $L[\ell]$ from L
  Use p to remove ℓ from the doubly-linked list $L[v]$ in $O(1)$ time.
return L

Figure 3.1: Remove-Leaves

Lemma 3.1.1. *Pendant vertices can be removed from an n-vertex graph in $O(n)$ time.*

Proof. Removing a pendant vertex from a graph can be done in $O(1)$ time if G 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. Figure 3.1 gives the function Remove-Leaves that uses this representation to run in $O(n)$ time. □

3.2 Recognizing ULP Trees in Linear Time

We shall show in chapter 4 that any ULP tree can be drawn in linear-time. However, we first need to determine if the tree in question is ULP before doing so. The next theorem gives our linear-time recognition algorithm.

Theorem 3.2.1. *Any n-vertex ULP tree $G(V, E)$ can be recognized in $O(n)$ time.*

Proof. First, we need to determine if G is a tree by checking that G is connected (can be done in $O(n)$ time using a depth-first search) and that G has $n - 1$ edges. 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 G is a caterpillar. Otherwise, we also need to determine whether G is a radius-2 star or a degree-3 spider. Figures 3.2, 3.3,

Algorithm: IS-CATERPILLAR

▷ `Recognize an` n`-vertex caterpillar in` $O(n)$ `time`

Input: Graph $G(V, E)$
Output: Return *true* if G is a caterpillar, *false* otherwise

if G *is not a tree* **then return** *false*
$T' \leftarrow$ REMOVE-LEAVES(G) ▷ `should be a path`
if $\Delta(T') \leq 2$ **then return** *true* **else return** *false*

Figure 3.2: IS-CATERPILLAR

Algorithm: IS-RADIUS-2-STAR

▷ `Recognize an` n`-vertex radius-2 star in` $O(n)$ `time`

Input: Graph $G(V, E)$
Output: Return *true* if G is a radius-2 star, *false* otherwise

if G *is not a tree* **then return** *false*
$T' \leftarrow$ REMOVE-LEAVES(G) ▷ `leaves of` T `should have degree-2 in` G
$T'' \leftarrow$ REMOVE-LEAVES(T') ▷ `should be an isolated vertex`
foreach *leaf* $v \in T'$ **do if** $deg(v) \neq 2$ *in* G **then return** *false*
if $\Delta(G) \geq 3$ **and** $n(T'') = 1$ **then return** *true* **else return** *false*

Figure 3.3: IS-RADIUS-2-STAR

Algorithm: IS-DEGREE-3-SPIDER

▷ `Recognize an` n`-vertex degree-3 spider in` $O(n)$ `time`

Input: Graph $G(V, E)$
Output: Return *true* if G is a degree-3 spider, *false* otherwise

if G *is not a tree* **then return** *false*
$V_{deg3} \leftarrow \{v : v \in V(G) \text{ and } deg(v) = 3\}$ ▷ `degree-3 vertices of` G
if $\Delta(G) = 3$ **and** $|V_{deg3}| = 1$ **then return** *true* **else return** *false*

Figure 3.4: IS-DEGREE-3-SPIDER

3.4, and 3.5 give the respective algorithms IS-CATERPILLAR, IS-RADIUS-2-STAR, IS-DEGREE-3-SPIDER, and IS-ULP-TREE. □

Algorithm: Is-ULP-Tree

▷ `Recognize an` n`-vertex ULP tree in` $O(n)$ `time`

Input: Graph $G(V,E)$, k number of levels
Output: Return *true* if G is an ULP tree, *false* otherwise

if $k < n(G)$ **then return** Is-Caterpillar(G)
else return Is-Caterpillar(G) **or** Is-Radius-2-Star(G) **or**
Is-Degree-3-Spider(G)

Figure 3.5: Is-ULP-Tree

3.3 Recognizing ULP Blocks in Linear Time

In this section, we provide the recognition algorithms for the seven cases of ULP blocks given in Fig. 3.6. Both P_2 and K_3 blocks are the two types of $(K_3)^*$ blocks and C_4 blocks are the one type of $(C_4)^*$ blocks that do not have vertices of degree greater than 2. As result, these three types of blocks require specialized recognition algorithms.

Lemma 3.3.1. *Any k-vertex ULP block $B(V,E)$ can be recognized in $O(k)$ time.*

Proof. Figures 3.7, 3.8, and 3.9 give the algorithms Is-P_2-Block, Is-K_3-Block, and Is-C_4-Block for the three possible ULP blocks with maximum degree 2.

The Is-$(K_3)^*$-Block and Is-$(C_4)^*$-Block algorithms given in Figs. 3.10 and 3.11, respectively, provide the additional pseudocode to recognize $(K_3)^m$ and $(C_4)^m$ blocks for $m \geq 2$ where the possible connectors are the vertices u and v

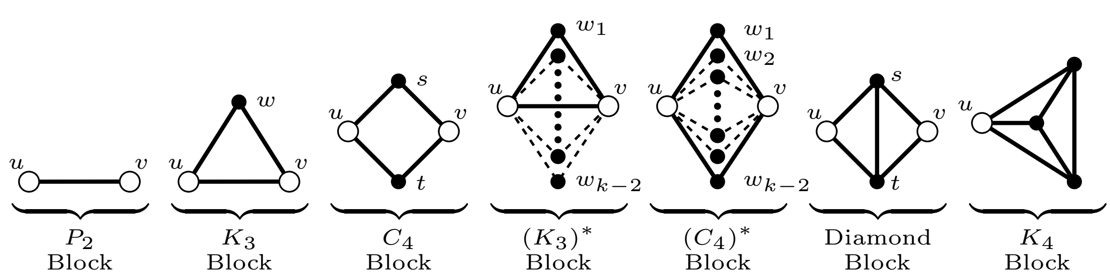


Figure 3.6: Seven cases of ULP blocks to recognize: P_2, K_3, C_4, $(K_3)^*$, $(C_4)^*$, diamond, and K_4 blocks. Note that P_2, K_3, and C_4 blocks are special cases of $(K_3)^*$ and $(C_4)^*$ blocks. The white vertices are the possible connectors for each block.

Algorithm: Is-P_2-Block

▷ `Recognize a` P_2 `block in` $O(1)$ `time`

Input: Block $B(V, E)$, connector $u \in V$, connector $v \in V$ (which is optional)
Output: Return *true* if B is a P_2 block ending on u (i.e., a pendant edge incident to u) or joining u to v (if given), *false* otherwise

if $|V| \neq 2$ **or** $|E| \neq 1$ **then return** *false* ▷ `ensure` $O(1)$ `running time`
if v *not given* **then** let $v \in N(u)$ ▷ `any vertex adjacent to` u `in` B
if $V = \{u, v\}$ **and** $E = \{(u, v)\}$ **then return** *true* **else return** *false*

Figure 3.7: Is-P_2-Block

Algorithm: Is-K_3-Block

▷ `Recognize a` K_3 `block in` $O(1)$ `time`

Input: Block $B(V, E)$, connector $u \in V$, connector $v \in V$ (which is optional)
Output: Return *true* if B is a K_3 block ending on u or joining u to v (if given), *false* otherwise

if $|V| \neq 3$ **or** $|E| \neq 3$ **then return** *false* ▷ `ensure` $O(1)$ `running time`
if v *not given* **then** let $v \in N(u)$ ▷ `any vertex adjacent to` u `in` B
$\{w\} \leftarrow V \setminus \{u, v\}$ ▷ `other vertex in` B
if $E = \{(u, v), (v, w), (u, w)\}$ **then return** *true* **else return** *false*

Figure 3.8: Is-K_3-Block

Algorithm: Is-C_4-Block

▷ `Recognize a` C_4 `block in` $O(1)$ `time`

Input: Block $B(V, E)$, connector $u \in V$, connector $v \in V$ (which is optional)
Output: Return *true* if B is a C_4 block ending on u or joining u to v (if given), *false* otherwise

if $|V| \neq 4$ **or** $|E| \neq 4$ **then return** *false* ▷ `ensure` $O(1)$ `running time`
if v *not given* **then** let $v \notin N(u)$ ▷ `any vertex not adjacent to` u `in` B
$\{s, t\} \leftarrow V \setminus \{u, v\}$ ▷ `other vertices in` B
if $E = \{(u, s), (u, t), (v, s), (v, t)\}$ **then return** *true* **else return** *false*

Figure 3.9: Is-C_4-Block

with degree greater than 2. All the other vertices must have degree 2 and must be adjacent to u and v.

Algorithm: Is-$(K_3)^*$-Block

$\triangleright$ `Recognize a` k`-vertex` $(K_3)^*$ `block in` $O(k)$ `time`

Input: Block $B(V, E)$, connector $u \in V$, connector $v \in V$ (which is optional)
Output: Return *true* if B is a $(K_3)^*$ block ending on u or joining u to v (if given), *false* otherwise

if v *not given* **then** let $v \in V \setminus \{u\}$ **where** $v = \Delta(B)$
if Is-P_2-Block(B, u, v) **or** Is-K_3-Block(B, u, v) **then return** *true*
$V_{deg=2} \leftarrow \{v : v \in V \text{ and } deg(v) = 2\}$ $\triangleright$ `degree-2 vertices of` B
$V_{deg>2} \leftarrow \{v : v \in V \text{ and } deg(v) > 2\}$ $\triangleright$ `vertices of degree` >2 `in` B
if $V_{deg>2} \neq \{u, v\}$ **or** $V \neq V_{deg=2} \cup V_{deg>2}$ **then return** *false*
else $\{w_1, \ldots, w_{k-2}\} \leftarrow V_{deg=2}$
if $u \in N(v)$ **and** $N(u) \cap N(v) = V_{deg=2}$ **and** $N(w_1) = \cdots = N(w_{k-2}) = \{u, v\}$
then return *true* **else return** *false*

Figure 3.10: Is-$(K_3)^*$-Block

Algorithm: Is-$(C_4)^*$-Block

$\triangleright$ `Recognize a` k`-vertex` $(C_4)^*$ `block in` $O(k)$ `time`

Input: Block $B(V, E)$, connector $u \in V$, connector $v \in V$ (which is optional)
Output: Return *true* if B is a $(C_4)^*$ block ending on u or joining u to v (if given), *false* otherwise

if v *not given* **then** let $v \in V \setminus \{u\}$ **where** $v = \Delta(B)$
if Is-C_4-Block(B, u, v) **then return** *true*
$V_{deg=2} \leftarrow \{v : v \in V \text{ and } deg(v) = 2\}$ $\triangleright$ `degree-2 vertices of` B
$V_{deg>2} \leftarrow \{v : v \in V \text{ and } deg(v) > 2\}$ $\triangleright$ `vertices of degree` >2 `in` B
if $V_{deg>2} \neq \{u, v\}$ **or** $V \neq V_{deg=2} \cup V_{deg>2}$ **then return** *false*
else $\{w_1, \ldots, w_{k-2}\} \leftarrow V_{deg=2}$
if $u \notin N(v)$ **and** $N(u) \cap N(v) = V_{deg=2}$ **and** $N(w_1) = \cdots = N(w_{k-2}) = \{u, v\}$
then return *true* **else return** *false*

Figure 3.11: Is-$(C_4)^*$-Block

Figure 3.12 gives the Is-Diamond-Block algorithm that determines whether the two connectors u and v have degree 2, and whether the remaining two vertices s and t are adjacent to each other and to u and v, and both have degree 3 as a result.

Figure 3.13 gives the Is-K_4-Block algorithm that checks that the block has four vertices and six edges. This is only block recognition algorithm that does not take a connector as an input, since any vertex of a K_4 block can serve as a connector.

Algorithm: Is-Diamond-Block

▷ `Recognize a diamond block in` $O(1)$ `time`

Input: Block $B(V, E)$, connector $u \in V$, connector $v \in V$ (which is optional)
Output: Return *true* if B is a diamond block ending on u or joining u to v (if given), *false* otherwise

if $|V| \neq 4$ **then return** *false* ▷ `ensure` $O(1)$ `running time`
if v *not given* **then** let $v \notin N(u)$ ▷ `any vertex not adjacent to` u `in` B
$\{s, t\} \leftarrow V \setminus \{u, v\}$
if $E = \{(u, s), (u, t), (v, s), (v, t), (s, t)\}$ **then return** *true*
else return *false*

Figure 3.12: Is-Diamond-Block

Algorithm: Is-K_4-Block

▷ `Recognize a` K_4 `block in` $O(1)$ `time`

Input: Block $B(V, E)$
Output: Return *true* if B is a K_4 block, *false* otherwise

if $|V| = 4$ **and** $|E| = 6$ **then return** *true* **else return** *false*

Figure 3.13: Is-K_4-Block

Algorithm: Is-ULP-Block

▷ `Recognize a` k`-vertex ULP block in` $O(k)$ `time`

Input: Block $B(V, E)$, connector $u \in V$, connector $v \in V$ (which is optional)
Output: Return *true* if B is an ULP block ending on u or joining u to v (if given), *false* otherwise

if v *not given* **then** ▷ `if` B `is an ending block`
  return Is-K_4-Block(B) **or** Is-Diamond-Block(B, u) **or** Is-$(K_3)^*$-Block(B, u) **or** Is-$(C_4)^*$-Block(B, u)
else ▷ `else` B `is a joining block`
  return Is-Diamond-Block(B, u, v) **or** Is-$(K_3)^*$-Block(B, u, v) **or** Is-$(C_4)^*$-Block(B, u, v)

Figure 3.14: Is-ULP-Block

Finally, Fig. 3.14 gives the algorithm Is-ULP-Block that combines the four previous algorithms to recognize either ULP ending blocks if only the one connector u is given or ULP joining blocks if both connectors u and v are given. □

3.4 Recognizing ULP Graphs in Linear Time

We shall show in chapter 7 that any ULP graph can be drawn in linear-time. As a necessary precursor, we need to determine whether a graph is ULP in linear time, and in the cases of generalized caterpillars and K_3-caterpillars, have a way to efficiently decompose the graph into blocks.

Lemma 3.4.1. *An n-vertex generalized caterpillar $G(V, E)$ can be recognized in $O(n)$ time.*

Proof. If G is biconnected, then G only has one block. We can pick the vertex of maximum degree, which will always correspond to a possible connector of an ULP block, and check if G is an ULP block that ends on u. Otherwise, G has at least one cut-vertex, and we can use depth-first search to determine the blocks and cut-vertices of G in linear time [81].

If any of the blocks of G are not 1-blocks or 2-blocks, then G cannot be a GC by Definition 2.3.6, since G only consists of ULP joining and ending blocks and pendant edges. We can replace each 2-block with connectors u and v in G with the edge (u, v) and replace each 1-block with the connector u with a pendant edge incident to u. This produces a tree T. If T is a not a caterpillar, then G cannot be a GC, since a tree is used to generalize G by Definition 2.3.6.

If T is a caterpillar, then we let $v_1, \ldots, v_k$ denote the cut-vertices of T, which are also the cut vertices of G, where $v_1 - \cdots - v_k$ forms the spine of T. If there is only one cut-vertex, then T must be a star. According to Definition 2.3.4, G can have at most two ULP blocks from Definition 2.3.3 that end on u, which are the only non-trivial blocks of G.

Else $v_1 \neq v_k$, and v_1 and v_k can each have at most one ending ULP block, B_0 and B_k, respectively, which are the only possible non-trivial 1-blocks of G. All the 2-blocks connect adjacent pairs of cut vertices in T. We can check that each non-trivial block B_i with connectors v_i and v_{i+1} for $i \in [1..k-1]$ is an ULP block joining v_i to v_{i+1} and that B_0 and B_k (if present) are ULP blocks ending on v_1

Algorithm: IS-GENERALIZED-CATERPILLAR

▷ `Recognize an` n`-vertex generalized caterpillar in` $O(n)$ `time`

Input: Graph $G(V, E)$
Output: Return *true* if G is a generalized caterpillar, *false* otherwise

```
if G is biconnected then                              ▷ G only has one block
    u ← any vertex of maximum degree in G
    return Is-ULP-Block(G, u)
ℬ ← non-trivial blocks of G                  ▷ done in O(n) time using DFS
foreach B ∈ ℬ do
    if B a k-block for k > 2 then return false
T ← tree created by replacing each 1-block and 2-block in ℬ with an edge
if not Is-Caterpillar(T) then return false
v_1−···−v_k ← Remove-Leaves(T)                          ▷ cut-vertices in G
if k = 1 then                                                ▷ if T is a star
    if |ℬ| > 2 then return false  ▷ has at most two non-trivial blocks
    {B_0, B_k} ← non-trivial 1-blocks of ℬ             ▷ either may be empty
else                                                  ▷ else T is not a star
    B_0 ← non-trivial 1-block of ℬ with connector v_1        ▷ maybe be empty
    B_1, ..., B_{k−1} ← 2-blocks of G where B_i has connectors v_i and v_{i+1}
    B_k ← non-trivial 1-block of ℬ with connector v_k        ▷ maybe be empty
    if ℬ ⊈ {B_0, B_1, ..., B_k} then       ▷ only B_0 and B_k can be 1-blocks
        return false
if B_0 is non-empty and not Is-ULP-Block(B_0, v_1) then return false
if B_k is non-empty and not Is-ULP-Block(B_k, v_k) then return false
foreach B_i ∈ {B_1, ..., B_{k−1}} do
    if not Is-ULP-Block(B_k, v_i, v_{i+1}) then return false
return true
```

Figure 3.15: IS-GENERALIZED-CATERPILLAR

and v_k, respectively. Figure 3.15 gives the final linear-time recognition algorithm IS-GENERALIZED-CATERPILLAR. □

We can adapt the previous algorithm IS-GENERALIZED-CATERPILLAR to recognize the first class of graphs, K_3-caterpillars, in $\mathcal{G}^*_{\mathsf{ULP}}$, by replacing each instance of the call to IS-ULP-BLOCK with a call to IS-K_3-BLOCK. This gives the algorithm IS-K_3-CATERPILLAR in Fig. 3.16 and the following corollary:

Algorithm: Is-K_3-Caterpillar

▷ `Recognize an` n`-vertex` K_3-caterpillar `in` $O(n)$ `time`

Input: Graph $G(V, E)$
Output: Return *true* if G is a K_3-caterpillar, *false* otherwise

if *G is biconnected* **then** ▷ G `only has one block`
  $u \leftarrow$ any vertex of maximum degree in G
  return Is-K_3-Block(G, u)

.
.
.

foreach $B_i \in \{B_1, \ldots, B_{k-1}\}$ **do**
  if not Is-K_3-Block(B_k, v_i, v_{i+1}) **then return** *false*
return *true*

Figure 3.16: Is-K_3-Caterpillar

Corollary 3.4.2. *An n-vertex K_3-caterpillar $G(V, E)$ can be recognized in $O(n)$ time.*

Section 3.2 contains the recognition algorithm Is-Radius-2-Star for the second class of graphs, radius-2 stars, in $\mathcal{G}_{\mathsf{ULP}}$. The recognition algorithm for the third class of graphs in $\mathcal{G}_{\mathsf{ULP}}$, extended 3-spiders, is given next.

Lemma 3.4.3. *An n-vertex extended 3-spider $G(V, E)$ can be recognized in $O(n)$ time.*

Proof. According to Definition 2.3.1, an E3S can either have one or three degree-3 vertices (depending on whether there is an edge connecting two of the three neighbors of the root of a degree-3 spider); see Fig. 2.1(g). Additionally, any other vertex of an E3S has degree 1 or 2 regardless of whether there is an edge that connects two of the three pendant vertices. Hence, an E3S has maximum degree 3. Thus, by checking if the maximum degree of G is 3 and if G has one or three vertices of degree 3 that are pairwise adjacent, suffices to determine whether G is an E3S. Algorithm Is-Extended-3-Spider given in Fig. 3.17 performs these checks and calls Is-K_3-Block from Fig. 3.8 to determine if the induced graph on the three degree-3 vertices forms a K_3 block. □

Algorithm: IS-EXTENDED-3-SPIDER

▷ `Recognize an` n`-vertex extended` 3-spider `in` $O(n)$ `time`

Input: Graph $G(V, E)$
Output: Return *true* if G is an extended 3-spider, *false* otherwise

if $\Delta(G) \neq 3$ **then return** *false* ▷ `maximum degree must be 3`
$V_{deg=3} \leftarrow \{v : v \in V \text{ and } deg(v) = 3\}$ ▷ `degree-3 vertices of` G
if $|V_{deg=3}| = 1$ **then return** *true* ▷ `has 1 degree-3 vertex`
if $|V_{deg=3}| \neq 3$ **then return** *false* ▷ `neither 1 or 3 degree-3 vertices`
$B \leftarrow G[V_{deg=3}]$ ▷ `induced graph on degree-3 vertices`
$u \in V_{deg=3}$ ▷ `any degree-3 vertex`
if IS-K_3-BLOCK(B, u) **then return** *true* ▷ `vertices pairwise adjacent`
else return *false*

Figure 3.17: IS-EXTENDED-3-SPIDER

The second class of graphs in $\mathcal{G}^*_{\mathsf{ULP}}$, graphs isomorphic to G_ω, are a special case of an extended 3-spider. We can adapt IS-EXTENDED-3-SPIDER by checking that there are exactly three degree-1 and three degree-3 vertices, which is sufficient to determine whether the graph is isomorphic to G_ω. This gives the algorithm IS-G_ω in Fig. 3.18 and following corollary:

Corollary 3.4.4. *A graph isomorphic to G_ω can be recognized in $O(1)$ time.*

Algorithm: IS-G_ω

▷ `Recognize graph isomorphic to` G_ω `in` $O(1)$ `time`

Input: Graph $G(V, E)$
Output: Return *true* if G is isomorphic to G_ω, *false* otherwise

if $|V| \neq 6$ **then return** *false* ▷ `ensure` $O(1)$ `running time`
$V_{deg=1} \leftarrow \{v : v \in V \text{ and } deg(v) = 1\}$ ▷ `degree-1 vertices of` G
$V_{deg=3} \leftarrow \{v : v \in V \text{ and } deg(v) = 3\}$ ▷ `degree-3 vertices of` G
if $|V_{deg=1}| = |V_{deg=3}| = 3$ **and** $V = V_{deg=1} \cup V_{deg=3}$ **then return** *true*
else return *false*

Figure 3.18: IS-G_ω

Instead of having to provide a recognition algorithm for the final class of extended K_4 subgraphs in $\mathcal{G}_{\mathsf{ULP}}$, we only need to consider the biconnected case as we see next.

Lemma 3.4.5. *A connected extended K_4 subgraph that is not biconnected is either a generalized caterpillar or an extended* 3*-spider.*

Proof. Let B be an extended K_4 as shown in Fig. 2.1(i), which according to Definition 2.3.2 consists of a K_4 on the vertices $\{s, t, u, v\}$ in which one edge (u, v) has been subdivided into the path $u \rightsquigarrow v$. Removing any edge e from the path $u \rightsquigarrow v$ gives a GC that consists of path in which at most one edge has been replaced with a K_3, C_4, or a diamond ULP joining or ending block.

If an edge e is removed from the diamond block B connecting u to v, then $B - e$ either forms a C_4 block joining u to v or consists of the edge (u, t) or (v, t) and a K_3 block joining t to v or u, respectively. If the pair of edges e_1 and e_2 are removed from B where e_1 and e_2 are (u, s) and (u, t) or e_1 and e_2 are (v, s) and (v, t), then $G - e_1 - e_2$ forms a GC that consists of a path and a K_3 ending block. If e_1 and e_2 are one of the other three pairs of edges of B, namely (u, t) and (v, t), (u, s) and (s, t), or (v, s) and (s, t), then $G - e_1 - e_2$ is an E3S with a single degree-3 vertex, which is either t, u, or v, respectively, that forms the root of the E3S. Finally, if three or more edges are removed from B, then G is either a cycle or a caterpillar. ☐

By considering the biconnected cases in the previous proof of Lemma 3.4.5, we have the following corollary that gives an alternate definition of biconnected EK4s in terms of ULP blocks:

Corollary 3.4.6. *A biconnected extended K_4 subgraph consists of a cycle with at most one edge (u, v) replaced by a K_3, C_4, or diamond block joining u to v.*

Using Corollary 3.4.6, we produce a linear-time recognition algorithm next.

Lemma 3.4.7. *An n-vertex biconnected extended K_4 subgraph $G(V, E)$ can be recognized in $O(n)$ time.*

Proof. First, we check that G is biconnected. If the maximum degree of G is 2, then G must be a cycle, which is a biconnected EK4. Otherwise, by Corollary 3.4.6 the maximum degree of G must be 3 where an edge (u, v) of a cycle has been replaced with a K_3, a C_4, or a diamond block B. If B is a K_3 or C_4 block, then u and v

Algorithm: IS-BICONNECTED-EXTENDED-K_4-SUBGRAPH

▷ `Recognize an` n`-vertex biconnected extended` K_4 `subgraph in` $O(n)$ `time`

Input: Graph $G(V, E)$
Output: Return *true* if G is a biconnected extended K_4 subgraph, *false* otherwise

```
if G is not biconnected then return false
if Δ(G) = 2 then return true                              ▷ G is a cycle
if Δ(G) ≠ 3 then return false                 ▷ maximum degree at most 3
V_deg=3 ← {v : v ∈ V and deg(v) = 3}            ▷ degree-3 vertices of G
if |V_deg=3| ≠ 2 or |V_deg=3| ≠ 4 then return false  ▷ only possibilities
if |V_deg=3| = 2 then                  ▷ if has two degree-3 vertices
    {u, v} ← V_deg=3
    B ← G[N[u] ∩ N[v]]                      ▷ block containing u and v
    return Is-K3-Block(B, u, v) or Is-K4-Block(B, u, v)
else                                   ▷ else has four degree-3 vertices
    B ← G[V_deg=3]                 ▷ induced graph on degree-3 vertices
    if B is not a block or Is-K4-Block(B) then return false
    Let {u, v} ⊂ V_deg=3 where v ∉ N(u)
    return Is-Diamond-Block(B, u, v)
```

Figure 3.19: IS-BICONNECTED-EXTENDED-K_4-SUBGRAPH

are the only degree-3 vertices. The induced graph on u and v and their common neighbor(s) must be the block B joining u to v.

Otherwise, B must be a diamond block, and G has four degree-3 vertices $\{u, v, s, t\}$. Moreover, B must be the induced graph $G[\{u, v, s, t\}]$ where B joins u to v, which are are the only two non-adjacent vertices of B Figure 3.19 gives the algorithm IS-BICONNECTED-EXTENDED-K_4-SUBGRAPH, which performs these checks to determine whether G is a biconnected EK4. □

Combining the three recognition algorithms IS-GENERALIZED-CATERPILLAR, IS-EXTENDED-3-SPIDER, and IS-BICONNECTED-EXTENDED-K_4-SUBGRAPH from Lemmas 3.4.1, 3.4.3, and 3.4.7 with algorithm IS-RADIUS-2-STAR from section 3.2, we can recognize graphs in $\mathcal{G}_{\mathsf{ULP}}$ in linear time. Similarly, combining algorithms IS-K_3-CATERPILLAR and IS-G_ω from Corollaries 3.4.2 and 3.4.4, we can also rec-

Algorithm: IS-ULP-GRAPH

▷ `Recognize an` n`-vertex ULP graph in` $O(n)$ `time`

Input: Graph $G(V, E)$, k number of levels
Output: Return *true* if G is an ULP graph, *false* otherwise

if $k < n(G)$ **then return** IS-K_3-CATERPILLAR(G) **or** IS-G_ω(G)
else return IS-GENERALIZED-CATERPILLAR(G) **or** IS-RADIUS-2-STAR(G)
or IS-EXTENDED-3-SPIDER(G)
or IS-BICONNECTED-EXTENDED-K_4-SUBGRAPH(G)

Figure 3.20: IS-ULP-GRAPH

ognize graphs in $\mathcal{G}^*_{\mathsf{ULP}}$ in linear time. This gives the algorithm IS-ULP-GRAPH in Fig. 3.20 and the final theorem of the chapter.

Theorem 3.4.8. *Any n-vertex ULP graph $G(V, E)$ can be recognized in $O(n)$ time.*

Chapter 4

Drawing ULP Trees

In this chapter, we show how to draw caterpillars, radius-2 stars, and degree-3 spiders for any distinct labeling and how to draw caterpillars for any duplicate labeling.

4.1 Drawing Caterpillars with Distinct Labels

Brass *et al.* [13] gave an algorithm that produces a simultaneous geometric embedding of a caterpillar and a path on n vertices on an $2n \times n$ grid. We give an algorithm for producing a more compact drawing with the next lemma.

Lemma 4.1.1. *An n-vertex caterpillar T with an m-vertex spine can be straight-line realized in $O(n)$ time on a $2m \times n$ grid for any distinct labeling.*

Proof. If $n \leq 2$, then T is an isolated vertex or edge and is easily drawn. Otherwise, T has spine $v_1 - \cdots - v_m$ that is drawn with vertices placed at odd x-coordinates.

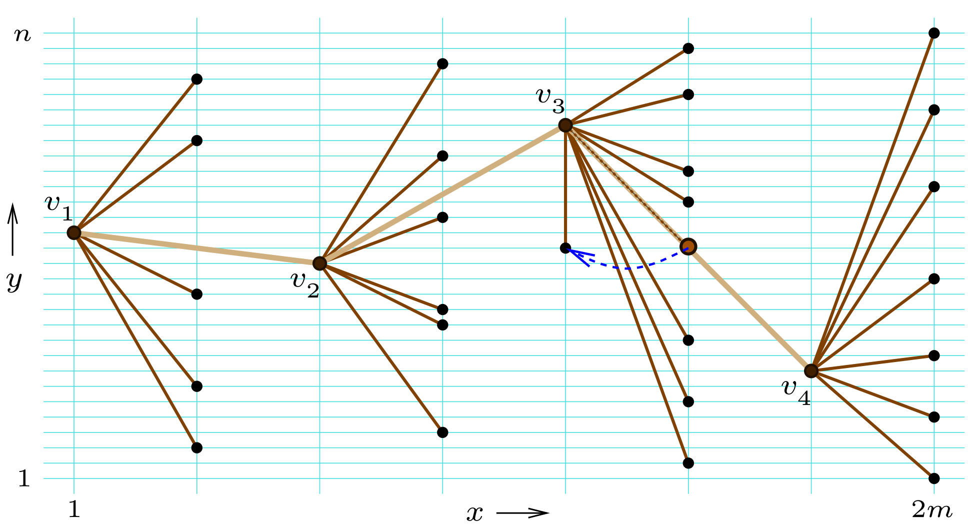


Figure 4.1: Realization of a caterpillar with distinct labels on a 8×30 grid

Algorithm: Draw-Caterpillar

▷ `Draw a caterpillar with distinct labels in` $O(n)$ `time`

Input: Caterpillar $T(V, E, \phi)$ with distinct labels ϕ where $|V| > 2$
Output: Level planar drawing of T

$S \leftarrow$ Remove-Leaves(T) ▷ `spine` $v_1 — \cdots — v_m$
for $i \leftarrow 1$ *to* m **do** draw spine S by placing v_i at $\big(2i-1,\, \phi(v_i)\big)$
for $i \leftarrow 1$ *to* $m-1$ **do** draw edge v_i–v_{i+1}
for $i \leftarrow 1$ *to* m **do**
  foreach *leaf* $\ell \in N(v_i)$ **do**
    if *leaf* ℓ *would lie on* v_i–v_{i+1} **then**
      Place leaf ℓ at $\big(2i-1,\, \phi(\ell)\big)$ above or below v_i
    else Place ℓ right of v_i at $\big(2i,\, \phi(\ell)\big)$
    Draw edge v_i–ℓ

Figure 4.2: Draw-Caterpillar with distinct labels

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. 4.1. Algorithm Draw-Caterpillar in Fig. 4.2 takes $O(n)$ time as the location of each vertex is determined in $O(1)$ time. □

4.2 Drawing Radius-2 Stars with Distinct Labels

Lemma 4.2.1. *An n-vertex radius-2 star T can be straight-line realized 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. 4.3. The x-coordinate ℓ_x of ℓ can be found by solving the equation $\ell_x - u_x = \phi(\ell) - \phi(u)$ to get $\ell_x = \phi(\ell) - \phi(u) + u_x$.

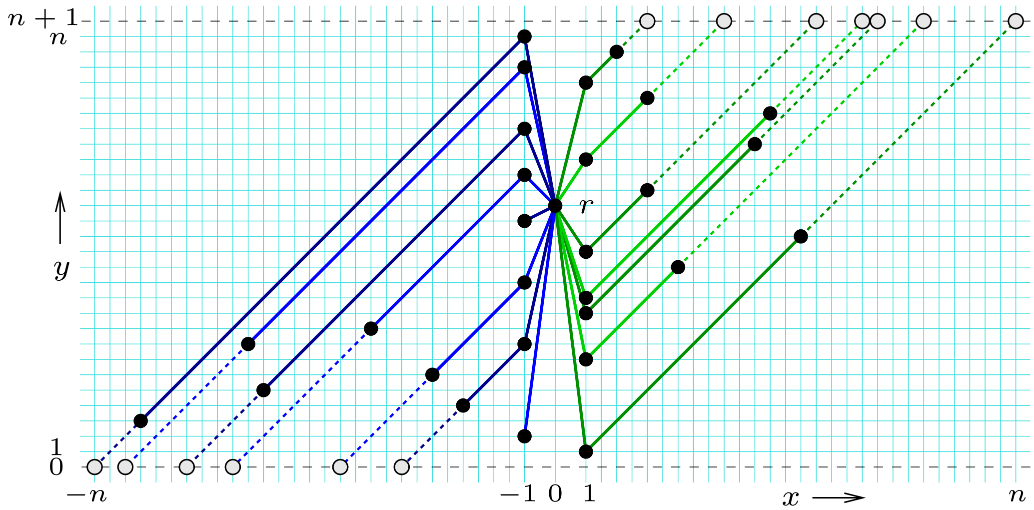


Figure 4.3: Realization of a radius-2 star with distinct labels on a 59×29 grid. The gray nodes indicate the intersection points of rays of slope 1 emanating from each leaf to imagined level-0 and level-$(n+1)$ that are drawn with dashed lines.

This means that $\ell_x = \phi(\ell) - \phi(u) + 1$ if $\phi(\ell) > \phi(u)$, otherwise, $\ell_x = \phi(\ell) - \phi(u) - 1$. Figure 4.4 gives the algorithm Draw-Radius-2-Star that takes $O(n)$ time since the coordinates of each vertex are determined in $O(1)$ time. □

Algorithm: Draw-Radius-2-Star

▷ `Draw a radius-2 star with distinct labels in` $O(n)$ `time`

Input: Radius-2 star $T(V, E, \phi)$ with distinct labels ϕ
Output: Level planar drawing of T

$r \leftarrow$ root of T ▷ `unique vertex of maximum degree`
Place r at $\big(0, \phi(r)\big)$
foreach $u \in N(r)$ **do**
  if $|N(u)| = 1$ **then** ▷ `if` u `is a leaf`
    Place u at $\big(-1, \phi(u)\big)$ and draw edge r–u
  else $\{\ell\} \leftarrow N(u) \setminus \{r\}$ ▷ `leaf vertex adjacent to` u
  if $\phi(\ell) > \phi(u)$ **then**
    Place u at $\big(1, \phi(u)\big)$ and ℓ at $\big(\phi(\ell) - \phi(u) + 1, \phi(\ell)\big)$
  else Place u at $\big(-1, \phi(u)\big)$ and ℓ at $\big(\phi(\ell) - \phi(u) - 1, \phi(\ell)\big)$
  Draw edges r–u and u–ℓ

Figure 4.4: Draw-Radius-2-Star

4.3 Drawing Degree-3 Spiders with Distinct Labels

Lemma 4.3.1. *An n-vertex degree-3 spider T 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; Fig. 4.5. 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 describe the high-level algorithm Draw-Degree-3-Spider given in Fig. 4.6 and the invariants it maintains. Then we 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 show 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

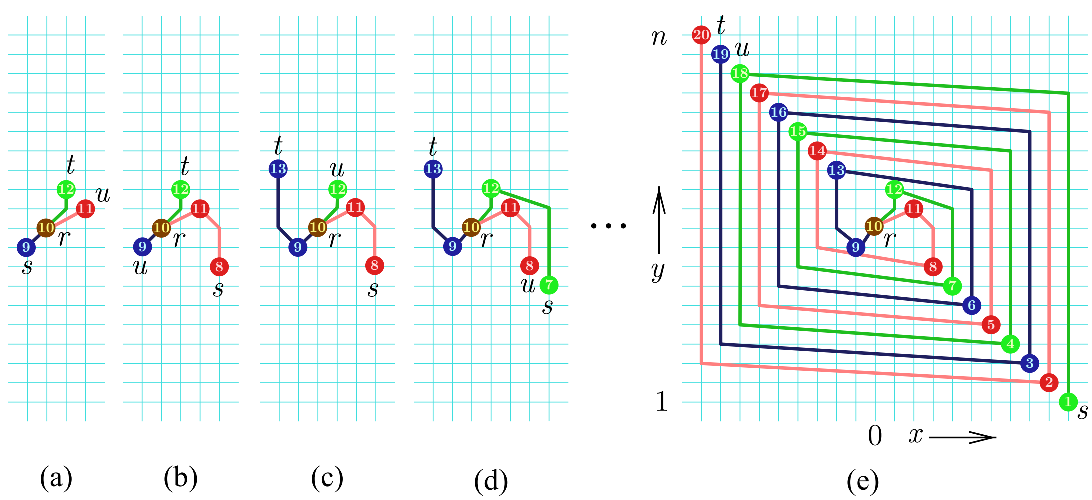


Figure 4.5: Step-by-step realization of a degree-3 spider with bends.

Algorithm: DRAW-DEGREE-3-SPIDER

▷ `Draw a degree-3 spider with distinct labels in` $O(n)$ `time`

Input: Degree-3 spider $T(V, E, \phi)$ with distinct labels ϕ
Output: Level planar drawing of T

$T' \leftarrow$ START-DRAWING-DEGREE-3-SPIDER$\big(T(V, E, \phi)\big)$
$U \leftarrow \{s, t, u\}$ be the leaves of T' such that $\phi(s) < \phi(u) < \phi(t)$
direction $\leftarrow$ RIGHT
while $|N(u)| \neq 1$ **do** ▷ `while` u `is not a leaf`
  $v \leftarrow$ EXPAND-CHAIN(T, T', u, *direction*)
  if $\phi(v) < \phi(s)$ *or* $\phi(v) > \phi(t)$ **then**
    if $\phi(v) < \phi(s)$ **then** update $u \leftarrow s$ and $s \leftarrow v$
    if $\phi(v) > \phi(t)$ **then** update $u \leftarrow t$ and $t \leftarrow v$
    Change *direction* ▷ `RIGHT to LEFT, and vice versa`
  else Update $u \leftarrow v$
if $x_s < x_t$ **then** *direction* $\leftarrow$ LEFT ▷ `if` s `is left of` t
else *direction* $\leftarrow$ RIGHT ▷ `else` s `is right of` t
while $|N(s)| \neq 1$ **do** ▷ `while` s `is not a leaf`
  $s \leftarrow$ EXPAND-CHAIN(T, T', s, *direction*)
Change *direction* ▷ `RIGHT to LEFT, and vice versa`
while $|N(t)| \neq 1$ **do** ▷ `while` t `is not a leaf`
  $t \leftarrow$ EXPAND-CHAIN(T, T', t, *direction*)

Figure 4.6: DRAW-DEGREE-3-SPIDER

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 and while the other is freely expanded to the right.

To guarantee that one chain can always be expanded to the left or to the right, two invariants are maintained by DRAW-DEGREE-3-SPIDER while there remains at least one vertex to place in each chain:

(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. 4.5(e), 4.10(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. 4.5(a) to (b) (here s, t, and u are vertices 9, 12, and 11 in Fig. 4.5(a), respectively, and v is vertex 8 in Fig. 4.5(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. 4.5(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. As a result, invariant (2) continues to hold with the old s now playing the role of the new u. This process continues until a chain is exhausted where the last placed vertex v is not ϕ-extreme. By invariant (1), the last placed vertices s and t of the remaining two chains must be ϕ-extreme as in Fig. 4.5(e). Hence, we can choose to freely expand the chains of s and t to the left (or to the right) and to the right (or to the left), respectively, if s lies to the left (or to the right) of t.

Initially drawing a degree-3 spider for which the two invariants hold is non-trivial as Fig. 4.7 illustrates. The algorithm Start-Drawing-Degree-3-Spider from Fig. 4.8 avoids this problem by first denoting the vertices adjacent to r by v_{min}, v_{mid}, and v_{max} such that $\phi(v_{min}) < \phi(v_{mid}) < \phi(v_{max})$. Fig. 4.7(a) gives an example of these three vertices with x-coordinates -1, 1, and 2, respectively.

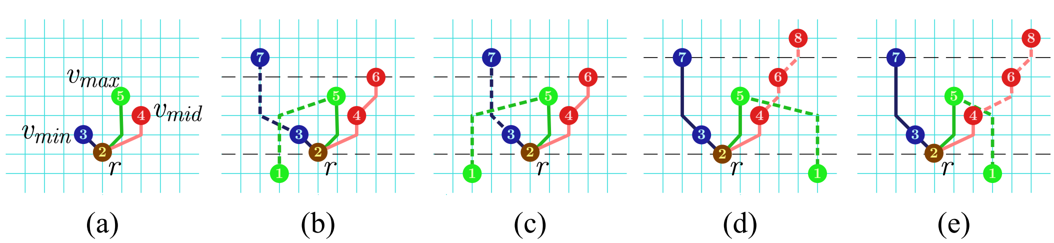


Figure 4.7: Four cases of expanding chains for $\phi(r) < \phi(v_{min}) < \phi(v_{mid}) < \phi(v_{max})$. All four can lead to crossings on the third expansion without taking precautions.

Algorithm: START-DRAWING-DEGREE-3-SPIDER

$\triangleright$ `Initially draw degree-3 spider until both invariants hold`

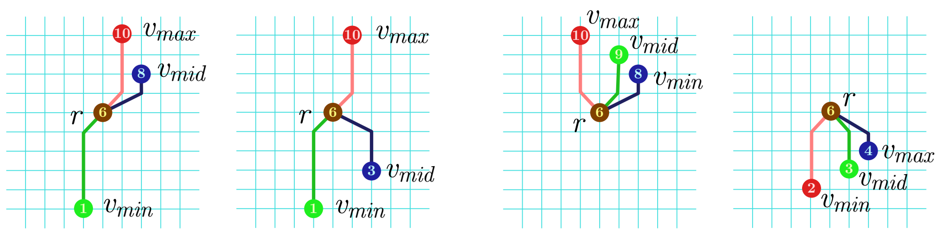
Input: Degree-3 spider $T(V, E, \phi)$ with distinct labels ϕ
Output: Degree-3 spider $T'(V', E')$ tree drawn so far

$r \leftarrow$ root of T $\triangleright$ `unique vertex of maximum degree`
Place r at $\big(0,\, \phi(r)\big)$
$\{v_{min}, v_{mid}, v_{max}\} \leftarrow N(r)$ $\triangleright$ `where` $\phi(v_{min}) < \phi(v_{mid}) < \phi(v_{max})$
$V' \leftarrow \{r\}$, $E' \leftarrow \varnothing$ $\triangleright$ $T'(V', E')$ `is the tree drawn so far`
if $\phi(v_{min}) < \phi(r) < \phi(v_{max})$ **then**
  DRAW-BENT-EDGE$\big(T,\, T',\, r,\, v_{min},\,$ LEFT$\big)$
  DRAW-BENT-EDGE$\big(T,\, T',\, r,\, v_{max},\,$ RIGHT$\big)$
  DRAW-BENT-EDGE$\big(T,\, T',\, r,\, v_{mid},\,$ RIGHT$\big)$
else
  $w_{extr} \leftarrow r$ $\triangleright$ `current most` ϕ-extreme `vertex`
  foreach $v \in N(r)$ **do**
    $w'_{extr} \leftarrow$ GET-EXTREME$(T,\, r,\, v)$
    if $\phi(r) \leq \phi(w_{extr}) < \phi(w'_{extr})$ **or** $\phi(r) \geq \phi(w_{extr}) > \phi(w'_{extr})$ **then**
      $w_{extr} \leftarrow w'_{extr}$, $v_{extr} \leftarrow v$
  $\{v_{left}, v_{right}\} \leftarrow N(r) \setminus \{v_{extr}\}$
  DRAW-BENT-EDGE$(T,\, T',\, r,\, v_{extr},\,$ RIGHT$)$ $\triangleright$ `draw` v_{extr}
  EXPAND-CHAIN$(T,\, T',\, v_{extr},\,$ RIGHT$,\, w_{extr})$
  DRAW-BENT-EDGE$(T,\, T',\, r,\, v_{right},\,$ RIGHT$)$ $\triangleright$ `draw` v_{right}
  DRAW-BENT-EDGE$(T,\, T',\, r,\, v_{left},\,$ LEFT$)$ $\triangleright$ `draw` v_{left}
  EXPAND-CHAIN$(T,\, T',\, v_{left},\,$ RIGHT$,\,$ **null**$)$
return $T'(V', E')$

Figure 4.8: START-DRAWING-DEGREE-3-SPIDER

Since v_{max} is the only ϕ-extreme leaf vertex in Fig. 4.7, either the chain of v_{mid} or v_{min} can be expanded next. However, Fig. 4.7(b) and (c) depict two cases in which the chain of v_{mid} is first expanded to the right leaving either v_{min} or v_{max} to be expanded next to the left. This leads to a crossing on the third expansion. Fig. 4.7(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.

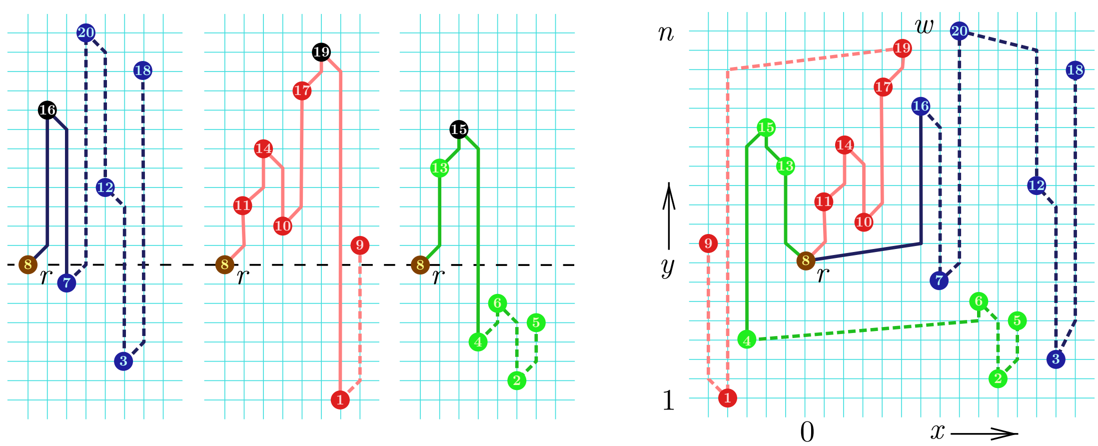
If $\phi(v_{min}) < \phi(r) < \phi(v_{max})$, then both invariants hold by placing v_{min} and v_{max} one unit to the left and to the right of r, respectively, and v_{mid} to the right of v_{max};



(a) (b)

Figure 4.9: Four initial cases for a degree-3 spider. Invariant (1) holds for the two initial degree-3 spiders in (a), but does not hold for the two in (b).

see Fig. 4.9(a). Otherwise, if all three vertices have labels less than or greater than r as in Fig. 4.9(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. 4.7. To avoid this, the chain C that reaches the most extreme point w_{extr} (of the three chains) 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. Fig. 4.10(a)–(c) illustrate determining this extreme point w_{extr} (which is the extreme of the chain in Fig. 4.10(b)), among the initial portions of the three chains drawn with solid edges. The solid edges differ in Fig. 4.10(d) by showing the initial part of the degree-3 spider that first satisfies both invariants.



(a) (b) (c) (d)

Figure 4.10: Determining the initial extreme used to start drawing a degree-3 spider

Algorithm: GET-EXTREME

$\triangleright$ `Find the next` ϕ`-extreme vertex of the chain that starts with vertex` u `before the chain cross the track of` r`.`

Input: Degree-3 spider $T(V, E, \phi)$ with distinct labels ϕ, root r, vertex u
Output: Extreme vertex of chain with vertex u

$w_{extr} \leftarrow u$ $\triangleright$ `set current extreme to` u
$v_{prev} \leftarrow r$ $\triangleright$ `previous vertex of the chain`
repeat
  $\{v\} \leftarrow N(u) \setminus \{v_{prev}\}$ $\triangleright$ `next vertex of the chain`
  Update $v_{prev} \leftarrow u$ and update $u \leftarrow v$
  if $\phi(v_{prev}) > \phi(r)$ **and** $\phi(v) > \phi(w_{extr})$ **then** update $w_{extr} \leftarrow v$
  if $\phi(v_{prev}) < \phi(r)$ **and** $\phi(v) < \phi(w_{extr})$ **then** update $w_{extr} \leftarrow v$
until $\phi(v_{prev}) < \phi(r) < \phi(v)$ **or** $\phi(v_{prev}) > \phi(r) > \phi(v)$ $\triangleright$ v `crosses` ℓ_r
return w_{extr}

Figure 4.11: GET-EXTREME

Let $v_{extr} \in \{v_{min}, v_{mid}, v_{max}\}$ be the initial vertex of chain C with the most extreme vertex w_{extr}. We first expand chain C to the right of r until C reaches w_{extr}. After expanding either of the other two chains to the left so that its last placed vertex w_{left} becomes the other ϕ-extreme after crossing ℓ_r, invariant (1) holds (with w_{extr} 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.

START-DRAWING-DEGREE-3-SPIDER determines the initial extreme of a chain using algorithm GET-EXTREME given in Fig. 4.11. Expansion of a chain is then accomplished by algorithm EXPAND-CHAIN given in Fig. 4.12.

Finally, we consider how to draw each edge with a bend using the algorithm DRAW-BENT-EDGE given in Fig. 4.13. 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; see Fig. 4.14.

Algorithm: EXPAND-CHAIN

$\triangleright$ `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.`

Input: Degree-3 spider $T(V, E, \phi)$ with distinct labels ϕ, degree-3 spider $T'(V', E')$ drawn so far, vertex u, *direction*, vertex w (optional)
Output: Updated level planar drawing of T

$\phi_{min} \leftarrow \min\{\phi(v) : v \in V' \cup \{w\}\}$ $\triangleright$ `minimum label in` $V' \cup \{w\}$
$\phi_{max} \leftarrow \max\{\phi(v) : v \in V' \cup \{w\}\}$ $\triangleright$ `maximum label in` $V' \cup \{w\}$
repeat
  $\{v\} \leftarrow N(u) \setminus V'$
  DRAW-BENT-EDGE(T, T', u, v, *direction*)
  Update $u \leftarrow v$
until $\phi(v) > \phi_{max}$ **or** $\phi(v) < \phi_{min}$ **or** $v = w$ **or** $|N(v)| = 1$
return v $\triangleright$ `last vertex added to` T'

Figure 4.12: EXPAND-CHAIN

Algorithm: DRAW-BENT-EDGE

$\triangleright$ `Vertex` u `has been placed. Vertex` v `will be drawn to the right of` u `if` *direction* `is RIGHT, and to the left of` u`, otherwise.`

Input: Degree-3 spider $T(V, E, \phi)$ with distinct labels ϕ, degree-3 spider $T'(V', E')$ drawn so far, vertex u, vertex v, *direction*
Output: Updated level planar drawing of T

$x_{min} \leftarrow \min\{x_v : v \in V'\}$ $\triangleright$ `minimum` x`-coordinate of` T'
$x_{max} \leftarrow \max\{x_v : v \in V'\}$ $\triangleright$ `maximum` x`-coordinate of` T'
if *direction* = RIGHT **then** $x_v \leftarrow x_{max} + 1$ $\triangleright$ `if drawing edge rightward`
else $x_v \leftarrow x_{min} - 1$ $\triangleright$ `else drawing edge leftward`
if $\phi(u) < \phi(v)$ **then** $y_b \leftarrow \phi(u) + 1$ $\triangleright$ `if drawing edge upward`
else $y_b \leftarrow \phi(u) - 1$ $\triangleright$ `else drawing edge downward`
Place v at $(x_v, \phi(v))$, bend b at (x_v, y_b), and draw edges u–b and b–v
$V' \leftarrow V' \cup \{v\}$, $E' \leftarrow E' \cup \{(u, v)\}$ $\triangleright$ `update` T'

Figure 4.13: DRAW-BENT-EDGE

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

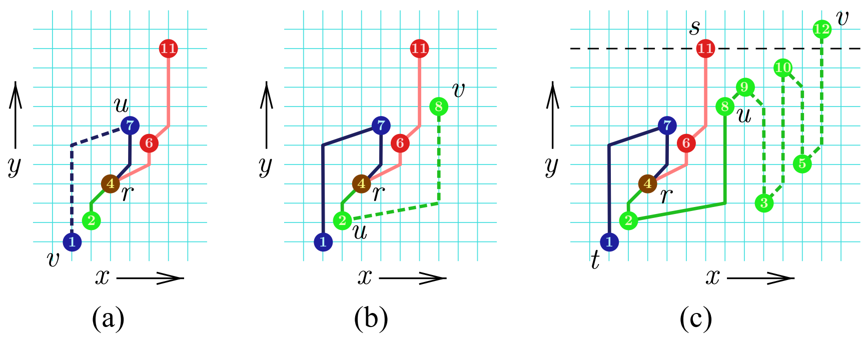


Figure 4.14: In (a) and (b), edge u–v is drawn to the left and to the right of u using DRAW-BENT-EDGE. In (c), the right chain is expanded to the right from u using EXPAND-CHAIN until v replaces s as the ϕ-maximum.

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. 4.14(a) and (b). 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 4.3.2. *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. Figure 4.15 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.

The algorithms of Lemma 4.3.1 can be modified to use straight-line edges with drawing algorithm DRAW-STRAIGHT-EDGE in Fig. 4.16 in lieu of

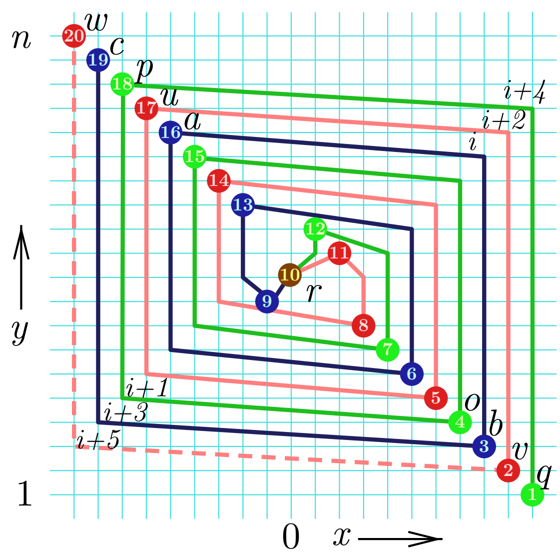


Figure 4.15: Degree-3 spider that can require $O(n!) \times n$ space when realized with no bends.

DRAW-BENT-EDGE in Fig. 4.13. 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. 4.15 in which edges a–b, o–p, u–v, b–c, p–q, and v–w are drawn in steps i, $i+1$, ..., $i+5$, respectively. For the last edge v–w in step $i+5$ not

Algorithm: DRAW-STRAIGHT-EDGE

▷ `Vertex` u `has been placed.  Vertex` v `will be drawn to the right of` u `if` *direction* `is RIGHT, and to the left of` u`, otherwise.`

Input: Degree-3 spider $T(V, E, \phi)$ with distinct labels ϕ, degree-3 spider $T'(V', E')$ drawn so far, vertex u, vertex v, *direction*

Output: Updated level planar drawing of T

$x_{min} \leftarrow \min\{x_v : v \in V'\}$, $x_{max} \leftarrow \max\{x_v : v \in V'\}$ ▷ `min/max` x`-coords`

if *direction* = RIGHT **then**

  Let $x_v > x_{max}$ be the least integer such that u–v does not intersect T'

else

  Let $x_v < x_{min}$ be the greatest integer such that u–v doest not intersect T'

Place v at $\big(x_v, \phi(v)\big)$ and draw edge u–v

$V' \leftarrow V' \cup \{v\}$, $E' \leftarrow E' \cup \{(u, v)\}$ ▷ `update` T'

Figure 4.16: DRAW-STRAIGHT-EDGE

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

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

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

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

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. $\square$

Combining Lemmas 4.1.1, 4.2.1, 4.3.1, and 4.3.2, we have our next theorem.

Theorem 4.3.3. *Caterpillars, radius-2 stars, and degree-3 spiders are all* ULP *with distinct labels. Each can be realized in $O(n)$ time.*

4.4 Drawing Caterpillars with Duplicate Labels

In this section, we show how to draw caterpillars for any duplicate labeling as in Fig. 4.17. We extend Lemma 4.1.1 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

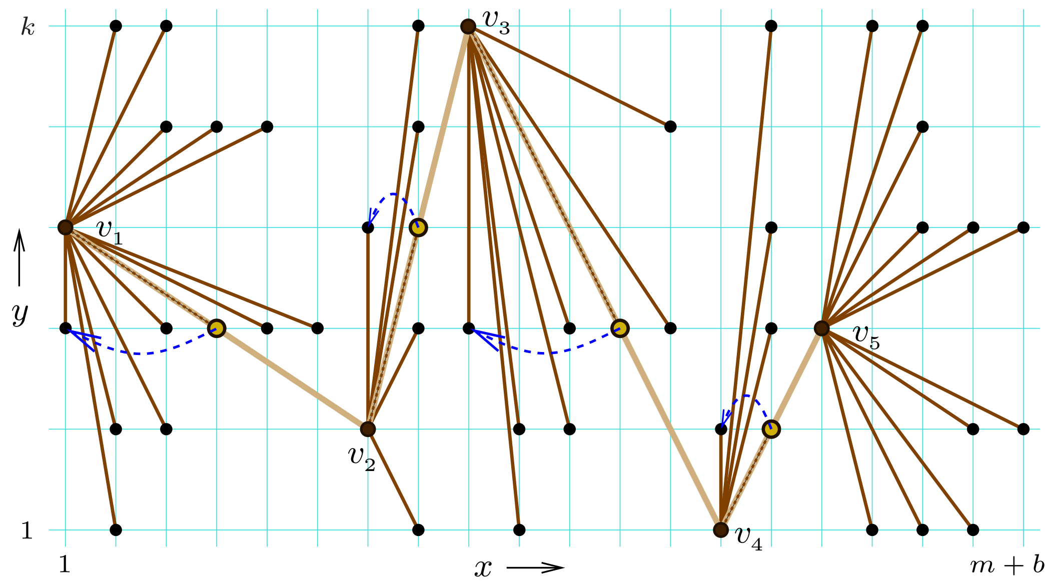


Figure 4.17: A realization of a 45-vertex caterpillar with duplicate labels on a 20×6 grid. Arrows indicate how vertices placed on spine edges are moved to avoid edge overlaps.

labels greater than and less 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 4.4.1. *An n-vertex caterpillar T 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 $b = \sum_{i=1}^{m} \max\left\{Dup\big(L_{above}(v_i)\big),\, Dup\big(L_{below}(v_i)\big)\right\}$.*

Proof. If $n \leq 2$, then T is easily drawn. Otherwise, T has a spine v_1–v_2–$\cdots$–v_m that we draw left to 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 using algorithm Place-Caterpillar-Leaves given in Fig. 4.18. Drawing the spine edge v_i–v_{i+1} with the leaf vertices of v_i takes a total of $\Big(1 + \max\big\{Dup\big(L_{above}(v_i)\big), Dup\big(L_{below}(v_i)\big)\big\}\Big) \times k$ space.

We place v_{i+1} at the next x-coordinate to the right of the leaf vertices of v_i; see Fig. 4.17. Since all 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

Algorithm: PLACE-CATERPILLAR-LEAVES

▷ `Place the leaves adjacent to a spine vertex` v_i `at` $\big(x_i, \phi(v_i)\big)$

Input: Caterpillar $T(V, E, \phi)$ with duplicate labels ϕ, vertex v_i, list L_i of leaves in $N(v_i)$ sorted by ϕ

Output: Maximum x-coordinate x_{max} of all leaves in L_i

$x_{above} \leftarrow x_{below} \leftarrow x + 1$
$\ell_{prev} \leftarrow \varnothing$, $\ell_{above} \leftarrow$ first element of L_i ▷ `previous/next leaf above` v_i
while $\ell_{above} \neq \varnothing$ **and** $\phi(\ell_{above}) > \phi(v_i)$ **do**
  Delete ℓ_{above} from L_i
  if $\phi(\ell_{above}) = \phi(\ell_{prev})$ **then** ▷ `if previous leaf on same level`
    $x_{above} \leftarrow x_{above} + 1$ ▷ `increment` x`-coord. for leaves above` v_i
  Place ℓ_{above} at $\big(x_{above},\, \phi(\ell_{above})\big)$
  $\ell_{prev} \leftarrow \ell_{above}$, $\ell_{above} \leftarrow$ first element of L_i
$\ell_{prev} \leftarrow \varnothing$, $\ell_{below} \leftarrow$ last element of L_i ▷ `previous/next leaf below` v_i
while $\ell_{below} \neq \varnothing$ **and** $\phi(\ell_{below}) < \phi(v_i)$ **do**
  Delete ℓ_{below} from L_i
  if $\phi(\ell_{below}) = \phi(\ell_{prev})$ **then** ▷ `if previous leaf on same level`
    $x_{below} \leftarrow x_{below} + 1$ ▷ `increment` x`-coord. for leaves below` v_i
  Place ℓ_{below} at $\big(x_{below},\, \phi(\ell_{below})\big)$
  $\ell_{prev} \leftarrow \ell_{below}$, $\ell_{below} \leftarrow$ last element of L_i
return $x_{max} \leftarrow \max\{x_{above}, x_{below}\}$

Figure 4.18: PLACE-CATERPILLAR-LEAVES with duplicate labels

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. Figure 4.19 gives the DRAW-CATERPILLAR algorithm for the case of duplicate labels.

Figure 4.20 gives the SORT-CATERPILLAR-LEAVES algorithm. The counting sort of L consists of a linear-time radix sort on the leaf vertices. This consists of two calls made to counting sort that each take $\Theta(n+k)$ time, sorting the adjacency lists of all the leaf vertices simultaneously. The first call sorts by neighbors so that $\ell_1 < \ell_2$ if $N(\ell_1) < N(\ell_2)$ where $N(\ell) = v_i$ if $\ell \in N(v_i)$ and $v_i < v_j$ if and only if $i < j$. The second call stably sorts by ϕ so that $\ell_1 < \ell_2$ if $N(\ell_1) = N(\ell_2)$ and $\phi(\ell_1) < \phi(\ell_2)$. Otherwise, it would take $\Theta(m(n+k))$ time if the lists were sorted

Algorithm: DRAW-CATERPILLAR

▷ `Draw a caterpillar with duplicate labels in` $O(n)$ `time`

Input: Caterpillar $T(V, E, \phi)$ with duplicate labels ϕ where $|V| > 2$
Output: Level planar drawing of T

$S \leftarrow$ REMOVE-LEAVES(T) ▷ `spine` $v_1 — \cdots — v_m$
$L_1, \ldots, L_m \leftarrow$ SORT-CATERPILLAR-LEAVES(T, S) ▷ `where` $L_i \subseteq N(v_i)$
$x_1 \leftarrow 1$, place v_1 at $\big(x_1, \phi(v_1)\big)$ ▷ x_i `is` x`-coordinate of` v_i
for $i \leftarrow 1$ *to* m **do**
    $x_{max} \leftarrow$ PLACE-CATERPILLAR-LEAVES(T, v_i, L_i)
    if $i < m$ **then**
        $x_{i+1} \leftarrow x_{max} + 1$
        Place v_{i+1} at $\big(x_{i+1}, \phi(v_{i+1})\big)$ and draw spine edge v_i–v_{i+1}
    foreach $\ell \in L_i$ **do**
        if $i < m$ **and** ℓ *lies on spine edge* v_i–v_{i+1} **then** move ℓ to $\big(x_i, \phi(\ell)\big)$
        Draw edge v_i–ℓ

Figure 4.19: DRAW-CATERPILLAR with duplicate labels

Algorithm: SORT-CATERPILLAR-LEAVES

▷ `Sort all the leaves of a caterpillar in` $O(n)$ `time`

Input: Caterpillar $T(V, E, \phi)$ with duplicate labels ϕ, spine $S = v_1 \cdots v_m$
Output: Lists $L_1, \ldots, L_m$ of leaves sorted by *phi* where $L_i \subseteq N(v_i)$

$L \leftarrow$ leaves of T
Perform a counting sort on the leaves L of T such that for each $\ell_1, \ell_2 \in L$, ℓ_1 proceeds ℓ_2 if $N(\ell_1) < N(\ell_2)$ or if $N(\ell_1) = N(\ell_2)$ and $\phi(\ell_1) < \phi(\ell_2)$
    ▷ `where` $N(\ell) = v_i$ `if` $\ell \in N(v_i)$ `and` $v_i < v_j$ `if and only if` $i < j$
for $i = 1$ *to* m **do** $L_i \leftarrow \varnothing$
$i \leftarrow 1$
repeat
    $\ell \leftarrow$ first element of L, delete ℓ from L
    while $\ell \notin N(v_i)$ **do** $i \leftarrow i + 1$
    $L_i \leftarrow L_i \cup \{\ell\}$
until $L = \varnothing$
return $L_1, \ldots, L_m$

Figure 4.20: SORT-CATERPILLAR-LEAVES with duplicate labels

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. □

We observe that an n-vertex caterpillar with an m-vertex spine has m internal vertices and $n-m$ leaves. Each leaf contributes at most once to the overall summation $b = \sum_{i=1}^{m} \max\left\{Dup\big(L_{above}(v_i)\big),\, Dup\big(L_{below}(v_i)\big)\right\}$ of the previous lemma, since $Dup\big(U\big) \leq |U|$ for any vertex subset U. Hence, the $(m+b) \times k$ grid used by the lemma is at worst an $n \times k$ grid (when the caterpillar is simply a path).

This allows us to restate Lemma 4.1.1 as follows:

Corollary 4.4.2. *Caterpillars of order n on k levels are* ULP *for any $0 \leq k \leq n$. Each can be straight-line realized in $O(n)$ time within an $n \times k$ grid for any labeling.*

Chapter 5

Characterizing ULP Trees

In this chapter, we begin by presenting labelings that force crossings in level planar drawings of the forbidden ULP trees T_7, T_8, and T_9; see Figs. 5.1 and 5.2. Then, we show that these forbidden trees are minimal in that the removal of any edge yields one or more ULP trees. We conclude the chapter with our characterizations of ULP trees, first for distinct labels, and then for duplicate labels.

5.1 Labelings of Forbidden ULP Trees

We next introduce the forbidden subtrees T_8 and T_9 shown in Fig. 5.1.

Lemma 5.1.1. *There exist distinct labelings that prevent T_8 and T_9 from being level planar.*

Proof. First, we consider T_8 in Fig. 5.1(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

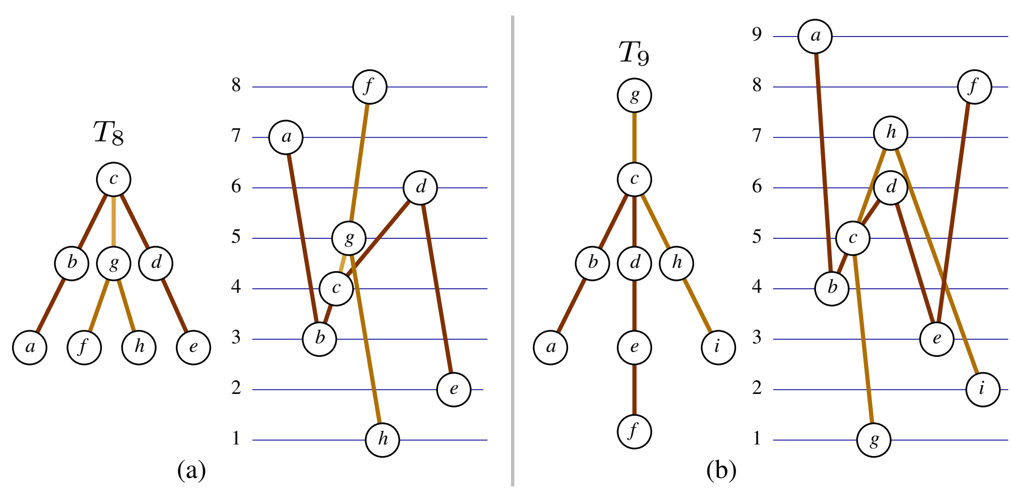


Figure 5.1: Distinct labelings preventing T_8 and T_9 from being ULP

contend that these labelings are level non-planar. 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 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. 5.1(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. 5.1(b). □

The forbidden tree T_7 in Fig. 5.2 is not ULP with duplicate labels for the given labelings that force a self intersection.

Lemma 5.1.2. *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 $\phi(a) = \phi(c) = \phi(f) = \phi(e) > \phi(b) = \phi(d) = \phi(g)$.

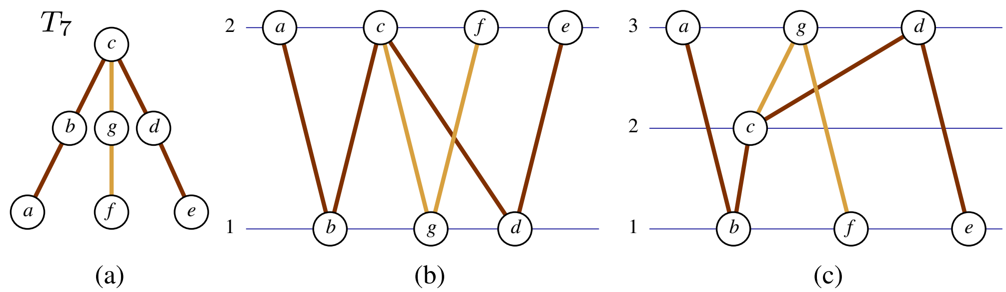


Figure 5.2: Duplicate labelings preventing T_7 from being ULP.

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 $2 < k \leq n$, 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. $\square$

The next lemma provides a tool to extend a forbidden labeling to a homeomorphic subgraph.

Lemma 5.1.3. *If a graph G contains a subgraph homeomorphic to a graph $\tilde{G}$ with a level non-planar labeling, then G also has a level non-planar labeling.*

Proof. Let $\tilde{G}$ be the level planar graph, and let $\tilde{\phi}$ be the labeling that forces a self crossing. We provide a labeling ϕ for G. Let h be the homeomorphism that maps an edge in $\tilde{G}$ to the path in G and a vertex in $\tilde{G}$ to the endpoint of the path in G. Label the vertices of $\tilde{G}$ using the labeling $\tilde{\phi}$.

We maintain the same relative ordering of the labels in G as in $\tilde{G}$. In particular, we want $\phi\big(h(u)\big) < \phi\big(h(v)\big)$ if and only if $\tilde{\phi}(u) < \tilde{\phi}(v)$ for each edge (u, v) in $\tilde{G}$.

For each path $h\big((u,v)\big) = p_{(u,v)} = v_1–v_2–\cdots–v_k$ in G that corresponds to an edge (u,v) in $\tilde{G}$, we want $\phi(v_1) < \phi(v_2) < \cdots < \phi(v_k)$ if $\tilde{\phi}(u) < \tilde{\phi}(v)$. We can assign the other vertices of G not in the image of h arbitrary labels. Then every edge (u,v) in $\tilde{G}$ corresponds to a strictly monotone path $p_{(u,v)}$ in G preserving the level non-planarity of the realization of $\tilde{G}$. □

Lemma 5.1.3 allows us to generalize Lemmas 5.1.1 and 5.1.2 into the following pair of corollaries:

Corollary 5.1.4. *If a tree contains a subtree homeomorphic to T_8 or T_9, then it cannot be ULP with distinct labels.*

Corollary 5.1.5. *If a tree contains a subtree isomorphic to T_7, then it cannot be ULP with duplicate labels.*

5.2 Minimality of Forbidden ULP Trees

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

Lemma 5.2.1. *Removing an edge from T_8 or T_9 yields a forest of ULP trees.*

Proof. If removing an edge from T_8 of Fig. 5.1(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. 5.1(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. □

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

Lemma 5.2.2. *Removing an edge from T_7 yields a forest of caterpillars.*

Proof. Removing an edge from T_7 incident to its root c; see Fig. 5.2(a), leaves a path and a lone edge. Otherwise, removing an edge leaves a caterpillar. $\square$

5.3 Characterizing ULP Trees with Distinct Labels

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

Theorem 5.3.1. *A tree T is a caterpillar, a radius-2 star, or a degree-3 spider if and only T does not contains a subtree homeomorphic to T_8 or T_9.*

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. 5.3(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 5.2.1, 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 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 4.3.3.

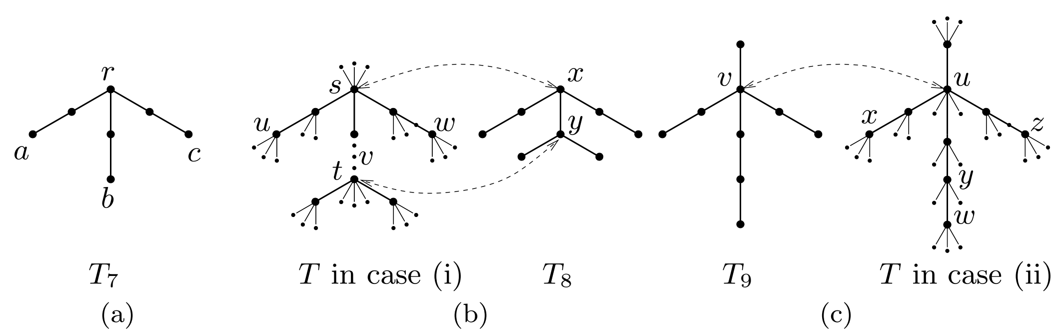


Figure 5.3: Homeomorphic copies of T_8 and T_9 in trees for Theorem 5.3.1.

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. 5.3(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. 5.3(c). Otherwise, if there were separate vertices of degree greater than 3, case (i) would apply. Let u be mapped to the degree4 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. □

Corollary 5.1.4 states that any tree containing a subtree homeomorphic to T_8 or T_9 has a labeling that forces a crossing. Theorem 4.3.3 states that each of the three classes of ULP trees have level planar drawings for any distinct labeling. Theorem 5.3.1 completes the characterization by stating that all trees either contain a subtree homeomorphic to T_8 or T_9 or belong to one of the three classes of ULP trees. Together these results can be summarized with our main theorem characterizing ULP trees with distinct labels.

Theorem 5.3.2. *For tree T, 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 with distinct labels.*

5.4 Characterizing ULP Trees with Duplicate Labels

Theorem 5.4.1. *A tree T is a caterpillar if and only if T does not contains a subtree isomorphic to T_7.*

Proof. Suppose that T is a tree. Let T' be the subtree of T that remains after removing all the leaves of T. If $\Delta(T') \leq 2$, then T' must be a path, and hence, T is a caterpillar by definition. Otherwise, there exists a vertex $r \in V(T')$ such that $deg(r) \geq 3$ in T' that is the endpoint of three incident edges e'_1, e'_2, and e'_3 in T'. Let $p_i := e'_i + e_i$ be a path in T for $i \in \{1, 2, 3\}$, where $e'_i \neq e_j$ for any $j \in \{1, 2, 3\}$ and e'_i is incident to e_j if and only if $i = j$. The three paths are internally disjoint, and hence, are isomorphic to T_7. Now suppose that T contains a subgraph H isomorphic to T_7. After removing all leaves, from T, to obtain the tree T', the degree-3 vertex in H must still remain in T', and have degree of at least 3 in T'. Hence, T cannot be a path, and thus, T is not a caterpillar. □

Corollary 5.1.5 states that any tree containing a subtree isomorphic to T_7 has a labeling that forces a crossing. Corollary 4.4.2 states that caterpillars are ULP with duplicate labels by having level planar drawings for any labeling. Theorem 5.4.1 completes the characterization by stating that all trees either contain a subtree isomorphic to T_7 or is a caterpillar. Together these results can be summarized with our main theorem characterizing ULP trees with duplicate labels.

Theorem 5.4.2. *For tree T, 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.*

Chapter 6

Certifying ULP Trees

In order to certify that a tree is either ULP or not, we need a certificate. If the graph is ULP, then the respective drawing algorithm given in chapter 4 provides a positive certificate in terms of a level planar drawing. However, if the graph is not ULP, then we need to find the forbidden subdivision in the tree that corresponds to one of the forbidden ULP trees, which we do first for T_7, and then for T_8 and T_9.

6.1 Finding Forbidden ULP Tree T_7

If a tree is not ULP with duplicate labels, then we know by Theorem 5.4.2 that the tree must contain a subtree isomorphic to the forbidden tree T_7. The next theorem describes the certification algorithm Find-T_7-Subtree given in Fig. 6.1 that shows how to find a T_7 subtree in linear time.

Algorithm: Find-T_7-Subtree
$\triangleright$ `Find a` T_7 `subtree in` $O(n)$ `time`

Input: Tree $T(V, E)$
Output: Return T_7 subtree in T if one exists, $\varnothing$ otherwise

if Is-Caterpillar(T) **then return** $\varnothing$
$T' \leftarrow$ Remove-Leaves(T)
$V'_{deg\geq 3} \leftarrow \{v : v \in V(T') \text{ and } deg_{T'}(v) \geq 3\}$ $\triangleright$ `degree` $\geq$ `3 vertices of` T'
Let $r \in V'_{deg\geq 3}$ $\triangleright$ `root of` T_7
Let $\{a, s, x\} \subseteq N_{T'}(r)$ $\triangleright$ `any three neighbors of` r `in` T'
Let $b \in N_T(a) \setminus \{r\}$ $\triangleright$ `any neighbor of` a `in` T `other than` r
Let $t \in N_T(s) \setminus \{r\}$ $\triangleright$ `any neighbor of` s `in` T `other than` r
Let $y \in N_T(x) \setminus \{r\}$ $\triangleright$ `any neighbor of` x `in` T `other than` r
return $T[\{r, a, b, s, t, x, y\}]$ $\triangleright$ `return induced subtree of` T

Figure 6.1: Find-T_7-Subtree

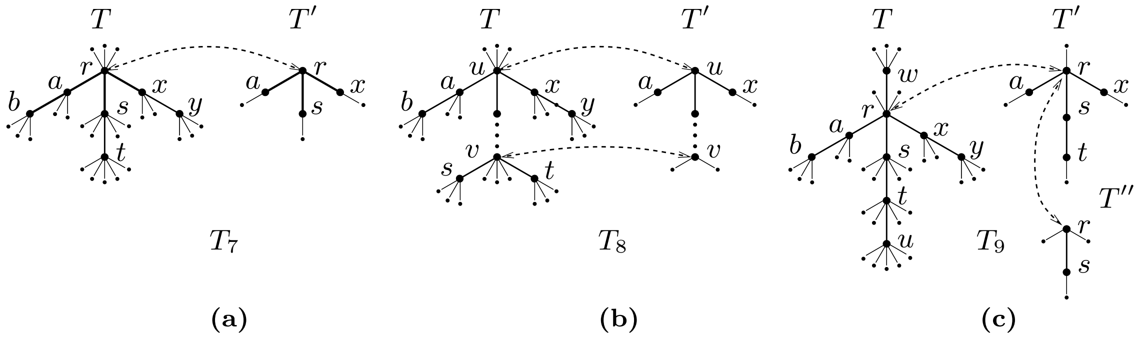


Figure 6.2: Find T_7, T_8 and T_9 in T by removing the leaf vertices in T to get the subtree T' in (a), (b) and (c), and repeat this process with T' to get the subtree T'' in (c).

Theorem 6.1.1. *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 Theorem 5.4.2, 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. 6.1(a). This allows us to find a subtree isomorphic to T_7 in $O(n)$ time using the algorithm FIND-T_7-SUBTREE given in Fig. 6.1. □

6.2 FINDING FORBIDDEN ULP TREES T_8 AND T_9

If a tree is not ULP with distinct labels, then we know by Theorems 5.3.2 that the tree must contain a subtree homeomorphic to either T_8 or T_9. The next theorem describes the certification algorithms FIND-T_8-SUBDIVISION and FIND-T_9-SUBDIVISION given in Figs. 6.3 and 6.4 that show how to find a T_8 or a T_9 subdivision in linear time.

Theorem 6.2.1. *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 Theorem 5.3.1, 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

Algorithm: FIND-T_8-SUBDIVISION
▷ `Find a` T_8 `subdivision in` $O(n)$ `time`
Input: Tree $T(V, E)$
Output: Return T_8 subdivision in T if one exists, $\varnothing$ otherwise

$T' \leftarrow$ REMOVE-LEAVES(T)
$V'_{deg\geq 3} \leftarrow \{v : v \in V(T') \text{ and } deg_{T'}(v) \geq 3\}$ ▷ `degree` $\geq$ `3 vertices of` T'
if $V'_{deg\geq 3} = \varnothing$ **then return** $\varnothing$
Let $u \in V'_{deg\geq 3}$ ▷ `any vertex of degree` $\geq$ `3 in` T'
$V_{deg\geq 3} \leftarrow \{v : v \in V(T) \text{ and } deg_T(v) \geq 3\}$ ▷ `degree` $\geq$ `3 vertices of` T
if $V_{deg\geq 3} = \{u\}$ **then return** $\varnothing$
Let $v \in V_{deg\geq 3} \setminus \{u\}$ ▷ `any other vertex of degree` $\geq$ `3 in` T
Let p be the unique path u to v in T
Let $\{s, t\} \subseteq N_T(v) \setminus V(p)$ ▷ `any neighbors of` v `in` T `not on` p
Let $\{a, x\} \subseteq N_{T'}(u) \setminus V(p)$ ▷ `any neighbors of` u `in` T' `not on` p
$b \leftarrow N_T(a) \setminus \{u\}$ ▷ `any neighbor of` a `in` T `other than` u
$y \leftarrow N_T(x) \setminus \{u\}$ ▷ `any neighbor of` x `in` T `other than` u
return $T[\{a, b, s, t, x, y\} \cup V(p)]$ ▷ `return induced subtree of` T

Figure 6.3: FIND-T_8-SUBDIVISION

any vertex u of degree 3 or more in T' (the subtree of T obtained by removing all of its leaf vertices); see Fig. 6.1(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 using the algorithm FIND-T_8-SUBDIVISION in Fig. 6.3.

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. 6.1(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

Algorithm: FIND-T_9-SUBDIVISION

$\triangleright$ `Find a` T_9 `subdivision in` $O(n)$ `time`

Input: Tree $T(V, E)$
Output: Return T_9 subdivision in T if one exists, $\varnothing$ otherwise

$T' \leftarrow$ REMOVE-LEAVES(T)
$T'' \leftarrow$ REMOVE-LEAVES(T')
$R \leftarrow \{v : v \in V(T''), deg_{T'}(v) \geq 3\}$ $\triangleright$ `degree` $\geq$ `3 vertices of` T' `in` T''
if $R = \varnothing$ **then return** $\varnothing$
Let $r \in R$ $\triangleright$ `root of` T_9 `subtree`
if $N_{T''}(r) = \varnothing$ **then return** $\varnothing$
Let $s \in N_{T''}(r)$ $\triangleright$ `any neighbor of` r `in` T''
Let $t \in N_{T'}(s) \setminus \{r\}$ $\triangleright$ `any neighbor of` s `(other than` r`) in` T'
Let $u \in N_T(t) \setminus \{s\}$ $\triangleright$ `any neighbor of` t `(other than` s`) in` T
Let $\{a, x\} \subseteq N_{T'}(r) \setminus \{s\}$ $\triangleright$ `any neighbors of` r `(other than` s`) in` T'
Let $b \in N_T(a) \setminus \{r\}$ $\triangleright$ `any neighbor of` a `(other than` r`) in` T
Let $y \in N_T(x) \setminus \{r\}$ $\triangleright$ `any neighbor of` x `(other than` r`) in` T
Let $w \in N_T(r) \setminus \{a, s, x\}$ $\triangleright$ `any neighbor of` r `(other than` a, s, x`) in` T
return $T[\{a, b, r, s, t, u, w, x, y\}]$ $\triangleright$ `return induced subtree of` T

Figure 6.4: FIND-T_9-SUBDIVISION

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 using the algorithm FIND-T_9-SUBDIVISION in Fig. 6.4. $\square$

Chapter 7

Drawing ULP Graphs

In section 4.2, we showed that radius-2 stars (R2S) are level planar for any distinct labeling. In this chapter, we show that this also holds for generalized caterpillars (GC), extended 3-spiders (E3S), and extended K_4 subgraphs (EK4). In section 4.4, we showed that caterpillars have level planar drawings for any duplicate labeling. In this chapter, we show that this also holds for K_3-caterpillars and G_ω.

7.1 Drawing Generalized Caterpillars with Distinct Labels

The next lemma show how a GC has a compact planar realization on tracks.

Lemma 7.1.1. *An n-vertex generalized caterpillar G with b ULP blocks where d of the blocks are diamond blocks can be straight-line realized in $O(n)$ time on a $2(b + d) \times n$ grid for any distinct labeling.*

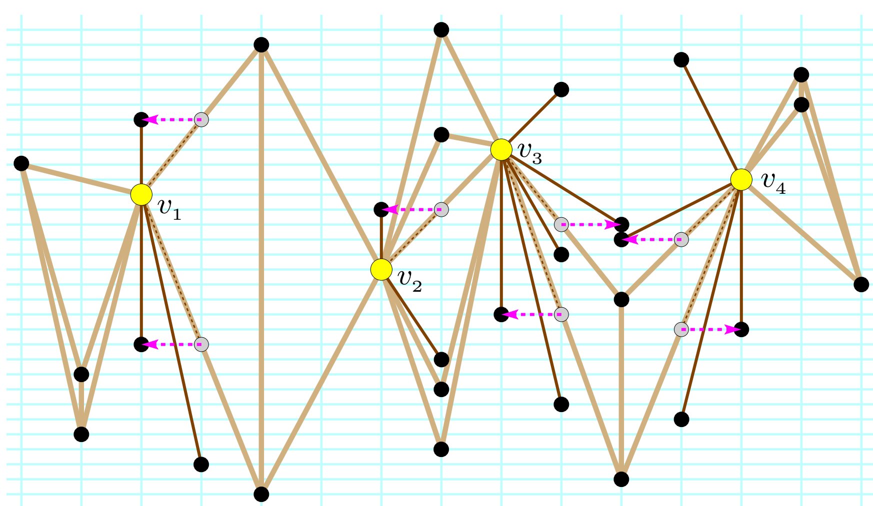


Figure 7.1: A straight-line realization of a 32-vertex GC with 5 ULP blocks (where 2 are diamond blocks) on a $2(5 + 2) \times 32 = 14 \times 32$ grid. Leaves are initially placed to the right of their cut vertices, except for the leaves placed to the left of the last cut vertex. Overlaps of gray leaves with edges are eliminated by moving the leaves left or right.

Proof. Conceptually drawing a GC is simple. If we were not constrained to an integer grid, we could draw each of the ULP joining and ending blocks of G from left to right. In doing so, each cut-vertex of the spine S of the caterpillar T that was used to construct G would proceed right to left in the order that the cut-vertices appear along S. We could then place the remaining leaf edges in a sufficiently narrow region above and below each cut vertex. Being restricted to integer coordinates requires significantly more work in order to obtain compact planar drawings where no leaf vertex or edge overlaps any of the ULP joining or ending blocks; see Fig. 7.1.

First, we show how to draw each of the non-trivial blocks of G compactly, and then we show how to draw G block by block. There are three distinct categories of ULP blocks of a GC: (i) K_4 blocks, (ii) diamond blocks, and (iii) $(K_3)^*$ or $(C_4)^*$ blocks. Each category of ULP blocks presents it own unique challenges.

We begin by showing how to draw a K_4 block B on a $2\times n$ grid using the algorithm Draw-K_4-Block given in Fig. 7.2. The difficulty here is correctly choosing the number of units to the right or left of the connector u to place the other three vertices of B. One vertex v can be placed two units to the left or to the right of

Algorithm: Draw-K_4-Block

▷ `Draw a` K_4 `block on` n `levels on a` $2 \times n$ `grid in` $O(1)$ `time to the left of the connector` u `if` *direction* `is LEFT and to the right of` u`, otherwise.`

Input: K_4 block $B(V, E, \phi)$ with distinct labels ϕ, connector u, x-coordinate x_u of u, *direction*
Output: Updated level planar drawing

if *direction* $=$ RIGHT **then** $dir \leftarrow +1$ **else** $dir \leftarrow -1$
$\{v_{min}, v_{mid}, v_{max}\} \leftarrow N(u)$ where $\phi(v_{min}) < \phi(v_{mid}) < \phi(v_{max})$
if $\phi(u) > \phi(v_{mid})$ **then**
| $v \leftarrow v_{max}$, $\{s, t\} \leftarrow \{v_{min}, v_{mid}\}$
else $v \leftarrow v_{min}$, $\{s, t\} \leftarrow \{v_{mid}, v_{max}\}$
$x_v \leftarrow x_u + 2 \cdot dir$, $x_s \leftarrow x_t \leftarrow x_u + dir$
Place u, v, s, t at $\big(x_u,\, \phi(u)\big)$, $\big(x_v,\, \phi(v)\big)$, $\big(x_s,\, \phi(s)\big)$, and $\big(x_t,\, \phi(t)\big)$, respectively. Draw edges u–v, u–s, u–t, v–s, v–t, and s–t.

Figure 7.2: Draw-K_4-Block

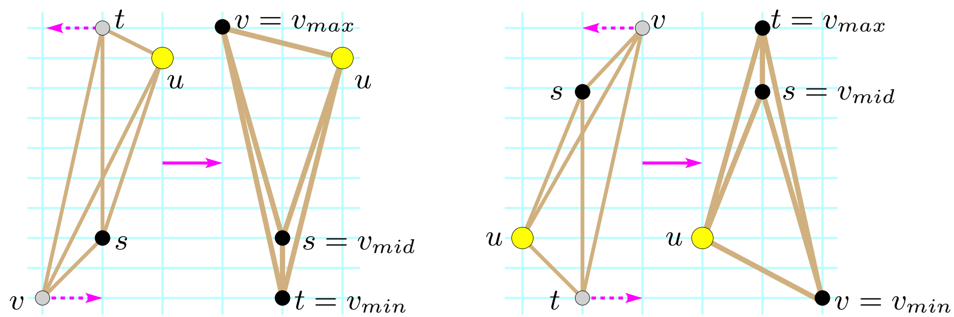


Figure 7.3: Avoiding edge crossings when drawing a K_4 block

u, and the other two vertices, s and t can be placed one unit to the left or to the right of u, respectively. To do this, we order the three vertices by ϕ and initially denote vertices by v_{min}, v_{mid}, and v_{max} where $\phi(v_{min}) < \phi(v_{mid}) < \phi(v_{max})$. If $\phi(u) > \phi(v_{mid})$, then we assign v to be v_{max}. Otherwise, we assign v to be v_{min}. This guarantees that the edge u–v will not to cross the edge s–t; see Fig. 7.3.

A diamond block B can be drawn using the algorithm Draw-Diamond-Block given in Fig. 7.4. All the other non-trivial blocks are drawn on a $2 \times n$ grid. Diamond blocks require a $4 \times n$ grid to allow sufficient room for leaves to be moved so as to avoid leaf edges overlapping any of the ULP ending or joining blocks; see Fig. 7.5.

Algorithm: Draw-Diamond-Block

▷ `Draw a diamond block on` n `levels on a` $4 \times n$ `grid in` $O(1)$ `time to the left of the connector` u `if` *direction* `is LEFT and to the right of` u`, otherwise.`

Input: Diamond block $B(V, E, \phi)$ with distinct labels ϕ, connector u, x-coordinate x_u of u, *direction*
Output: Updated level planar drawing

if *direction* = RIGHT **then** $dir \leftarrow +1$ **else** $dir \leftarrow -1$
$\{s, t\} \leftarrow \{v : v \in V \text{ and } deg(v) = 3\}$ ▷ `degree-3 vertices of` B
$\{v\} \leftarrow V \setminus \{u, s, t\}$
$x_v \leftarrow x_u + 4 \cdot dir$, $x_s \leftarrow x_t \leftarrow x_u + 2 \cdot dir$
Place u, v, s, t at $\big(x_u, \phi(u)\big)$, $\big(x_v, \phi(v)\big)$, $\big(x_s, \phi(s)\big)$, and $\big(x_t, \phi(t)\big)$, respectively. Draw edges u–s, u–t, v–s, v–t, and s–t.

Figure 7.4: Draw-Diamond-Block

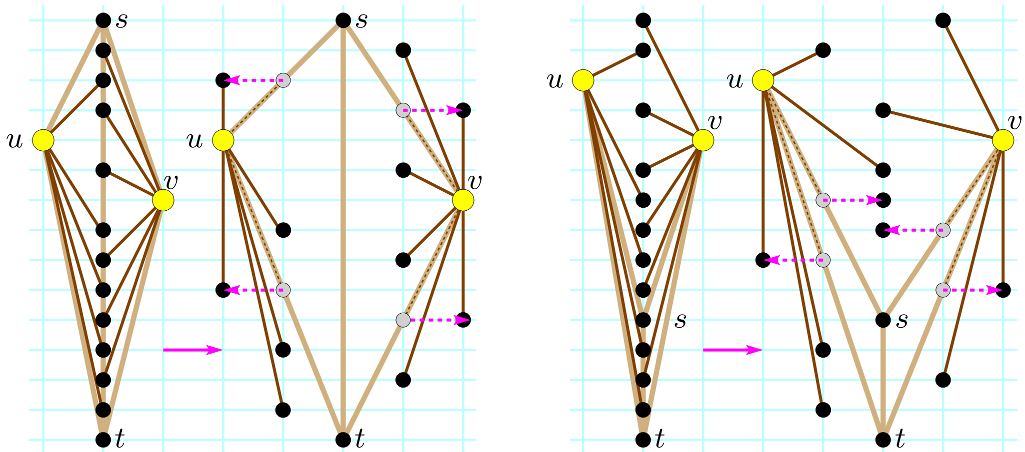


Figure 7.5: Avoiding edge crossings when drawing a diamond block

A k-vertex $(K_3)^*$ or a $(C_4)^*$ block B can be drawn in $O(k)$ time on a $2 \times n$ using the algorithm Draw-$(K_3)^*$-Or-$(C_4)^*$-Block given in Fig. 7.6.

The previous three drawing algorithms are combined into the algorithm

Algorithm: Draw-$(K_3)^*$-Or-$(C_4)^*$-Block

$\triangleright$ `Draw a` k`-vertex` $(K_3)^*$ `or` $(C_4)^*$ `block on` n `levels on a` $2 \times n$ `grid in` $O(k)$ `time to the left of the connector` u `if` *direction* `is LEFT and to the right of` u`, otherwise.`

Input: $(K_3)^*$ or $(C_4)^*$ block $B(V, E, \phi)$ with distinct labels ϕ, connector u, x-coordinate x_u of u, *direction*

Output: Updated level planar drawing

if Is-P_2-Block(B, u) **then** $\{v\} \leftarrow V \setminus \{u\}$
else if Is-K_3-Block(B, u) **then** $\{v, w_1\} \leftarrow V \setminus \{u\}$
else if Is-C_4-Block(B, u) **then** $\{v\} \leftarrow V \setminus N(u)$, $\{w_1, w_2\} \leftarrow V \setminus \{u, v\}$
else
  $\{w_1, \ldots, w_{k-2}\} \leftarrow \{v : v \in V \text{ and } deg(v) = 2\}$
  $\{v\} \leftarrow V \setminus \{u, w_1, \ldots, w_{k-2}\}$
if *direction* $=$ RIGHT **then** $dir \leftarrow +1$ **else** $dir \leftarrow -1$
$x_v \leftarrow x_u + 2 \cdot dir$, $x_w = x_u + dir$
Place u and v at $\big(x_u,\, \phi(u)\big)$ and $\big(x_v,\, \phi(v)\big)$
if Is-$(K_3)^*$-Block(B, u) **then** draw edge u–v
for $i \leftarrow 1$ *to* $k-2$ **do** place w_i at $\big(x_w,\, \phi(w_i)\big)$ and draw edges u–w_i and v–w_i

Figure 7.6: Draw-$(K_3)^*$-Or-$(C_4)^*$-Block

Algorithm: DRAW-ULP-BLOCK

▷ `Draw a` n`-vertex ULP block` B `on` n `levels on a` $2 \times n$ `or a` $4 \times n$ `(if` B `is a diamond block) grid in` $O(k)$ `time to the left of the connector` u `if` *direction* `is LEFT and to the right of` u`, otherwise.`

Input: ULP block $B(V, E, \phi)$ with distinct labels ϕ, connector u,
x-coordinate x_u of u, *direction*
Output: Updated level planar drawing

if *direction* = RIGHT **then** $dir \leftarrow +1$ **else** $dir \leftarrow -1$
if IS-K_4-BLOCK(B, u) **then**
  DRAW-K_4-BLOCK(B, u)
else if IS-DIAMOND-BLOCK(B, u) **then**
  DRAW-DIAMOND-BLOCK(B, u)
else
  DRAW-$(K_3)^*$-OR-$(C_4)^*$-BLOCK(B, u)

Figure 7.7: DRAW-ULP-BLOCK

DRAW-ULP-BLOCK given in Fig. 7.7 that can draw any ULP joining or ending block of a GC.

We proceed to draw G using the algorithm DRAW-GENERALIZED-CATERPILLAR given in Fig. 7.8. If G is biconnected, then G is a single block. We pick any vertex u of maximum degree so that DRAW-ULP-BLOCK(G, u) draws G.

Otherwise, G has one or more cut-vertices, which we can determine in linear time [81] and the blocks that connect them. Since all blocks of G are either 1-blocks or 2-blocks, we can replace each block with an edge to produce the caterpillar that was used to construct G in Definition 2.3.6.

Let $v_1 - \cdots - v_k$ denote the spine of T where $v_1, \ldots, v_k$ are the cut-vertices of G. If $k = 1$, then T is a star. By Definition 2.3.4, G has at most two ULP blocks ending on u, which we denote as B_0 and B_1. Else $k > 1$, and v_1 and v_k each have at most one ending ULP block, B_0 and B_k, respectively. Each 2-block B_i connects v_i to v_{i+1} for $i \in [1..k-1]$. If B_0 exists, then we draw B_0 to the left of the first cut vertex. Then we draw block B_i to the right of vertex v_i for $i \in [1..k]$.

We attempt to draw the leaves adjacent to v_i one unit to the right of v_i if $i < k$

```
Algorithm: DRAW-GENERALIZED-CATERPILLAR
▷ Draw a generalized caterpillar in O(n) time

Input: Generalized caterpillar G(V, E, φ) with distinct labels φ
Output: Level planar drawing of G

if G is biconnected then                              ▷ G only has one block
    u ← any vertex of maximum degree in G
    DRAW-ULP-BLOCK(G, u)
else
    𝓑 ← non-trivial blocks of G              ▷ either 1-blocks or 2-blocks
    T ← caterpillar created by replacing each block in 𝓑 with an edge
    v_1–···–v_k ← REMOVE-LEAVES(T)                     ▷ cut-vertices in G
    if k = 1 then                                        ▷ if T is a star
        {B_0, B_k} ← non-trivial 1-blocks of 𝓑        ▷ either may be empty
    else                                           ▷ else T is not a star
        B_0 ← non-trivial 1-block of 𝓑 with connector v_1    ▷ maybe be empty
        B_1, ..., B_{k-1} ← 2-blocks of G where B_i has connectors v_i and v_{i+1}
        B_k ← non-trivial 1-block of 𝓑 with connector v_k    ▷ maybe be empty

    x ← 0                                 ▷ x-coordinate of current cut-vertex
    if B_0 is non-empty then
        u ← any vertex of maximum degree in B_0
        if IS-DIAMOND-BLOCK(B_i, u) then x ← x + 4 else x ← x + 2
        DRAW-ULP-BLOCK(B_0, u, x, RIGHT)

    for i ← 1 to k do
        DRAW-ULP-BLOCK(B_i, u, LEFT)
        if i < k then  dir ← +1, B ← B_i else  dir ← −1, B ← B_{k-1}
        foreach leaf ℓ ∈ N(v_i) do
            Place leaf ℓ at (x + dir, φ(ℓ))
            if leaf ℓ lies on B then
                if IS-DIAMOND-BLOCK(B, u) then
                    Move leaf ℓ to (x + 2 · dir, φ(ℓ))
                    if leaf ℓ still lies on B then  move leaf ℓ to (x, φ(ℓ))
                else  move leaf ℓ to (x, φ(ℓ))

        Draw edge v_i–ℓ
        if IS-DIAMOND-BLOCK(B_i, u) then x ← x + 4 else x ← x + 2
```

Figure 7.8: DRAW-GENERALIZED-CATERPILLAR

and one unit to the left of v_k (in order to avoid any leaves overlapping a K_4 ending block). Suppose a leaf edge overlaps a block B. If B is not a diamond block, then we can always move the leaf above or below the adjacent cut vertex. However, if B is a diamond block, we first attempt to move the leaf above or below the vertical edge of B, if possible, and then above or below the adjacent cut vertex, otherwise.

Draw-Generalized-Caterpillar runs in $O(n)$ time since each vertex is placed and each edge is drawn in $O(1)$ time. Each of the b ULP blocks of G can be drawn on a $2 \times n$ grid except for the d diamond blocks that require an additional $2 \times n$ space. All leaves are drawn in the same grid space as the ULP blocks, so that Draw-Generalized-Caterpillar uses a total of $2(b + d) \times n$ space. □

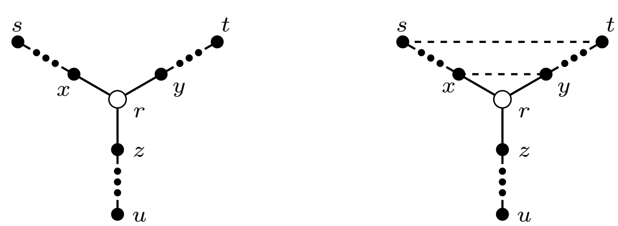
7.2 Drawing Extended 3-Spiders with Distinct Labels

The following lemma extends algorithm Draw-Degree-3-Spider of Lemma 4.3.1 to accommodate the extra edges in an extended 3-spider.

Lemma 7.2.1. *An n-vertex extended 3-spider G can be realized in $O(n)$ time on an $(n + 1) \times n$ grid for any distinct labeling.*

Proof. We extend the algorithm Draw-Degree-3-Spider from Fig. 4.6 in section 4.3 to accommodate the two extra possible edges in an extended 3-spider shown in Fig. 7.9 to produce realizations as in Fig. 7.10 with at most one bend per edge.

We start by extracting a spanning degree-3 spider T with root r from G using the algorithm Get-Spanning-Degree-3-Spider from Fig. 7.11, which we will use as the basis for drawing G. By Definition 2.3.1, the difference between G and T is



Degree-3 Spider (a) Extended 3-Spider (b)

Figure 7.9: A degree-3 spider in (a) is a spanning tree of an extended 3-spider in (b) that has up to two extra edges (x, y) and (s, t).

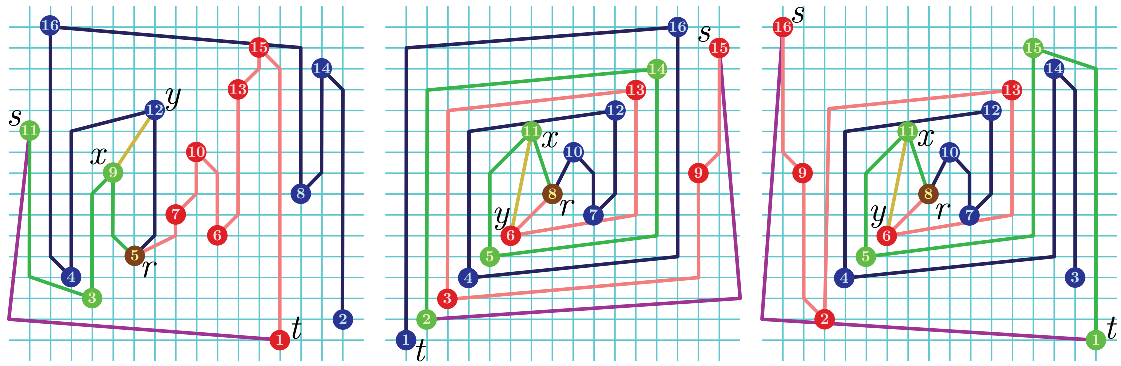


Figure 7.10: Example realizations of three different extended 3-spiders

at most two edges e_1 and e_2, where e_1 is the edge (x, y) connecting two neighbors x and y of r and e_2 is the edge connecting two of three leaves s, t, and u of T.

For a given root r, the edge e_1 is fixed, whereas, the edge e_2 can be any edge

Algorithm: GET-SPANNING-DEGREE-3-SPIDER

$\triangleright$ `Get a spanning degree-3 spider`

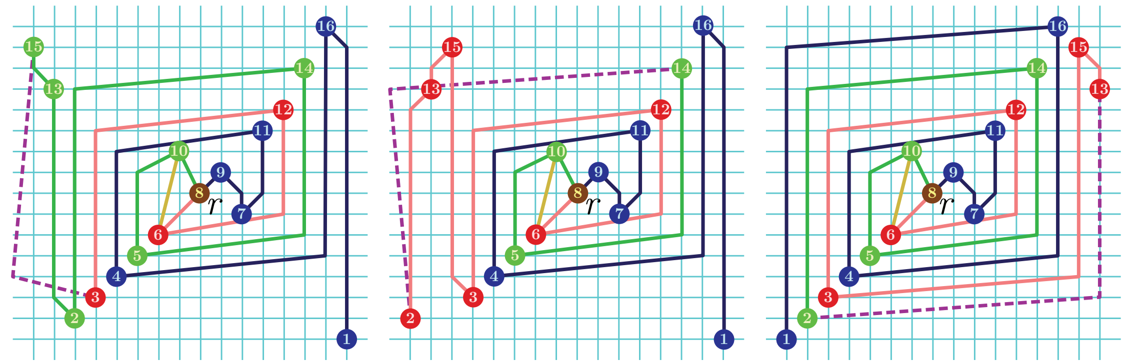
Input: Extended 3-spider $G(V, E, \phi)$ with distinct labels ϕ, root r
Output: Spanning degree-3 spider $T(V, E', \phi)$ of G

$E' \leftarrow E$ $\triangleright$ `edge set of spanning degree-3 spider`
$\{x, y, z\} \leftarrow N(r)$ **where** $deg(x) = deg(y) \geq deg(z)$
if $deg(x) = deg(y) = 3$ **then** $\triangleright$ `if` G `has edge` (x, y)
  Delete edge (x, y) from E'
if G *has no cycle* **then return** $T(V, E, \phi)$
$C \leftarrow$ cycle in G $\triangleright$ k`-cycle for some` $k \geq 4$ `found in` $O(n)$ `using DFS`
$\phi_{min}(C) \leftarrow \min\{\phi(v) : v \in V(C)\}$ $\triangleright$ `minimum label in` C
$\phi_{max}(C) \leftarrow \max\{\phi(v) : v \in V(C)\}$ $\triangleright$ `maximum label in` C
r–v_1–$\cdots$–v_{k-1}–$r \leftarrow C$ $\triangleright$ `where` $|C| = k \geq 4$

$i \leftarrow 1$, **repeat** $i \leftarrow i + 1$ **until** $\phi(v_i) = \phi_{min}(C)$ **or** $\phi(v_i) = \phi_{max}(C)$

if $i < k - 1$ **then** $\triangleright$ `if neither` y`-extreme of` C `is` r
  $e \leftarrow (v_i, v_{i+1})$ $\triangleright$ e `is not incident to` r
else $\triangleright$ `otherwise` r `must be one of the` y`-extremes of` C
  $e \leftarrow (v_i, v_{i-1})$ $\triangleright$ `pick other edge incident to` v_i `but not to` r
Delete edge e from E' and **return** $T(V, E', \phi)$

Figure 7.11: GET-SPANNING-DEGREE-3-SPIDER



(a) (b) (c)

Figure 7.12: Three realizations of the same extended 3-spider in which the dashed edge of the cycle 8–10–5–14–2–13–15–3–12–6–8 has been chosen to be drawn last. While all three dashed edges are incident to either vertex 2 or 15, the ϕ-extreme vertices of the cycle, the edge $(3, 15)$ in (a) and the edge $(2, 14)$ in (b) result in paths from the root 8 to an endpoint of the edge that contain both ϕ-extremes 2 and 15, which leads to a crossing. In (c), the edge $(2, 13)$ is chosen so that neither of the paths from 8 to the endpoints 2 and 13 of the edge contain both ϕ-extremes 2 and 15, which allows a level planar drawing.

along the k-cycle C (for some $k = |C| \geq 4$), r–v_1–$\cdots$–v_{k-1}–r, except for the edges (r, v_1) or (r, v_{k-1}). The algorithm Get-Spanning-Degree-3-Spider extracts a spanning degree-3 spider T from G where the edge e_2 omitted from C is carefully chosen. Suppose that v_{min} and v_{max} are the vertices of C with the minimum and maximum labels of C, respectively. In order to draw edge e_2 with endpoints s and t, the chosen edge (s, t) must meet three prerequisites:

(i) neither s or t can be r,

(ii) either s or t must be one of the two vertices v_{min} and v_{max},

(iii) neither the path $r \rightsquigarrow s$ nor the path $r \rightsquigarrow t$ can contain both v_{min} and v_{max} unless $r = v_{min}$ or $r = v_{max}$.

We need all three edges incident to r to be in T so as to draw our spanning degree-3 spider, which gives prerequisite (i). If prerequisite (ii) is not met, then after all the edges of the T have been drawn, s and t may lie on opposite sides of the path from $v_{min} \rightsquigarrow v_{max}$, which would lead to a crossing with the edge (s, t). Figure 7.12 illustrates how a crossing can result if the prerequisite (iii) is not met.

To ensure that we pick an edge that meets all three prerequisites, we scan through the vertices $v_1, v_2, \ldots, v_{k-1}$ of C in order starting from v_1 until we reach v_i some $i \in \{1, \ldots, k-1\}$ where $v_i = v_{min}$ or $v_i = v_{max}$. If $i < k-1$, then we can pick the edge (v_i, v_{i+1}), which satisfies (i), (ii), and (iii). Otherwise, $i = k-1$, which implies that $r = v_{min}$ or $r = v_{max}$ where prerequisite (iii) does not apply, so we must pick the edge (v_i, v_{i-1}) in order to meet both prerequisites (i) and (ii).

Now that we have chosen a spanning degree-3 spider T, we draw T edge by edge proceeding outwards from the root r. In section 4.3, we presented the algorithm Draw-Degree-3-Spider from Fig. 4.6 that maintains the following two invariants for the T' drawn so far:

(1) Two of the leaves s and t of T' 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.

Recall that if T' does not intersect ℓ_u to the left or to the right of u, then we can always place the next neighbor v of u in $T - T'$ one unit to the left or to the right of T', respectively. We repeat this process until v becomes ϕ-extreme. Either the previous ϕ-extreme leaf s or t then becomes u. Since that leaf was previously ϕ-extreme by invariant (1), T' now only intersects the track of the leaf either to its left or to its right, maintaining invariant (2).

Also recall that whenever we draw an edge from u to v, we use the algorithm Draw-Bent-Edge from Fig. 4.13 that bends the edge (u, v) at $\big(x_v,\, \phi(u) - 1\big)$ if $\phi(u) > \phi(v)$, and at $\big(x_v,\, \phi(u) + 1\big)$. This allows the bent edge to be routed above or below other edges of T' as needed; see Fig. 4.5(d).

To obtain an initial subtree T' that obeys both invariants, we pass T to the algorithm Start-Drawing-Extended-3-Spider given in Fig. 7.13. This is in lieu of using the algorithm Start-Drawing-Degree-3-Spider from section 4.3 so that we can accommodate the drawing the extra possible edge e_1 of G as a straight edge. We denote the neighbors of r by v_{min}, v_{mid}, and v_{max} where $v_{min} < v_{mid} < v_{max}$ and also by x, y, and z where (x, y) is the edge e_1. If not all three chains start upward

Algorithm: START-DRAWING-EXTENDED-3-SPIDER

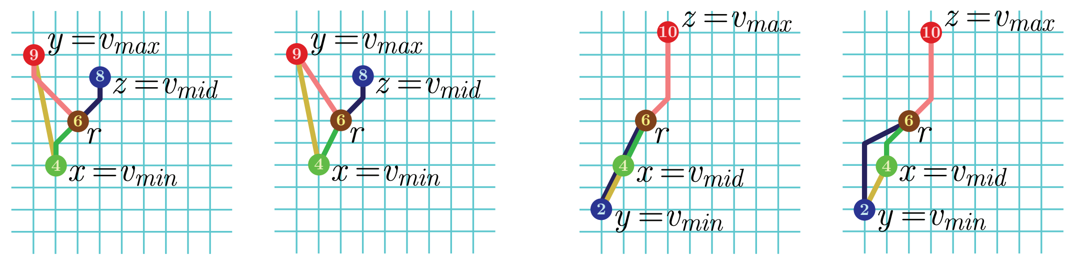
▷ `Initially draw extended 3-spider until both invariants hold allowing for an edge` (x, y) `between neighbors of root` r

Input: Degree-3 spider $T(V, E, \phi)$ with distinct labels ϕ, root r, edge e connecting neighbors of r (optional)
Output: Degree-3 spider $T'(V', E')$ graph drawn so far

Place r at $\big(0,\, \phi(r)\big)$, $\{x, y, z\} \leftarrow N(r)$ **where** (x, y) is edge e if given
$\{v_{min}, v_{mid}, v_{max}\} \leftarrow N(r)$ **where** $\phi(v_{min}) < \phi(v_{mid}) < \phi(v_{max})$
$V' \leftarrow \{r\}$, $E' \leftarrow \varnothing$ ▷ $T'(V', E')$ `is the tree drawn so far`
if $\phi(v_{min}) < \phi(r) < \phi(v_{max})$ **then** ▷ `if not all three start up or down`
  if $\phi(x) < \phi(r) < \phi(y)$ **or** $\phi(x) > \phi(r) > \phi(y)$ **then**
    Place x, y, and z at $\big(-1,\, \phi(x)\big)$, $\big(-2,\, \phi(x)\big)$, $\big(1,\, \phi(z)\big)$, respectively
    Draw straight edges r–x, r–y, and r–z
    Update $V' \leftarrow \{r, x, y, z\}$ and $E' \leftarrow \{(r, x), (r, y), (r, z)\}$
  else
    DRAW-BENT-EDGE$\big(T,\, T',\, r,\, x,$ LEFT$\big)$
    DRAW-BENT-EDGE$\big(T,\, T',\, r,\, y,$ LEFT$\big)$
    DRAW-BENT-EDGE$\big(T,\, T',\, r,\, z,$ RIGHT$\big)$
else
  $w_{extr} \leftarrow r$ ▷ `current most` ϕ-extreme `vertex`
  foreach $v \in N(r)$ **do**
    $w'_{extr} \leftarrow$ GET-EXTREME$(T,\, r,\, v)$
    if $\phi(r) \le \phi(w_{extr}) < \phi(w'_{extr})$ **or** $\phi(r) \ge \phi(w_{extr}) > \phi(w'_{extr})$ **then**
      $w_{extr} \leftarrow w'_{extr}$, $v_{extr} \leftarrow v$
  $\{v_{left}, v_{right}\} \leftarrow N(r) \setminus \{v_{extr}\}$
  if $\{x, y\} = \{v_{left}, v_{right}\}$ **then** ▷ `draw` v_{right} `before` v_{extr}
    DRAW-BENT-EDGE$(T,\, T',\, r,\, v_{right},$ RIGHT$)$
  DRAW-BENT-EDGE$(T,\, T',\, r,\, v_{extr},$ RIGHT$)$ ▷ `draw` v_{extr}
  EXPAND-CHAIN$(T,\, T',\, v_{extr},$ RIGHT$,\, w_{extr})$
  if $\{x, y\} \neq \{v_{left}, v_{right}\}$ **then** ▷ `draw` v_{right} `after` v_{extr}
    DRAW-BENT-EDGE$(T,\, T',\, r,\, v_{right},$ RIGHT$)$
  DRAW-BENT-EDGE$(T,\, T',\, r,\, v_{left},$ LEFT$)$ ▷ `draw` v_{left}
  EXPAND-CHAIN$(T,\, T',\, v_{left},$ RIGHT, **null**$)$
return $T'(V', E')$

Figure 7.13: START-DRAWING-EXTENDED-3-SPIDER

or downward, i.e., $\phi(v_{min}) < \phi(r) < \phi(v_{max})$, then there are two cases to consider as illustrated in Fig. 7.14.

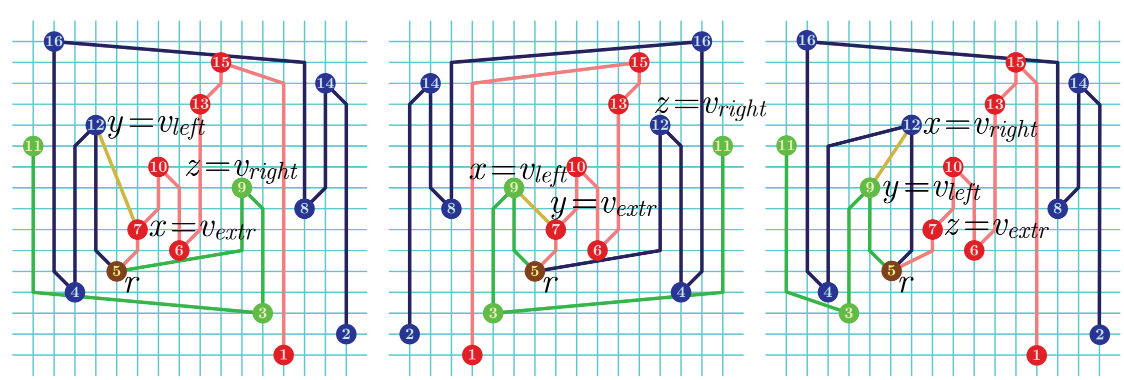


(a) (b) (c) (d)

Figure 7.14: Two initial cases for an extended 3-spider when $\phi(v_{min}) < \phi(r) < \phi(v_{max})$

If x and y are on opposite sides of ℓ_r (i.e., $\phi(x) < \phi(r) < \phi(y)$ or $\phi(x) > \phi(r) > \phi(y)$), then drawing edges (r, x) and (r, y) with bent edges can lead to a crossing when drawing (x, y) with a straight edge as seen in Fig. 7.14(a). Using straight edges to draw the edges incident to r avoids this problem provided that x and y are both drawn to the left of r as in Fig. 7.14(b). However, if x and y are on the same side of ℓ_r, then drawing edges (r, x) and (r, y) with straight edges can lead to overlapping edges as seen in Fig. 7.14(c). Here we revert to the standard procedure of using Draw-Bent-Edge to draw the edges incident to r as in Fig. 7.14(d).

However, if all three chains start upward or downward, then we must be more careful in order to accommodate edge e_1 that has the three possibilities shown in Fig. 7.15. We now denote the three neighbors of r by v_{extr}, v_{right}, and v_{left}. The vertex v_{extr} is on the chain that has the most extreme vertex w_{extr} before it cross



(a) **(b)** **(c)**

Figure 7.15: Examples of three extended 3-spiders with the edge (x, y)

the track ℓ_r of r; see Fig. 4.14. The algorithm GET-EXTREME from Fig. 4.11 determines w_{extr}, and the algorithm EXPAND-CHAIN from Fig. 4.12 can expand the chain from v_{extr} to w_{extr}. The vertices v_{right} and v_{left} are the other neighbors of r where v_{left} will always be the leftmost vertex of the three. Previously, in the algorithm START-DRAWING-DEGREE-3-SPIDER from Fig. 4.8, v_{right} was always the rightmost vertex, which is fine so long as (v_{right}, v_{left}) is the not the edge e_1 as in Fig. 7.15(a)–(b). However, if (v_{right}, v_{left}) is the edge (x, y) as in Fig. 7.15(c), then we need to draw the chain with v_{right} *before* we draw the chain with v_{extr}.

After extracting the appropriate spanning degree-3 spider T and drawing T initially to allow for the edge e_1, we now show how to draw the remainder of T to allow for the edge e_2. Let C denote the cycle in $T + e_2$ (provided e_2 exists) and $P = T - C$. Recall that the algorithm DRAW-DEGREE-3-SPIDER in section 4.3 kept expanding the chain with leaf u while $\phi(s) < \phi(u) < \phi(t)$. Once this was no longer true, i.e., $\phi(u) < \phi(s)$ or $\phi(u) > \phi(t)$, then the algorithm updated s or t to u, respectively, where the old s or t became the new u.

The algorithm DRAW-DEGREE-3-SPIDER does not keep track of which side of a track that T' intersects. Rather it simply alternates directions each time it switches the chain being extended. For instance, if T' intersects the track ℓ_u of u to the left, the algorithm continues extending the chain to the right until it intersects the track ℓ_s of s below or the track ℓ_t of t above. The algorithm can safely extend the next chain to the left, and so on.

However, by including edge e_2, the algorithm now needs to stop once s, t, or u becomes one of the endpoints of e_2 so that it can drawn the remainder of the cycle C in $T + e_2$. Figure 7.16 show three scenarios where the first two result in crossings by blindly following the algorithm as before. Another difference is that the final edge may require its bend to be shifted one to the left or to the right as in Fig. 7.17

The algorithm DRAW-EXTENDED-3-SPIDER given in Fig. 7.18 is the modified version of the algorithm DRAW-DEGREE-3-SPIDER that considers three cases for when one of the vertices s, t, or u become an endpoint of e_2. In each case, vertices v and w are assigned to be the leaves of T' of the remaining two chains of $T - T'$,

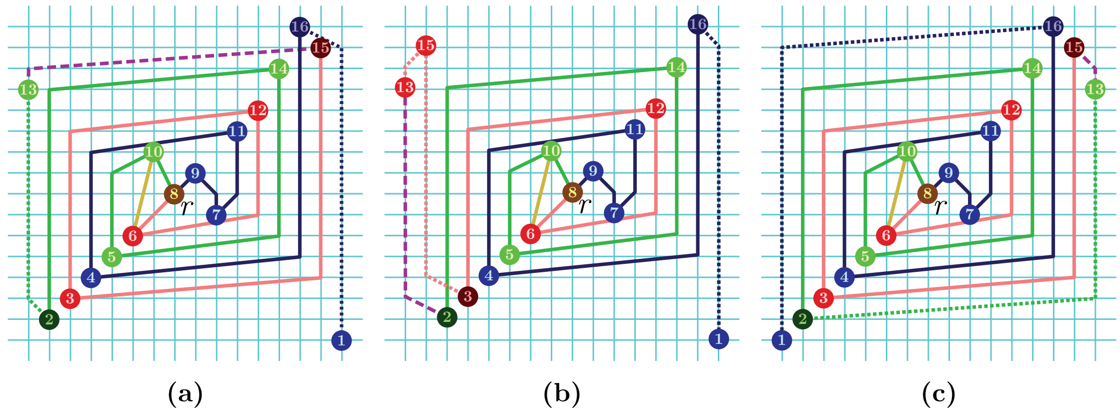
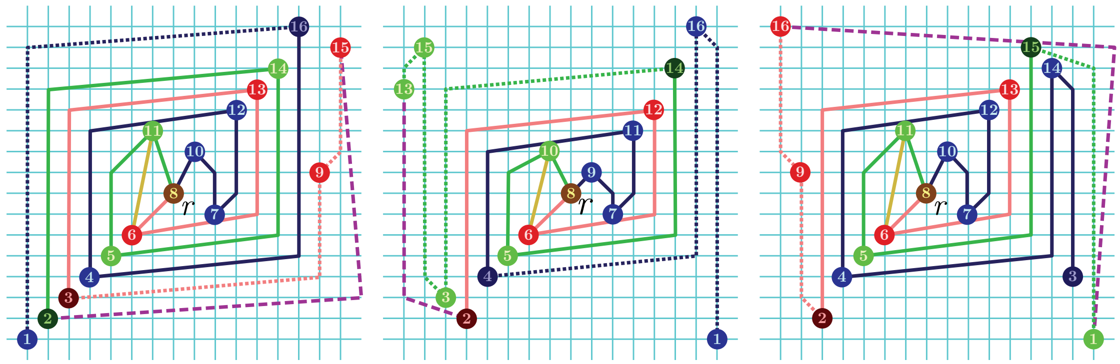


Figure 7.16: Three scenarios for when the algorithm Draw-Degree-3-Spider reaches an endpoint of the dashed edge e_2 that is not in spanning degree-3 spider T. The dotted edges are drawn after the edge e_2 is reached. In both (a) and (b), the other two chains are expanded in the same directions as before, which leads to a crossing. To avoid this problem, the directions must be sometimes be switched as in (c).

where the chain of v is expanded first in previous direction of u (with the exception of the case noted in Fig. 7.16(c)), and then the chain with w is expanded next in the opposite direction. The three cases are as follows:

(1) If u is an endpoint of e_2, then the next chain to expand is the one containing the second endpoint of e_2. Expanding this chain in the same direction that the chain of u was proceeding will allow e_2 to be drawn without a crossing, since u is guaranteed to be a ϕ-extreme vertex of C by Get-Spanning-Degree-3-Spider. The remainder of the third chain P can be freely expanded in the opposite direction, which is always possible, since it was ϕ-extreme when the edge e_2 was reached. Figure 7.17(a) gives an example of this where $s = 2$, $t = 16$, and $u = 3$.

(2) Else if the chain of u still remains to be expanded (which must be the chain P since u is not an endpoint of e_2), then this need to be done next, proceeding in the same direction as before until P is exhausted. The remaining chain is expanded in the opposite direction until it reaches the other endpoint of e_2. The edge e_2 can always be drawn without a crossing, since u is guaranteed to be a ϕ-extreme vertex of the cycle C given by Get-Spanning-Degree-3-Spider. Figure 7.17(b) gives an example of this where $s = 2$, $t = 14$, and $u = 4$.



(a) **(b)** **(c)**

Figure 7.17: The three cases for when the algorithm DRAW-EXTENDED-3-SPIDER reaches the dashed edge e_2, where the dotted edges are drawn after e_2 is reached. The x-coordinate of the bend of the final edge e_2 is either the maximum or the minimum x-coordinate of T as in (b), or becomes x-extreme of T as in (a) and (c) in order to avoid an edge overlap.

(3) The only remaining possibility is that s and t are endpoints of the edge e_2 where u is the endpoint of the chain P that has been completely drawn. We need to be careful to expand each of the two remaining chains in opposite directions so as to avoid a crossing between the two chains. Figure 7.17(c) gives an example of this where $s = 2$, $t = 15$, and $u = 3$.

Finally, we draw the edge e_2 with the algorithm DRAW-CYCLE-EDGE given in Fig. 7.19. This algorithm attempts to draw the edge with a bend like DRAW-BENT-EDGE does, but it can result in overlapping edges if the edge is going in the same direction as a previously drawn incident edge; see Fig. 7.17. The algorithm takes advantage of the following two conditions regarding e_2:

(a) One of the endpoints of e_2 is ϕ-extreme of the chain C.

(b) One of the endpoints of e_2 is placed last in T', and hence, is x-extreme in T.

It is possible for each of the endpoints to meet both conditions (a) and (b) if DRAW-EXTENDED-3-SPIDER reached edge e_2 under case (3) above where both endpoints of e_2 are ϕ-extreme as in Fig. 7.17(c). However, it is not possible for one endpoint s to meet neither condition and for the other endpoint t to meet both

Algorithm: DRAW-EXTENDED-3-SPIDER

▷ `Draw an extended 3-spider with distinct labels in` $O(n)$ `time`

Input: Extended 3-spider $G(V, E, \phi)$ with distinct labels ϕ
Output: Level planar drawing of G

```
r ← root of G                                  ▷ any degree-3 vertex in G
T ← GET-SPANNING-DEGREE-3-SPIDER(G, r)
{e1, e2} ← E(G − T) where V(e1) ⊂ N(r) and V(e2) are leaves in T
C ← r–···–s–t–···–r where e2 = (s, t)
T′ ← START-DRAWING-EXTENDED-3-SPIDER(T, r, e1)
if e1 ≠ ∅ then draw edge x–y              ▷ draw e1 straight if it exists
U ← {s, t, u} be the leaves of T′ where φ(s) < φ(u) < φ(t)
direction ← RIGHT
while |N(u)| ≠ 1 and {s, t, u} ∩ V(e2) = ∅ do
  | v ← EXPAND-CHAIN(T, T′, u, direction)
  | if φ(v) < φ(s) or φ(v) > φ(t) then
  |   | if φ(v) < φ(s) then update u ← s and s ← v
  |   | if φ(v) > φ(t) then update u ← t and t ← v
  |   | Change direction                  ▷ RIGHT to LEFT, and vice versa
  | else update u ← v
if u ∈ V(e2) then                               ▷ if u endpoint of e2
  | if s ∈ V(C) then v ← s, w ← t  else v ← t, w ← s
else if |N(u)| ≠ 1 then                ▷ else if u is not a leaf in T
  | v ← u                       ▷ need to expand remainder of chain of u
  | if |N(s)| ≠ 1 then w ← s else w ← t
else if s ∈ V(C) and t ∈ V(C) then             ▷ else s and t both in C
  | v ← s, w ← t
  | if x_v < x_w then  direction ← LEFT else  direction ← RIGHT
while |N(v)| ≠ 1 do                         ▷ while v is not a leaf in T
  | v ← EXPAND-CHAIN(T, T′, v, direction)
Change direction                          ▷ RIGHT to LEFT, and vice versa
while |N(w)| ≠ 1 do                         ▷ while w is not a leaf in T
  | w ← EXPAND-CHAIN(T, T′, w, direction)
DRAW-CYCLE-EDGE(T, T′, r, e2)
```

Figure 7.18: DRAW-EXTENDED-3-SPIDER

conditions. That would imply that the subpath $r \rightsquigarrow t$ of $C - e_2$ would contain both ϕ-extreme vertices of C, violating the prerequisite (iii) of how edge e_2 is chosen by GET-SPANNING-DEGREE-3-SPIDER.

Algorithm: DRAW-CYCLE-EDGE

▷ `Draw cycle edge of extended 3-spider` G `as a bent edge. One endpoint of the edge is leftmost or rightmost of all vertices in` G. `May shift the edge bend over to avoid an edge overlap.`

Input: Extended 3-spider $G(V, E, \phi)$ with distinct labels ϕ, edge e
Output: Updated level planar drawing of T

```
x_min ← min{x_v : v ∈ V}                          ▷ minimum x-coordinate of G
x_max ← max{x_v : v ∈ V}                          ▷ maximum x-coordinate of G
{u, v} ← V(e) where v = x_min or v = x_max          ▷ v is x-extreme in G
w ← N(v) \ {u}                                     ▷ edge (v, w) already drawn
if φ(u) < φ(v) then                                ▷ if u below v
    y_b ← φ(u) + 1                                 ▷ place bend one above u
else                                               ▷ else u above v
    y_b ← φ(u) − 1                                 ▷ place bend one below u
x_b ← x_v                                  ▷ start with bend above or below v
if (φ(u) < φ(v) and φ(w) < φ(v)) or (φ(u) > φ(v) and φ(w) > φ(v)) then
    if x_u < x_v then                              ▷ if u left of v
        x_b ← x_b + 1                              ▷ move bend right of v
    else                                           ▷ else u right of v
        x_b ← x_b − 1                              ▷ move bend left of v
Place bend b at (x_b, y_b) and draw edges u–b and b–v
```

Figure 7.19: DRAW-CYCLE-EDGE

The algorithm DRAW-CYCLE-EDGE given in Fig. 7.19 denotes the endpoint of e_2 that meets condition (b) by v, and denotes the other endpoint by u, which must meet condition (a). Since u is ϕ-extreme of C, the bend b can be placed one unit below or above u, depending on whether v is above or below u, respectively. The bend b can be placed directly above or below v only if the edge (v, w) (where w is the other neighbor v) is incident to v from below or above, respectively. Otherwise, the bend needs to be shifted one unit to the left or to the right (becoming a new x-extreme of G) if v is leftmost or rightmost, respectively. This avoids the bent edge u–b–v from overlapping with the previous drawn edge (v, w) incident to v. This only happens once. Thus, the final drawing of G takes $(n + 1) \times n$ space instead of the $n \times n$ space taken by the drawing produced by DRAW-DEGREE-3-SPIDER. □

We can apply the same argument given in Lemma 4.3.2 with respect to extended 3-spiders to obtain the following corollary:

Corollary 7.2.2. *An n-vertex extended 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.*

7.3 Drawing Extended-K_4 Subgraphs with Distinct Labels

From Lemma 3.4.5, we know that if an EK4 is not biconnected then it is either a GC or an E3S, which the previous algorithms can draw. Hence, we only need to realize a biconnected EK4 as in Fig. 7.20, which we can do with at most two edge bends.

Lemma 7.3.1. *An n-vertex biconnected extended K_4 subgraph G can be realized in $O(n)$ time on an $(n-1) \times n$ grid for any labeling.*

Proof. We take the approach to add a set of edges E_{extra} to $E(G)$ so that $G' = G + E_{extra}$ is an extended K_4, and then show how to draw G'. Before drawing any edge e, we first verify that $e \notin E_{extra}$ while generating our realization of G. To determine the extra edges E_{extra}, we first use the algorithm Get-Diamond-Block from Fig. 7.21, which selects a subset of vertices $\{u, v, s, t\}$ of $V(G)$ such that the induced graph on the four vertices $G'[\{u, v, s, t\}]$ is an extended K_4, where G' is an

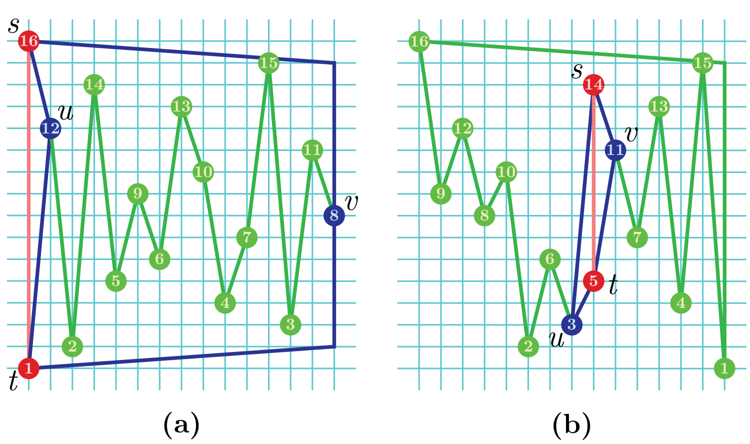


Figure 7.20: Realizations of two 16-vertex biconnected EK4s on 15×16 grids illustrating the two cases in which s and t are or are not ϕ-extreme. If they are, the edge connecting s and t requires two bends as in (a). If not, only one edge bend is required as in (b).

Algorithm: GET-DIAMOND-BLOCK

▷ `Get the diamond block of an extended` K_4 `supergraph`

Input: Biconnected extended K_4 subgraph $G(V,\ E,\ \phi)$ with distinct labels ϕ
Output: Block $B(V', E')$ where $V' = \{u, v, s, t\} \subseteq V$ is a diamond block joining u to v in the extended K_4 supergraph $G(V, E \cup E')$

$V_{deg=3} \leftarrow \{v : v \in V$ and $deg(v) = 3\}$ ▷ `degree-3 vertices of` G
if $|V_{deg=3}| = 0$ **then** ▷ `if` G `is a cycle`
  $\{u, v, s, t\} \subseteq V$ **where** $\{(v, s), (s, t), (t, u)\} \subseteq E$
else if $|V_{deg=3}| = 2$ **then** ▷ `else if` G `has a` K_3 `or a` C_4 `block`
  $\{u, v\} \leftarrow V_{deg=3}$
  if $u \notin N(v)$ **then** ▷ `if` G `has a` C_4 `block`
    $\{s, t\} \leftarrow N(u) \cap N(v)$ ▷ `where` u—s—v—t—u `is a` C_4 `block`
  else ▷ `else` G `has a` K_3 `block`
    $\{u, s\} \leftarrow V_{deg=3}$ ▷ `where` s—t—u—s `is a` K_3 `block`
    $\{t\} \leftarrow N(u) \cap N(s)$, $\{v\} \leftarrow N(s) \setminus \{u\}$
else if $|V_{deg=3}| = 4$ **then** ▷ `else if` G `has a diamond block`
  $\{u, v, s, t\} \leftarrow V_{deg=3}$ **where** $u \notin N(v)$ **and** $s \in N(t)$
if $\phi(u) < \phi(v)$ **then** $u \leftrightarrow v$ ▷ `guarantee that` $\phi(u) > \phi(v)$
if $\phi(s) < \phi(t)$ **then** $s \leftrightarrow t$ ▷ `guarantee that` $\phi(s) > \phi(t)$
$V' \leftarrow \{u, v, s, t\}$, $E' \leftarrow \{(s, t), (u, s), (u, t), (v, s), (v, t)\}$
return $B(V', E')$

Figure 7.21: GET-DIAMOND-BLOCK

ULP diamond block B joining u to v. By Lemma 3.4.6, we know that G consists of a cycle in which at most one edge has been replaced with an ULP K_3, C_4, or diamond block. We consider each of the four possibilities:

1. If G is a cycle, then we pick u, v, s, t such that v–s–t–u is a subpath of G.

2. If G has a K_3 block, then we let u and s be the two vertices of degree 3 where $N(u) \cap N(s) = \{t\}$ so that s–t–u–s forms the 3-cycle.

3. If G has a C_4 block, then we let u and v be the two vertices of degree 3 where $N(u) \cap N(v) = \{s, t\}$ so that u–s–v–t–u forms the 4-cycle.

4. If G has a diamond block, then we let u and v be the two vertices of degree 3 that not adjacent, and let s and t be the other two vertices of degree 3.

Algorithm: DRAW-BICONNECTED-EXTENDED-K_4-SUBGRAPH

▷ `Draw a biconnected extended-`K_4` subgraph with distinct labels in `$O(n)$` time.`

Input: Biconnected extended K_4 subgraph $G(V, E, \phi)$ with distinct labels ϕ where $|V| \geq 4$
Output: Level planar drawing of G

$B(V', E') \leftarrow$ GET-DIAMOND-BLOCK(G)
$\{u, v, s, t\} \leftarrow V'$ **where** $u \notin N_B(v)$ **and** $\phi(u) > \phi(v)$ **and** $\phi(s) > \phi(t)$
$E_{extra} \leftarrow E' \setminus E$, $E \leftarrow E \cup E'$
$\phi_{min} \leftarrow \min\{\phi(v) : v \in V\}$, $\phi_{max} \leftarrow \max\{\phi(v) : v \in V\}$
if $\phi(s) = \phi_{max}$ **and** $\phi(t) = \phi_{min}$ **then**
    v_1–v_2–$\cdots$–v_{n-1}–$v_1 \leftarrow$ cycle $G - t$ **where** $v_1 = s$ **and** $v_2 = u$ **and** $v_n = v$
else
    if $\phi(s) = \phi_{max}$ **or** $\phi(t) = \phi_{max}$ **then**
        Let $v_1 \in V \setminus \{t\}$ **where** $\phi(v_1) = \phi_{min}$
    else Let $v_1 \in V \setminus \{t\}$ **where** $\phi(v_1) = \phi_{max}$
    v_1–v_2–$\cdots$–v_{n-1}–$v_1 \leftarrow$ cycle $G - t$ **where** $u = v_i$ **and** $v = v_j$ **and** $i < j$
for $i = 1$ *to* $n - 1$ **do**
    Place v_i at $\big(i, \phi(v_i)\big)$
    if $i < n - 1$ **and** $(v_i, v_{i+1}) \notin E_{extra}$ **then**
        Draw edge v_i–v_{i+1}
    else if $i = n - 1$ **and** $(v_1, v_{n-1}) \notin E_{extra}$ **then**
        if $\phi(v_1) = \phi_{max}$ **then** Place bend b at $\big(n - 1, \phi_{max} - 1\big)$
        else Place bend b at $\big(n - 1, \phi_{min} + 1\big)$ ▷ `else `$\phi(v) = \phi_{min}$
        Draw edges v_1–b and b–v_{n-1}
$t_x \leftarrow s_x$ and place t at $\big(t_x, \phi(t)\big)$ ▷ `place `t` above or below `s
if $(s, t) \notin E_{extra}$ **then** draw edge s–t
if $(u, t) \notin E_{extra}$ **then** draw edge u–t
if $(v, t) \notin E_{extra}$ **then**
    if $t_x = 1$ **and** $v_x = n - 1$ **then**   ▷ `where `$\phi(t) = \phi_{min}$
        Place bend b at $\big(n - 1, \phi_{min} + 1\big)$ and draw edges v–b and b–t
    else Draw edge v–t

Figure 7.22: DRAW-BICONNECTED-EXTENDED-K_4-SUBGRAPH

We observe that in each of the four cases that the path $u \rightsquigarrow v$ is not part of the block B that connects u to v in G. Finally, we swap s and t as well as u and v if needed to ensure that $\phi(s) > \phi(t)$ and $\phi(u) > \phi(v)$. We proceed to draw G' using the algorithm DRAW-BICONNECTED-EXTENDED-K_4-SUBGRAPH given in Fig. 7.22.

First, we show how to draw the cycle v_1–v_2–$\cdots$–v_{n-1}–v_1 of $G - t$. There are two cases: (i) either s and t are both ϕ-extreme as in Fig. 7.20(a) or (ii) only one or neither of s and t are ϕ-extreme as in Fig. 7.20(b). In the first case, we set $v_1 = s$, $v_2 = u$, and $v_{n-1} = v$ so that when we draw the path v_1–v_2–$\cdots$–v_{n-1} left to right, s is leftmost and v is rightmost, and the edge (v, s) can be drawn with a bend placed directly above v at $\big(x_v,\, \phi(s) - 1\big)$. We place t directly under s at $\big(x_s,\, \phi(t)\big)$, so that we can draw edge (v, t) with a bend placed directly below v at $\big(x_v,\, \phi(t)+1\big)$. Finally, we draw the edges (s, t) and (u, t) as straight edges; see Fig. 7.20(a). In the second case, we set v_1 to be a ϕ-extreme vertex other than s or t. Suppose, w.l.o.g. that v_1 is ϕ-maximal. We draw the path v_1–v_2–$\cdots$–v_{n-1} from left to right, and draw the edge v_1–v_{n-1} with a bend placed directly above v_{n-1} at $\big(n-1,\, \phi(v_1)-1\big)$. We place t directly under s at $\big(x_s,\, \phi(t)\big)$, so that we can draw the incident edges (s, t), (u, t), and (v, t) each with straight edges. $\square$

We observe that Draw-Biconnected-Extended-K_4-Subgraph uses at most two edge bends placed directly above or below v. In the worst case, the top and bottom parts of the bent edges have slopes of $\mp 1/(n-1)$, respectively. By moving the rightmost vertex v from $\big(n-1,\, \phi(v)\big)$ to $\big((n-1)^2,\, \phi(v)\big)$, both edges can be drawn straight without crossing any other edge, which gives the following corollary:

Corollary 7.3.2. *An n-vertex extended K_4 subgraph can be realized with no bends in $O(n)$ time though it may require up to $O(n^2) \times n$ area for some distinct labelings.*

Combining Lemmas 4.2.1, 7.2.1, 7.1.1, and 7.3.1, which provide the drawing algorithms for radius-2 stars, generalized caterpillars, extended 3-spiders, and biconnected extended K_4 subgraphs, respectively, and the fact that all other extended K_4 subgraphs are either generalized caterpillars or extended 3-spiders by Lemma 3.4.5, we have our next theorem.

Theorem 7.3.3. *Generalized caterpillars, radius-2 stars, extended 3-spiders, and extended K_4 subgraphs are all ULP with distinct labels and can be realized in $O(n)$ time.*

7.4 Drawing K_3-Caterpillars with Duplicate Labels

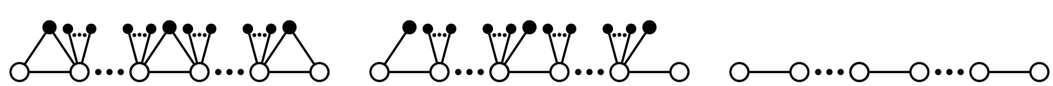
In this section, we show how to draw K_3-caterpillars for any duplicate labeling. We extend the caterpillar drawing algorithm from section 4.4 to accommodate the extra edges of the K_3 blocks of a K_3-caterpillar. To do so, we need to extract a very particular spanning tree given by the following definition:

Definition 7.4.1. *A* left-to-right caterpillar *of a K_3-caterpillar G is a spanning caterpillar of G with a* left-to-right path $v_r \rightsquigarrow v_l$ *constructed by replacing each K_3 block in G with an edge to obtain a caterpillar T where:*

1. *If G is biconnected, then $n(G) \leq 3$. If $V = \{v\}$, then $v_r = v_l = v$. If $V = \{u, v\}$, then $v_r = u$ and $v_l = v$. Finally, if G is isomorphic to the K_3 block u–v–w–u, then add the edge (u, w) to T where $v_r = u$ and $v_l = v$.*

2. *Otherwise, G has k cut-vertices $v_1, \ldots, v_k$, where v_1–$\cdots$–v_k forms the initial spine of T and $v_r = v_1$ and $v_l = v_k$ initially. For each K_3 block B_i joining v_i to v_{i+1} for $i \in \{1, \ldots, k-1\}$, i.e., $B_i = v_i$–v_{i+1}–w–v_i, add the edge (v_i, w) to T. If G has the K_3 block B_0 ending on v_1, i.e., $B_0 = v_1$–u–w–v_1, then add the edge (u, w) to T where $v_r = u$. If G has the K_3 block B_k ending on v_k, i.e., $B_k = v_k$–u–w–v_k, then add the edge (v_k, w) to T where $v_l = w$.*

Figure 7.23 illustrates the relationship between a K_3-caterpillar and its left-to-right caterpillar and its left-to-right path. Figure 7.24 gives an example K_3-caterpillar where the red edges are the not in the left-to-right caterpillar and v_1–v_2–$\cdots$–v_6 is the left-to-right path.



K_3-Caterpillar (a) Left-To-Right Caterpillar (b) Left-to-Right Path (c)

Figure 7.23: A K_3-caterpillar in (a) with corresponding left-to-right caterpillar and left-to-right path in (b) and (c), respectively.

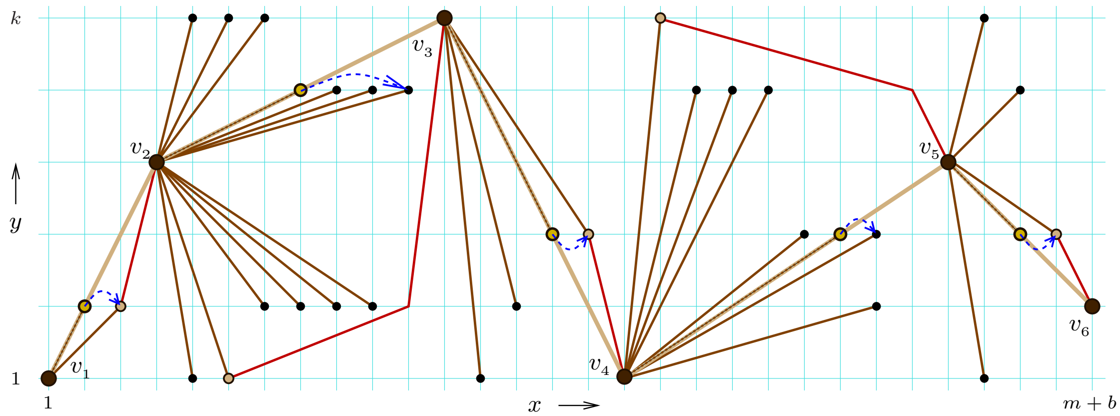


Figure 7.24: A realization of a 33-vertex K_3-caterpillar on a 30×6 grid. Arrows indicate how vertices initially placed on spine edges are moved in order to avoid edge overlaps.

Figure 7.25 gives the algorithm GET-LEFT-TO-RIGHT-CATERPILLAR, which extracts the left-to-right caterpillar and left-to-right path from a given K_3-caterpillar in linear time. Let us define $L_{above}(v)$ and $L_{below}(v)$ to be the sets of pendant vertices that are adjacent to vertex v in a left-to-right caterpillar T of a K_3-caterpillar G with labels greater than and less than $\phi(v)$, respectively. Using these definitions, we next determine the space required to realize a K_3-caterpillar for any labeling.

Lemma 7.4.2. *An n-vertex K_3-caterpillar G on k levels with a m-edge left-to-right path can be realized with at most one bend per edge in $O(n)$ time on a $(2m+b) \times k$ grid for any labeling where $b = \sum_{i=1}^{m} \max\left\{\left|L_{above}(v_i)\right|, \left|L_{below}(v_i)\right|\right\}$.*

Proof. We extend the high-level algorithm DRAW-CATERPILLAR from Fig. 4.19 in section 4.4 to the corresponding algorithm DRAW-K_3-CATERPILLAR given in Fig. 7.26 to allow for the extra edges of the K_3 blocks of a K_3-caterpillar as shown in Fig. 7.23(a).

We extract the left-to-right caterpillar T as in Fig. 7.23(b) and left-to-right path P as in Fig. 7.23(c) in linear time using GET-LEFT-TO-RIGHT-CATERPILLAR in Fig. 7.25. If G is biconnected, then either G is an isolated vertex (as is P) or G is a single P_2 or K_3 block where P is single edge. On the other hand if G is not biconnected and has k cut-vertices, then T is constructed from a caterpillar T' formed by replacing each K_3 block in G with an edge. For each K_3 2-block B_i in G

Algorithm: GET-LEFT-TO-RIGHT-CATERPILLAR

▷ `Extract right-to-left spanning caterpillar with right-to-left path from an` n`-vertex` K_3`-caterpillar in` $O(n)$ `time`

Input: K_3-caterpillar $G(V, E)$
Output: Left-to-right caterpillar $T(V, E')$, left-to-right path $v_r \rightsquigarrow v_l$

if *G is biconnected* **then** ▷ `if` G `is a lone vertex, edge or` K_3 `block`
  $T \leftarrow G$, **if** $|V| = 1$ **then** $\{v_r\} \leftarrow \{v_l\} \leftarrow V$ **else** $\{v_r, v_l\} \subseteq V$
else ▷ `else` G `is not biconnected`
  $\mathcal{B} \leftarrow K_3$ blocks of G ▷ `either 1-blocks or 2-blocks`
  $T \leftarrow$ caterpillar created by replacing each block in $\mathcal{B}$ with an edge
  $v_1 - \cdots - v_k \leftarrow$ REMOVE-LEAVES(T) ▷ `cut-vertices in` G
  $v_r \leftarrow v_1, v_l \leftarrow v_k$ ▷ `right and left vertices of path`
  if $k = 1$ **then** ▷ `if` T `is a star`
    $\{B_0, B_k\} \leftarrow K_3$ 1-blocks of $\mathcal{B}$ ▷ `either may be empty`
  else ▷ `else` T `is not a star`
    $B_0 \leftarrow K_3$ 1-block of $\mathcal{B}$ with connector v_1 ▷ `maybe be empty`
    $B_1, \ldots, B_{k-1} \leftarrow$ 2-blocks of G **where** B_i has connectors v_i and v_{i+1}
    $B_k \leftarrow K_3$ 1-block of $\mathcal{B}$ with connector v_k ▷ `maybe be empty`
  if *B_0 is non-empty* **then**
    $\{u, w\} \leftarrow V(B) \setminus \{v_1\}$ ▷ `other vertices of` B
    $v_r \leftarrow u$ ▷ `new right vertex of path`
    $E(T) \leftarrow E(T) \cup \{(u, w)\}$ ▷ `add edge` (v_r, w) `to` T
  for $i \leftarrow 1$ *to* $k - 1$ **do**
    if IS-K_3-BLOCK(B_i, v_i, v_{i+1}) **then**
      $w \leftarrow V(B) \setminus \{v_i, v_{i+1}\}$ ▷ `third vertex of` B
      $E(T) \leftarrow E(T) \cup \{(v_i, w)\}$ ▷ `add edge` (v_i, w) `to` T
  if *B_k is non-empty* **then**
    $\{u, w\} \leftarrow V(B) \setminus \{v_k\}$ ▷ `other vertices of` B
    $v_l \leftarrow w$ ▷ `new left vertex of path`
    $E(T) \leftarrow E(T) \cup \{(v_k, w)\}$ ▷ `add edge` (v_k, v_l) `to` T
return *T, path $v_r \rightsquigarrow v_l$ in T*

Figure 7.25: GET-LEFT-TO-RIGHT-CATERPILLAR

that corresponds to the edge $(v_i, v_i + 1)$ of the spine $v_1 - \cdots - v_k$ of T', the leaf edge (v_i, w) is added to T' for $i \in \{1, \ldots, k-1\}$ where w is degree-2 vertex of B_i.

If G has no ending blocks, then P is merely the spine of T' with endpoints v_1 and v_k. Otherwise, G has at most two K_3 blocks B_0 and B_k that end on the

Algorithm: DRAW-K_3-CATERPILLAR

$\triangleright$ `Draw an` n`-vertex` K_3`-caterpillar with duplicate labels in` $O(n)$ `time`

Input: K_3-caterpillar $G(V, E, \phi)$ with duplicate labels ϕ where $|V| > 2$
Output: Level planar drawing of G

$T, P \leftarrow$ GET-LEFT-TO-RIGHT-CATERPILLAR(G) $\triangleright$ P `is right-to-left path` $v_1 — v_2 — \cdots — v_{m+1}$
$E_{K_3} \leftarrow E(G) - E(T)$ $\triangleright$ `extra edges of` K_3 `blocks`
$V_{K_3} \leftarrow V(G - T)$ $\triangleright$ `endpoints of each edge in` E_{K_3}
$L_1, \ldots, L_m \leftarrow$ SORT-CATERPILLAR-LEAVES(T, P) $\triangleright$ `where` $L_i \subseteq N(v_i)$
$x_1 \leftarrow 1$, place v_1 at $\big(x_1, \phi(v_1)\big)$ $\triangleright$ x_i `is` x`-coordinate of` v_i
for $i \leftarrow 1$ *to* $m + 1$ **do**
  $x_{max} \leftarrow$ PLACE-CATERPILLAR-LEAVES(T, v_i, L_i)
  if $i \leq m$ **then**
    $x_{i+1} \leftarrow x_{max} + 2$ $\triangleright$ `extra space needed for` K_3 `edge`
    Place v_{i+1} at $\big(x_{i+1}, \phi(v_{i+1})\big)$ and draw edge v_i–v_{i+1}
  foreach *leaf* $\ell \in N_T(v_i)$ **do** $\triangleright$ `for each leaf of` v_i `in` T
    if ℓ *lies on edge* v_i–v_{i+1} **or** $\ell \in V_{K_3}$ **then**
      $V_{\phi(\ell)} \leftarrow \{v \in N_T(v_i) : \phi(v) = \phi(\ell)\}$
      $x_\ell \leftarrow 1 + \max\{x_{\ell'} : \ell' \in V_{\phi(\ell)}\}$ $\triangleright$ x`-coordinate of` ℓ'
      Move ℓ to $\big(x_\ell,\, \phi(\ell)\big)$
    Draw edge v_i–ℓ
  foreach $(v, w) \in E_{K_3}$ **do** $\triangleright$ `where` $v \in V_S$
    if $\phi(v) = \phi(w) \pm 1$ **or** $x_w = x_v - 1$ **then** $\triangleright$ `coords differ by 1`
      Draw edge v–w $\triangleright$ `no edge bend needed`
    else
      if $\phi(v) < \phi(w)$ **then** $y_b \leftarrow \phi(v) + 1$ **else** $y_b \leftarrow \phi(v) - 1$
      Place bend b at $\big(x_v - 1,\, y_b\big)$ and draw edges v–b and b–w

Figure 7.26: DRAW-K_3-CATERPILLAR with duplicate labels

endpoints v_1 and v_k, respectively. In the case that G has the K_3 ending block B_0 that was replaced with the edge (u, v_1) in T', then the edge (u, w) is added to T', where w is the other vertex of B_0 and u becomes the new right endpoint of P. In the case that G has the K_3 ending block B_k that was replaced with the edge (v_k, u) in T', then the edge (v_k, w) is added to T, where w is the other vertex of B_0 and becomes the new left endpoint of P.

Algorithm: Place-K_3-Caterpillar-Leaves

▷ `Place the leaves adjacent to a vertex` v_i `at` $\big(x_i, \phi(v_i)\big)$

Input: Caterpillar $T(V, E, \phi)$ with duplicate labels ϕ, vertex v_i, list L_i of leaves in $N(v_i)$ sorted by ϕ

Output: Maximum x-coordinate x_{max} of all leaves in L_i

$x_{above} \leftarrow x_{below} \leftarrow x + 1$

$\ell_{above} \leftarrow$ first element of L_i ▷ `next leaf above` v_i

while $\ell_{above} \neq \varnothing$ **and** $\phi(\ell_{above}) > \phi(v_i)$ **do**

  $x_{above} \leftarrow x_{above} + 1$, place ℓ_{above} at $\big(x_{above},\ \phi(\ell_{above})\big)$

  Delete ℓ_{above} from L_i, $\ell_{above} \leftarrow$ first element of L_i

$\ell_{below} \leftarrow$ last element of L_i ▷ `next leaf below` v_i

while $\ell_{below} \neq \varnothing$ **and** $\phi(\ell_{below}) < \phi(v_i)$ **do**

  $x_{below} \leftarrow x_{below} + 1$, place ℓ_{below} at $\big(x_{below},\ \phi(\ell_{below})\big)$

  Delete ℓ_{below} from L_i, $\ell_{below} \leftarrow$ last element of L_i

return $x_{max} \leftarrow \max\{x_{above}, x_{below}\}$

Figure 7.27: Place-K_3-Caterpillar-Leaves with duplicate labels

Let E_{K_3} denote the extra edges that are in G but are not in T. Observe that each edge in E_{K_3} is in a distinct K_3 block; compare Fig. 7.23(a)–(b). Hence, all the edges of E_{K_3} have distinct endpoints. Let $V_{K_3} = V(E_{K_3})$ be these endpoints. We use the algorithm Sort-Caterpillar-Leaves from Fig. 4.20 in section 4.4 to sort the leaves adjacent to each vertex by their labeling ϕ in P in linear time.

We proceed to draw P edge by edge from its right endpoint v_1 to its left endpoint v_m and place leaves adjacent to each path vertex as we go. We start by placing the right endpoint v_1 of P at coordinate $\big(1,\ \phi(v_1)\big)$. For each edge (v_i, v_{i+1}) of P, we use the algorithm Place-K_3-Caterpillar-Leaves from Fig. 7.27 to place the leaves of T incident to v_i. Each leaf above (or below) v_i sorted by descending (or ascending) labels is placed one more unit to the right from v_i; see Fig. 7.28(c). The maximum x-coordinate x_{max} that this procedure takes is then returned.

The algorithm Place-Caterpillar-Leaves from Fig. 4.18 in section 4.4 placed the leaves more compactly by using the next available grid point in radial clockwise (or counterclockwise) sweep for the leaves above (or below) v_i; see Fig. 7.28(c). Both algorithms produce unique slopes for each incident leaf edge.

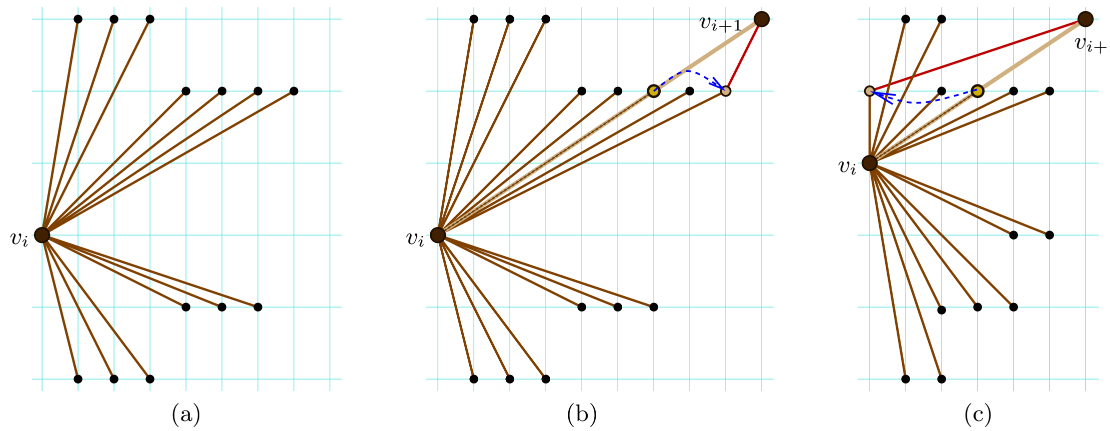


Figure 7.28: Comparison of Place-K_3-Caterpillar-Leaves in (a) and (b) to Place-Caterpillar-Leaves in (c) between how the two algorithms place leaves so that an overlapping leaf edge can be moved. Moving a leaf of a left-to-right caterpillar can be problematic if it is also a degree-2 vertex of a K_3 block in G as is evidenced by the crossing of the red edge in (c), while this problem is avoided in (b).

However, the only open grid position that an overlapping leaf can be moved to after being placed by Place-Caterpillar-Leaves is either above or below its adjacent vertex. This works when drawing a caterpillar, but the leaf ℓ being moved in a left-to-right caterpillar can be the endpoint of an edge $(\ell, v_{i+1}) \in E_{K_3}$, and hence, this placement of ℓ could lead to a crossing; see Fig. 7.28(c). By not using every available grid point when placing the leaves, an overlapping leaf can always be moved to the unit to right of all the other leaves on the same level as in see Fig. 7.28(b).

The edge v_i–v_{i+1} is drawn with next endpoint v_{i+1} placed $x_{max} + 2$ units to the left of v_i. The extra two units are required to accommodate drawing the edges from E_{K_3} later. Next, we must move any leaves that have been placed on the edges of P as well as any leaves that happen to be in V_{K_3}. Suppose that ℓ lies on the edge v_i–v_{i+1} as in Fig. 7.28(b). We move ℓ one unit to the right of the maximum x-coordinate of all leaves with the same label as ℓ; see Fig. 7.28(b). If a leaf ℓ is in V_{K_3}, then ℓ is the endpoint of the edge $(\ell, v_{i+1}) \in E_{K_3}$, and hence, also needs to be moved to the right of all the other leaves adjacent to v_i with the same label.

After moving all leaves of T to avoid overlaps with P and allow edges in E_{K_3} to be drawn, we can now draw all of the leaf edges in T. For each edge $(\ell, v_{i+1}) \in E_{K_3}$,

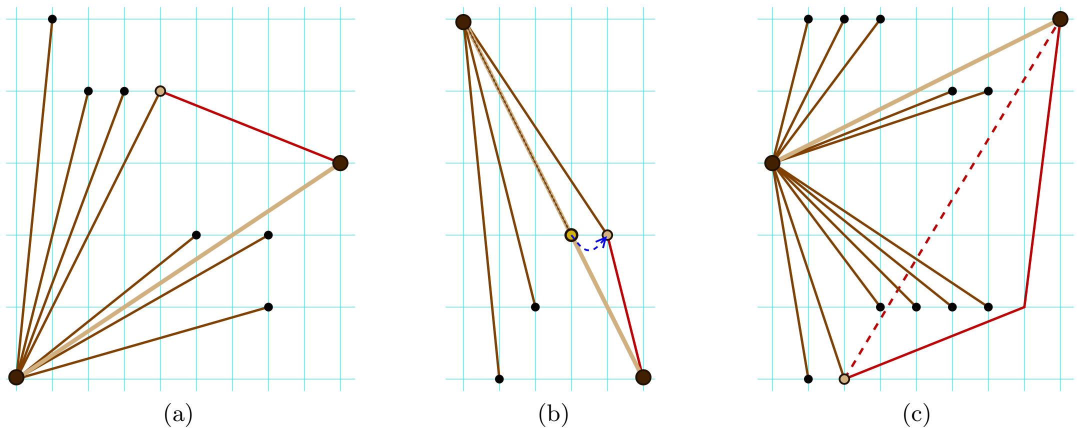


Figure 7.29: The extra edges in a K_3-caterpillar that are not in its left-to-right caterpillar can sometimes be drawn straight as in (a) and (b), but often require a bend as in (c).

we can draw the edge straight if $\phi(\ell) = \phi(v_{i+1}) \pm 1$ or ℓ lies one unit to the left of v_{i+1}; see Fig. 7.29(a)–(b). Otherwise, let b be the point $\big(x_{i+1} - 1,\, y_b\big)$ where x_{i+1} is the x-coordinate of v_{i+1} and y_b is $\phi(\ell) + 1$ if $\phi(\ell) > \phi(v_{k+1})$, and $\phi(\ell) - 1$ if $\phi(\ell) < \phi(v_{k+1})$. We observe that the bent edge ℓ–b–v_{i+i} will not cross any leaves since x_{i+1} is always two more than the maximum x-coordinate of the leaves of v_i so that the edge bend will never be placed on top of any leaf; see Fig. 7.29(c).

Aside from the $O(n)$-time calls to Get-Left-To-Right-Caterpillar, Sort-Caterpillar-Leaves, and Place-K_3-Caterpillar-Leaves, the algorithm Draw-K_3-Caterpillar places each vertex and draws each edge in $O(1)$ time, and thus, takes $O(n)$ time in total. Each of the m edges (v_i, v_{i+1}) of P with incident leaf leaves of v_i uses $(2 + \max\big\{\big|L_{above}(v_i)\big|,\, \big|L_{below}(v_i)\big|\big\}) \times k$ space for $i \in \{1, \ldots, m\}$. Hence, Draw-K_3-Caterpillar draws G on a $(2m + b) \times k$ grid for any labeling where $b = \sum_{i=1}^{m} \max\big\{\big|L_{above}(v_i)\big|,\, \big|L_{below}(v_i)\big|\big\}$. □

We observe that an n-vertex K_3-caterpillar with an m-edge left-to-right path has m internal vertices and $n - m$ leaves in the left-to-right caterpillar. Each leaf contributes at most once to the overall summation $b = \sum_{i=1}^{m} \max\big\{\big|L_{above}(v_i)\big|,\, \big|L_{below}(v_i)\big|\big\}$ of the previous lemma. Thus, the $(2m + b) \times k$ grid used by the lemma is at worst a $2(n-1) \times k$ grid (when K_3-caterpillar is simply a path). This allows us to restate Lemma 4.4.1 as follows:

Corollary 7.4.3. *K_3-caterpillars of order n on k levels are* ULP *for any $0 \le k \le n$. Each can be straight-line realized in $O(n)$ time within an $O(n) \times k$ grid for any labeling.*

7.5 Drawing G_ω with Duplicate Labels

We conclude this chapter by drawing a graph isomorphic to G_ω as in Fig. 7.30 in constant time.

Lemma 7.5.1. *A 6-vertex graph on k levels isomorphic to G_ω can be realized with straight-line edges in $O(1)$ time on a $3 \times k$ grid for any labeling.*

Proof. Figure 7.31 gives the algorithm Draw-G_ω that draws a graph isomorphic to G_ω on a $3 \times k$ grid for any labeling. Let B be the K_3 block of G where B is the 3-cycle u–v–w–u. We order the vertices of B by ϕ so that $\phi(u) < \phi(v) < \phi(w)$.

We place v and its leaf ℓ_v in the center at x-coordinate of 2. We place the leaf ℓ_w of w to the left at x-coordinate 1 and the leaf ℓ_u of u to the right at x-coordinate 3. If $\phi(\ell_v) > \phi(v)$ as in Fig. 7.30(a), then we place w to the left at x-coordinate 1 and place u below v. Otherwise, if $\phi(\ell_v) < \phi(v)$ as in Fig. 7.30(b), then we place w above v and place u to the right at x-coordinate 3. Then we draw all the edges.

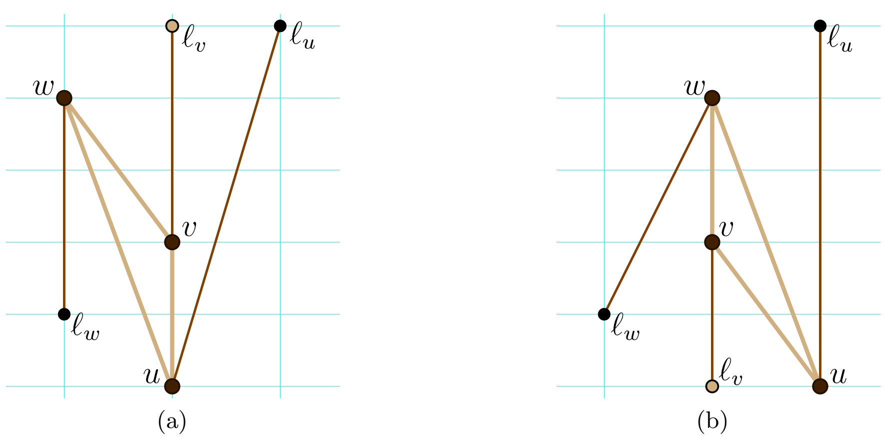


Figure 7.30: The 6-vertex graph G_ω has two distinct realizations on $3 \times k$ grid depending on whether $\phi(\ell_v) > \phi(v)$ as in (a) or $\phi(\ell_v) < \phi(v)$ as in (b).

Algorithm: DRAW-G_ω

$\triangleright$ `Draw` G_ω `with duplicate labels in` $O(1)$ `time`

Input: G_ω isomorphic graph $G(V, E, \phi)$ with duplicate labels ϕ
Output: Level planar drawing of G

$B \leftarrow K_3$ block of G, $L \leftarrow$ leaves G
$\{u, v, w\} \leftarrow V(B)$ **where** $\phi(u) < \phi(v) < \phi(w)$
$\{\ell_u\} \leftarrow N(u) \cap L$, $\{\ell_v\} \leftarrow N(v) \cap L$, $\{\ell_w\} \leftarrow N(w) \cap L$
Place ℓ_u at $\big(1,\, \phi(\ell_u)\big)$ and ℓ_w at $\big(3,\, \phi(\ell_w)\big)$
Place v at $\big(2,\, \phi(v)\big)$ and ℓ_v at $\big(2,\, \phi(\ell_v)\big)$
if $\phi(\ell_v) > \phi(v)$ **then**
  Place u at $\big(2,\, \phi(u)\big)$ and w at $\big(1,\, \phi(w)\big)$
else Place u at $\big(3,\, \phi(u)\big)$ and w at $\big(2,\, \phi(w)\big)$ $\triangleright$ `else` $\phi(\ell_v) < \phi(v)$
Draw edges u–v, u–w, v–w, u–ℓ_u, v–ℓ_v, and w–ℓ_w

Figure 7.31: DRAW-G_ω

A K_3 cannot self intersect when drawn with straight edges. Hence, we can see that there are no crossings by examining each of the leaf edges. The leaf ℓ_w is leftmost and the leaf ℓ_u is rightmost. Since w and u are either also leftmost or rightmost, respectively, or share the same x-coordinate as their leaves, the leaf edges (w, ℓ_w) and (u, ℓ_u) cannot cross any other edge in G. The leaf ℓ_v is drawn either above or below v. Since v is the middle vertex with respect to ϕ, then either $\phi(\ell_v) > \phi(v) > \phi(u)$, in which case u is drawn below v, or $\phi(\ell_v) < \phi(v) < \phi(w)$, in which case w is drawn above v. Hence, the leaf edge (v, ℓ_v) also does not cross any other edge in G. Therefore, DRAW-G_ω produces a level planar drawing of G on a $3 \times k$ grid in $O(1)$ time. $\square$

Together lemmas 7.4.2 and 7.5.1, which provide the drawing algorithms for K_3-caterpillars and the graph G_ω, give us our final theorem of this chapter.

Theorem 7.5.2. *K_3-caterpillars and graphs isomorphic to G_ω on $2 \leq k \leq n$ levels are ULP with duplicate labels and can be realized in $O(n)$ time.*

Chapter 8

Characterizing ULP Graphs

We define the following two sets of forbidden graphs:

Definition 8.0.1.

1. $\mathcal{F}_{\mathit{ULP}} = \{T_8, T_9, G_5, G_6, G_\alpha, G_\kappa, G_\delta\}$ *and*
2. $\mathcal{F}^*_{\mathit{ULP}} = \{T_7, C_4, G_\kappa\}$,

which we will show in this chapter are the forbidden ULP graphs with duplicate and distinct labels, respectively; see Fig. 8.1.

We begin this chapter by presenting labelings that force crossings in level planar drawings of the graphs in $\mathcal{F}_{\mathsf{ULP}}$ and $\mathcal{F}^*_{\mathsf{ULP}}$. Then, we show that these graphs are minimal in that the removal of an edge yields one or more ULP graphs. We conclude the chapter with our characterizations of ULP graphs, first for the case of distinct labels, and then for the case of duplicate labels.

8.1 Labelings of Forbidden ULP Graphs

Next we see that none of the seven graphs in $\mathcal{F}_{\mathsf{ULP}}$ is ULP with distinct labels.

Lemma 8.1.1. *There exist labelings that prevent each graph in $\mathcal{F}_{\mathit{ULP}}$ from having a planar realization on tracks.*

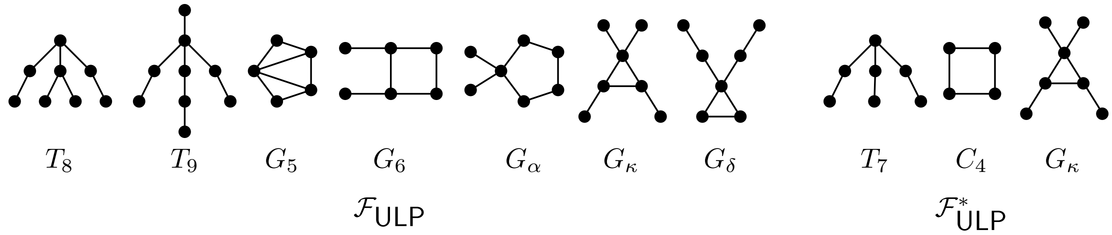
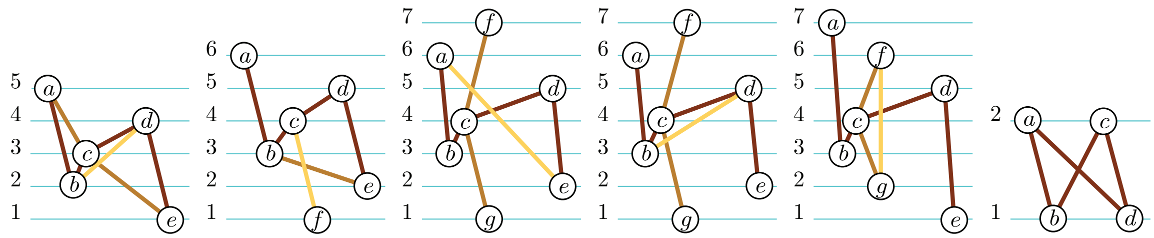


Figure 8.1: Forbidden graphs of $\mathcal{F}_{\mathsf{ULP}}$ and $\mathcal{F}^*_{\mathsf{ULP}}$



(a) G_5 **(b)** G_6 **(c)** G_α **(d)** G_κ **(e)** G_δ **(f)** C_4

Figure 8.2: Labelings preventing G_5, G_6, G_α, G_κ, G_δ, and C_4 from being ULP

Proof. The labelings of trees T_8 and T_9 were shown not to have planar realizations in Lemma 5.1.1. We need to do the same for the labelings of the remaining five unicyclic graphs in $\mathcal{F}_{\mathsf{ULP}}$ given in Fig. 8.2(a)–(e).

Let C denote the chain a–b–c–d–e where $\phi(a) > \phi(d) > \phi(c) > \phi(b) > \phi(e)$ in which C forms a backwards 'N'. If the left half of C (the chain a–b–c) and the right half of C (the chain c–d–e) intersects the track ℓ_c only to the left or only to the right of c, then some part of a–b–c crosses some part of c–d–e. Hence, we only need to consider embeddings where c lies between the edges a–b and d–e, i.e., one of those edges intersects the track ℓ_c to the left of c, while the other intersects ℓ_c to the right of c. To avoid a self crossing of C, a–b must intersect the tracks ℓ_c and ℓ_d on the same side of both vertices, i.e., intersect ℓ_c and ℓ_d both to the left or both to the right of c and d, respectively. The same applies for the edge d–e intersecting the tracks ℓ_b and ℓ_c on the same side of b and c, respectively. So we can assume w.l.o.g. that edge a–b intersects tracks ℓ_c and ℓ_d to the left of both c and d, while edge d–e intersects tracks ℓ_b and ℓ_c to the right of both b and c as in Fig. 8.2(a)–(e).

For G_5, edge a–b intersecting ℓ_d to the left d and edge d–e intersecting ℓ_b to the right of b means that the edge b–d must intersect ℓ_c to the right of where a–b and to the left of where d–e intersects ℓ_c. Hence, if edge b–d intersects ℓ_c to the left of c, then b–d crosses a–c (since a–b also intersects ℓ_c to the left of c). Otherwise, edge b–d intersects ℓ_c to the right of c so that b–d crosses c–e (since b–e also intersects ℓ_c to the right of c) as in Fig. 8.2(a).

For G_6, from the assumptions, edge c–f either crosses

(i) a–b if c–f intersects ℓ_b to the left of b (since a–b intersects ℓ_c to the left of c),

(ii) d–e if c–f intersects ℓ_e to the right of e (since d–e intersects ℓ_c to the right of c), or

(iii) b–e, otherwise (since c–f must then intersect ℓ_b to the right of b and intersect ℓ_e to the left of e as in Fig. 8.2(b)).

In G_α, G_δ and G_κ, for c–f and c–g not to cross C, c–f must intersect ℓ_d to the left of d, while c–g must intersect ℓ_b to the right of b. Also in G_α and G_κ, c–f must intersect ℓ_a to the right of a, while c–g must intersect ℓ_e to the left of e. Also in G_δ, a–b must intersect ℓ_f to the left of f, while d–e must intersect ℓ_g to the right of g.

In G_α, for a–e to avoid crossing C as in Fig. 8.2(c), a–e must either intersect ℓ_d to the right of d, where it crosses c–f, or ℓ_b to the left of b, where it crosses c–g.

In G_κ, if b–d intersects ℓ_c to the right of c as in Fig. 8.2(d), then b–d crosses c–g. Otherwise, b–d intersects ℓ_c to the left of c and crosses c–f.

In G_δ, if f–g intersects ℓ_b to the left of b or ℓ_d to the right of d, then f–g crosses a–b or d–e, respectively. Else assume that f–g intersects ℓ_b to the right of b and ℓ_d to the left of d. If f–g intersects ℓ_c to the right of c as in Fig. 8.2(e), then f–g crosses c–d. Otherwise f–g intersects ℓ_c to the left of c so that f–g crosses b–c. $\square$

For the case of duplicate labels, we see that none of the three graphs in $\mathcal{F}^*_{\mathsf{ULP}}$ is ULP either.

Lemma 8.1.2. *There exist labelings that prevent each graph in $\mathcal{F}^*_{\mathsf{ULP}}$ from having a planar realization on tracks.*

Proof. The labelings of T_7 and G_κ were shown not to have planar realizations in Lemmas 5.1.2 and 8.1.1. For C_4, let C be the 4-cycle a–b–c–d–a where ϕ obeys $\phi(a) = \phi(c) > \phi(b) = \phi(d)$ for $2 \geq k \geq n$ as in Fig. 8.2(f). W.l.o.g. assume that the subpath a–b–c–d of C proceeds left to right in order to avoid self intersections. This means that a lies to the left of c along track ℓ_a and b lies to the left of d along the track ℓ_b. However, for a–d not to cross b–c, if b lies to the left of d along ℓ_b, then a must lie to the right of c along ℓ_a. Hence, C must be self intersecting. $\square$

Lemma 5.1.3 allows us to generalize Lemmas 8.1.1 and 8.1.2 into the following pair of corollaries:

Corollary 8.1.3. *If a graph G contains a subgraph homeomorphic to a graph in $\mathcal{F}_{\mathsf{ULP}}$, then G cannot be* ULP *with distinct labels.*

Corollary 8.1.4. *If a graph G contains a subgraph homeomorphic to a graph in $\mathcal{F}^*_{\mathsf{ULP}}$, then G cannot be* ULP *with duplicate labels.*

8.2 Minimality of Forbidden ULP Graphs

The next lemma shows that the seven forbidden graphs of $\mathcal{F}_{\mathsf{ULP}}$ are minimal; the removal of an edge from any of the seven yields one or more graphs from $\mathcal{G}_{\mathsf{ULP}}$.

Lemma 8.2.1. *Each forbidden graph in $\mathcal{F}_{\mathsf{ULP}}$ is minimal in that the removal of an edge yields one or more generalized caterpillars, radius-2 stars, extended 3-spiders, or extended K_4 subgraphs.*

Proof. The removal of an edge from T_8 or T_9 yields one or more of caterpillars, radius-2 stars, and degree-3 spiders (all members of $\mathcal{G}_{\mathsf{ULP}}$) by Lemma 5.2.1.

For G_5 in Fig. 8.2(a) where b–a–c–e–d–b forms a 5-cycle, the removal of either chord b–c or c–d forms an EK4, while the removal of any other edge forms a GC. For G_6 in Fig. 8.2(b) where b–c–d–e–b forms a 4-cycle, the removal of either pendant edge a–b or c–f leaves a GC, while the removal of any internal edge yields a caterpillar (also a GC). For G_α in Fig. 8.2(c) where a–b–c–d–e–a forms a 5-cycle, the removal of either pendant edge c–f or c–g leaves an E3S, while the removal of any internal edge yields a caterpillar (a GC). For G_κ in Fig. 8.2(d) where b–c–d–b forms a 3-cycle, the removal of either pendant edge c–f or c–g leaves a graph isomorphic to G_ω (an E3S), while the removal of any other edge leaves a GC. For G_δ in Fig. 8.2(e) where c–f–g–c forms a 3-cycle and a–b–c and c–d–e are chains, the removal of a chain edge leaves a GC, and possibly, a lone edge. The removal of cycle edge f–g leaves a caterpillar (a GC), and the removal of one of the other cycle edges c–f or c–g leaves a graph isomorphic to T_7 (an E3S). □

Next, we show that the three forbidden graphs of $\mathcal{F}^*_{\mathsf{ULP}}$ are also minimal.

Lemma 8.2.2. *Each forbidden graph in $\mathcal{F}^*_{\mathsf{ULP}}$ is minimal in that the removal of an edge yields one or more K_3-caterpillars or a graph isomorphic to G_ω.*

Proof. Showing that the removal of an edge from T_7 yielded a forest of caterpillars was done in Lemma 5.2.2. For C_4 in Fig. 8.2(f), the removal of any edge leaves a path. Both are special cases of K_3-caterpillars. For G_κ in Fig. 8.2(d) where c–f–g–c forms a 3-cycle, the removal of either pendant edge c–f or c–g leaves a graph isomorphic to G_ω, while the removal of any other edge leaves a K_3-caterpillar. □

8.3 Characterizing ULP Graphs with Distinct Labels

As we shall see, the cycle C_5 is a forbidden graph of the ULP blocks that compose generalized caterpillars. The next lemma describes these 2-connected subgraphs.

Lemma 8.3.1. *The only 2-connected subgraphs that have cycle length $k \leq 4$ are subgraphs isomorphic to P_2, K_4, or $(K_3)^m$ or $(C_4)^m$ for some $m \geq 1$.*

Proof. Let $B = G[U]$ be a 2-connected subgraph induced on $G(V, E)$ for some $U \subseteq V$. If $|U| \leq 4$ then B is isomorphic to a 2-connected subgraph of K_4, which is either P_2, $K_3 = (K_3)^1$, $(K_3)^2$, $C_4 = (C_4)^1$, or K_4.

Otherwise, assume that $|U| > 4$ where the cycle length of B is $k \leq 4$. If all the vertices of B have degree 2, then B would be a k-cycle where $k = |U| > 4$. Hence, B must contain at least two vertices u and v such that $deg(u) > 2$ and $deg(v) > 2$.

Suppose that there is a vertex $x \in N(u)$ such that $x \notin N[v]$. Since B is 2-connected, there are at least two internally disjoint paths p_1 and p_2 from u to v such that x is along p_1. Path p_1 must have length $|p_1| \geq 3$, since otherwise $x \in N[v]$. Thus, p_1 must contain a vertex $y \in N(v)$ such that $y \neq x$ and $y \notin N(u)$. If $|p_1| \geq 4$ and $|p_2| \geq 1$ or if $|p_1| = 3$ and $|p_2| \geq 2$, then p_1 and p_2 form a k-cycle for some $k \geq 5 > 4$. Hence, $|p_1| = 3$ and $|p_2| = 1$ so that $p_1 = u$–x–y–v and $p_2 = u$–v.

Since $deg(u) > 2$, let $s \in N(u) \setminus \{v, x\}$. Given that B is 2-connected, there must be a path p_3 connecting s to a vertex t in $\{v, x, y\}$ such that p_3 is disjoint from vertices

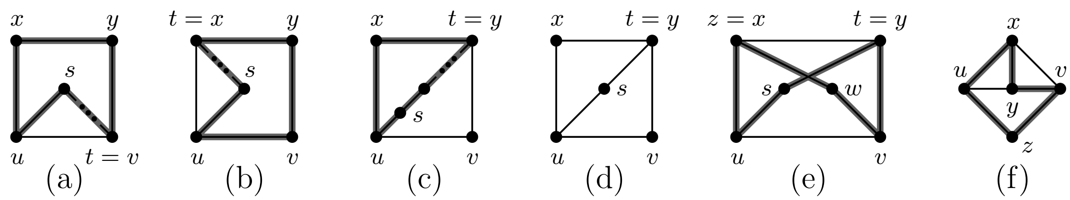


Figure 8.3: Cases for when not all neighbors of u and v are common

in $\{u, v, x, y\} \setminus \{t\}$. If $t = v$ or $t = x$, then u–s–$\cdots$–v–y–x–u or u–s–$\cdots$–x–y–v–u would form a k-cycle for $k \geq 5$; see Fig. 8.3(a)–(b). Hence, $t = y$. However, the cycle u–s–$\cdots$–y–x–u has length $k = 3 + |p_3|$. Either $|p_3| \geq 2$ as in Fig. 8.3(c) forming a k-cycle for $k \geq 5$, or $p_3 = s$–y as in Fig. 8.3(d). Hence, $s \in N(u) \cap N(y) \setminus \{v, x\}$. Analogously, there must be a vertex $w \in N(v) \cap N(x) \setminus \{u, y\}$ and a path p_4 from w to vertex $z = x$ where $p_4 = w$–x. However, then u–s–y–v–w–x–u would form a 6-cycle; see Fig. 8.3(e).

Thus, u and v only have neighbors in common (except for possibly each other). Let $S = N(u) \cap N(v)$ be the set of common neighbors. Since $|U| > 4$, then $|S| \geq 3$. If x and y are adjacent vertices in S and z is a third vertex in S, then u–x–y–v–z–u forms a 5-cycle; see Fig. 8.3(f). Hence, all the vertices in S are non-adjacent so that B is either $P_2 \vee \overline{K_{|S|}}$ $\big($i.e., $(K_3)^m$ for some $m = |S| \geq 3\big)$ if (u, v) is in B, or $\overline{P_2} \vee \overline{K_{|S|}}$ $\big($i.e., $(C_4)^m$ for some $m = |S| - 1 \geq 2\big)$ if (u, v) is not in B. ☐

We can restate the previous lemma in terms of C_5.

Corollary 8.3.2. *Every block of a connected graph G is isomorphic to P_2, K_4, or $(K_3)^m$ or $(C_4)^m$ for some $m \geq 1$, or G contains a C_5 subdivision.*

As we shall also see, G_ω is a forbidden graph of generalized caterpillars. The next lemma provides a tool to extract a G_ω subgraph.

Lemma 8.3.3. *If a graph G has a k-block B for some $k \geq 3$, then G contains a subgraph homeomorphic to G_ω.*

Proof. Since B is a k-block for some $k \geq 3$, then B must have three connectors v_1, v_2, and v_3 with incident edges e_1, e_2, and e_3, respectively, from three other blocks; see Fig. 8.4. Let p_1 be the shortest path connecting v_1 to v_2, p_2 be the shortest

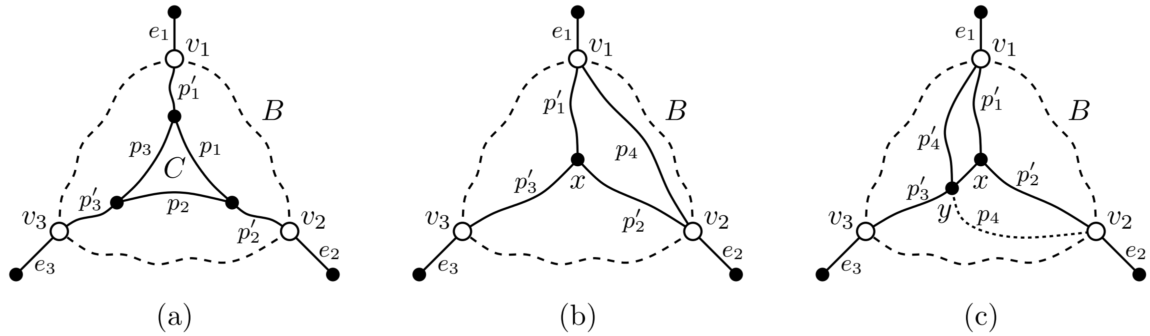


Figure 8.4: Cases for a 3-block forming G_ω

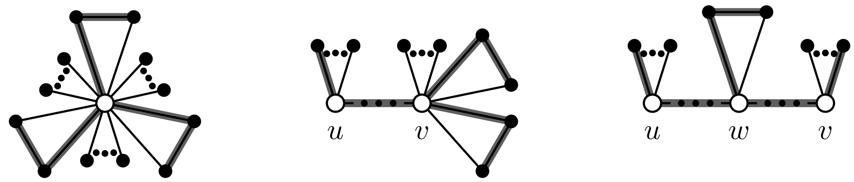
path connecting v_2 to v_3, and p_3 be the shortest path connecting v_3 to v_1. The paths pairwise share internally disjoint subpaths p'_1, p'_2, and p'_3 from endpoints v_1, v_2, and v_3, respectively, to either (i) a cycle C that is internally disjoint from p'_1, p'_2, and p'_3 as in Fig. 8.4(a) or (ii) a common endpoint x as in Fig. 8.4(b)–(c).

In case (i), the edges e_1, e_2, and e_3 and the paths p_1, p_2, and p_3 form a subgraph homeomorphic to G_ω. In case (ii), since B is 2-connected, there must be path p_4 connecting v_1 and v_2 that is internally disjoint from $p_1 = p'_1 + p'_2$ where either p_4 is disjoint from p'_3 as in Fig. 8.4(b) or there is some initial subpath p'_4 of p_4 to some vertex y of p'_3 as in Fig. 8.4(c). In the first case, the edges e_1 and e_2 and the paths p'_1, p'_2, p'_3, and p_4 form a subgraph homeomorphic to G_ω. In the second case, the edge e_1 and the paths p'_1, p'_2, p'_3, and p'_4 form a subgraph homeomorphic to G_ω. $\square$

The next lemma shows that a tree generalized with ULP joining and ending blocks must be a caterpillar in order for the resulting graph not to contain a subgraph homeomorphic to T_7.

Lemma 8.3.4. *If G is a connected graph with no subgraph isomorphic to T_7 where every block is either a pendant edge or an ULP joining or ending block from a set of ULP blocks $\mathcal{B}$, then G must be a caterpillar generalized with blocks from $\mathcal{B}$.*

Proof. Assume that the antecedent holds. Since every non-trivial block B is a 1-block or a 2-block, we can replace each such block B in G with an edge to obtain a tree T. If T is not a caterpillar, then G would contain a subgraph isomorphic to T_7 by Theorem 5.4.1.



(a) (b) (c)

Figure 8.5: Cases for when a generalized graph can contain a T_7 subgraph

First suppose that T is a star. If G has three or more non-trivial blocks, then they must share a common vertex, and hence, would contain a subgraph isomorphic to T_7, where each leg of length 2 is from a distinct non-trivial block; see Fig. 8.5(a).

Next suppose that T is not a star, where S is the spine of T with distinct endpoints u and v. If two non-trivial blocks B_1 and B_2 end on v (or u), then G would contain a subgraph isomorphic to T_7, where two of the length-2 legs come from B_1 and B_2, and the third length-2 leg comes from S, and possibly, a pendant edge incident to u; see Fig. 8.5(b). Finally, if a non-trivial block B in G has a connector that is an internal vertex w of S (i.e., B corresponds to a leaf edge in T incident to w), then G also would contain a subgraph isomorphic to T_7, where one length-2 leg comes from B and the other two come from S, and possibly, leaf edges e_1 and e_2 incident to u and v, respectively; see Fig. 8.5(c).

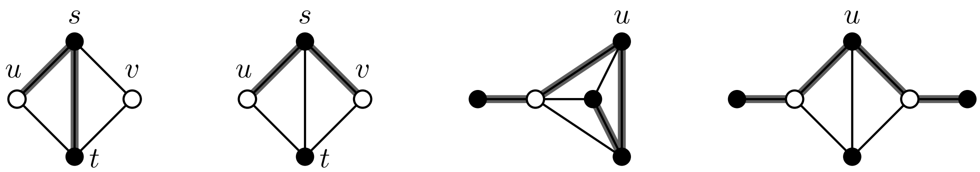
Therefore, for G not to contain a subgraph isomorphic to T_7, G must be a caterpillar generalized with ULP blocks from $\mathcal{B}$ according to Definition 2.3.4. $\square$

The next two observations show that all ULP graphs only have ULP trees for spanning trees as would be expected.

Observation 8.3.5. *Every spanning tree of a* GC *is a caterpillar, or equivalently, a* GC *does not contain a subtree isomorphic to* T_7*.*

Proof. Let G be the GC in question. Suppose T is a spanning tree of G that is not a caterpillar. Then by Theorem 5.4.1, T contains a subtree T' isomorphic to T_7 with root vertex r that has a corresponding vertex u in G.

Let G' be the blocks and leaf edges of G that contain edges of T', which is clearly a GC. If u is a cut vertex in G', then let G'_1 and G'_2 be two GCs in which their union



(a) (b) (c) (d)

Figure 8.6: Cases for when a GC can have internally disjoint length-2 paths

is G' and their intersection is u. Suppose w.l.o.g. that two of the three of legs of T' are contained in G'_1. However, this cannot happen. By definition, the vertex u of G'_1 is the connecting vertex of a $(K_3)^*$, $(C_4)^*$, diamond, or K_4 block. For a $(K_3)^*$, $(C_4)^*$, or K_4 block, $ecc(u) \leq 2$, which immediately prevents two internally disjoint paths of length 2 from u. While $ecc(u) = 3$ for a diamond block on the vertices $\{u, v, s, t\}$, any path of length 2 either uses both vertices s and t or one of the vertices s or t and the vertex v, again preventing two internally disjoint paths of length 2; see Fig. 8.6(a)–(b).

On the other hand, if u is not a cut vertex, then u must be some non-connector in one of the four types of ULP blocks. For a $(K_3)^*$ or $(C_4)^*$ block, this implies $\deg(u) = 2$, which contradicts $deg(u) \geq 3$ given that u corresponds to the root r of T_7. For a diamond or K_4 block with two or three non-connectors, respectively, each with degree 3, only two internally disjoint paths of length 2 can originate from such a non-connector u; see Fig. 8.6(c)–(d). This prevents u from corresponding to the root r of T_7 consisting of three internally disjoint length-2 paths with the common endpoint r. □

Observation 8.3.6. *Every spanning tree of an E3S and of an EK4 is either a degree-3 spider or a path.*

Proof. Exactly one vertex v of a degree-3 spider has $deg(v) = 3$ where all the other vertices have degree 1 or 2. Since an E3S either has one or three vertices of maximum degree 3, all of which are neighbors, any spanning tree T has at most one vertex of degree 3, i.e., T is a degree-3 spider or a path; see Fig. 8.7(a)–(b).

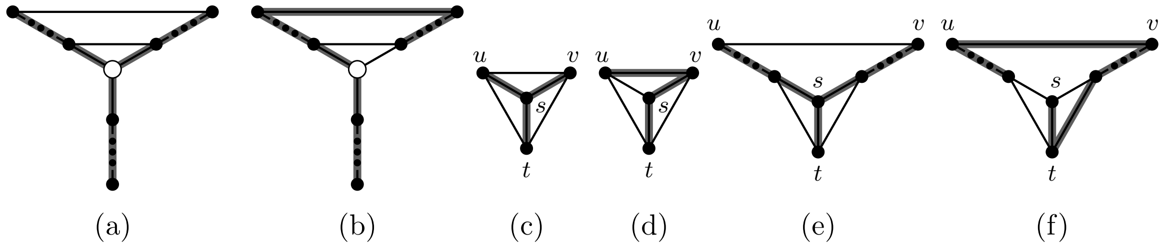


Figure 8.7: Spanning trees of extended 3-spiders and extended K_4's

Every spanning tree of a K_4 is a claw or the path P_4; see Fig. 8.7(c)–(d). For an extended K_4 in which the edge (u, v) is subdivided, only two of the four vertices of degree 3 are nonadjacent, namely u and v. However, since they have common neighbors s and t, any spanning tree can have at most one vertex of degree 3. Hence, any spanning tree is either a degree-3 spider or a path; see Fig. 8.7(e)–(f). The same holds for any connected subgraph, i.e., for an EK4. ☐

Before we fully characterize the graphs of $\mathcal{G}_{\mathsf{ULP}}$ in terms of the graphs in $\mathcal{F}_{\mathsf{ULP}}$, we first characterize GCs in terms of the four forbidden graphs given in Fig. 8.8.

Lemma 8.3.7. *A connected graph G is a generalized caterpillar if and only if G does not contain a subgraph homeomorphic to G_6, C_5, G_ω, or T_7.*

Proof. To show necessity, suppose that G is a GC. By Observation 8.3.5, G does not contain a subtree isomorphic to T_7. By Corollary 8.3.2, G does not have a subgraph homeomorphic to C_5, since every block is isomorphic to P_2, K_4, or $(K_3)^m$ or $(C_4)^m$ for some $m \geq 1$. Given that every block of GC is one of the ULP ending or joining blocks, there is no pair of vertices x and y such that x and y (i) each have degree at least 3, (ii) are in a 4-cycle C_4, (iii) are adjacent in C_4, and (iv) can each have an incident edge disjoint from the C_4 in G; see Fig. 2.2. Thus, there is no C_4

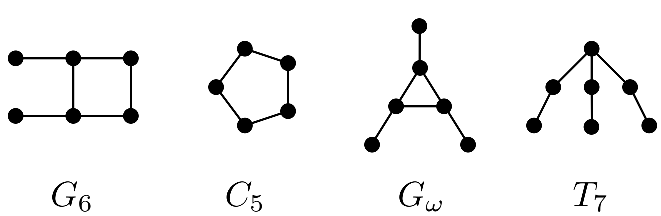


Figure 8.8: The four forbidden graphs of generalized caterpillars

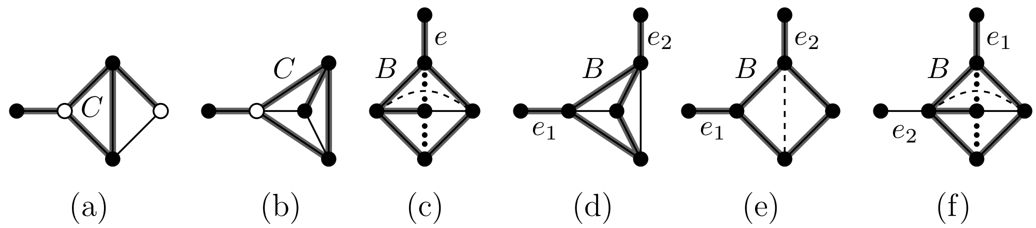


Figure 8.9: Cases for Lemma 8.3.7 for when an ULP block may contain G_ω or G_6

subdivision in G that can correspond to the 4-cycle of G_6, and hence, G does not contain a subgraph homeomorphic to G_6.

Finally, suppose that G contains G_ω as a homeomorphic subgraph H. The only ULP blocks of Definition 2.3.3 that contain a cycle C with three minimum degree-3 vertices are either K_4 or diamond blocks. However, a K_4 block B can only be an ULP ending block and each non-connector of diamond block B has degree 3 in G. In either case, any cycle C that could correspond to the 3-cycle of H can have at most two disjoint edges in G incident to C; see Fig. 8.9(a)–(b). Therefore, G cannot contain a subgraph homeomorphic to G_ω.

To show sufficiency, next suppose that G does not contain a subgraph homeomorphic to G_6, C_5, G_ω, or T_7 from Fig. 8.8. By Lemma 8.3.1, for G to have no k-cycle for some $k \geq 5$ (which would give a subgraph homeomorphic to C_5), each block must be isomorphic to P_2, K_4, or $(K_3)^m$ or $(C_4)^m$ for some $m \geq 1$. Additionally, every block in G must either be a 1-block or a 2-block, or G would contain a subgraph homeomorphic to G_ω by Lemma 8.3.3.

Suppose B is a 1-block with connector v. If B is isomorphic to P_2, then B is a pendant edge incident to v. If B is isomorphic to K_4, K_3, $(K_3)^2$, or C_4, then B is an ULP block ending on v. If B is isomorphic to $(K_3)^{m+1}$ or $(C_4)^m$ for some $m \geq 2$, then B is either an ULP block ending on v if $deg(v) > 2$ in $G[B]$, or $B + e$ (where e is an edge incident to v in G) would contain a subgraph isomorphic to G_6 as in Fig. 8.9(c). Suppose B is a 2-block with connectors v_1 and v_2 such that $deg(v_1) \leq deg(v_2)$ in $G[B]$. Let e_1 and e_2 be edges not in B that are incident to v_1 and v_2, respectively.

There are five cases for B to consider:

1. If B is isomorphic to K_4, then $C + e_1 + e_2$ would form a subgraph isomorphic to G_6 where C is any 4-cycle in B as in Fig. 8.9(d).

2. If B is isomorphic to K_3, then B is a K_3 block that joins v_1 to v_2.

3. If B is isomorphic to $(K_3)^2$, then either B is a diamond or $(K_3)^2$ block that joins v_1 to v_2, or $B + e_1 + e_2$ would contain a G_6 subgraph if $deg(v_1) = 2$ and $deg(v_2) = 3$ in $G[B]$ as in Fig. 8.9(e).

4. If B is isomorphic to C_4, then either B is a C_4 block that joins v_1 to v_2, or $B + e_1 + e_2$ would contain a G_6 subgraph as in Fig. 8.9(e).

5. If B is isomorphic to $(K_3)^{m+1}$ or $(C_4)^m$ for some $m \geq 2$, then either B is a $(K_3)^{m+1}$ or $(C_4)^m$ block that joins v_1 to v_2, or $B + e_1$ would contain a G_6 subgraph if $deg(v_1) = 2$ in $G[B]$ as in Fig. 8.9(f).

Hence, every block in G is a pendant edge or an ULP joining or ending block. Since G does not contain a homeomorphic subgraph of T_7, G cannot contain a subgraph isomorphic to T_7. As a result, G must be a caterpillar generalized with $(K_3)^*$, $(C_4)^*$, diamond, and K_4 blocks by Lemma 8.3.4. Therefore, G is a GC by Definition 2.3.6. □

According to Lemma 8.3.7, GCs have four forbidden graphs: G_6, C_5, G_ω, or T_7, only one of which, $G_6 \in \mathcal{F}_{\mathsf{ULP}}$. For the remaining three forbidden graphs, we determine which graphs can contain one of these forbidden graphs and still can be ULP with distinct labels. We start with G_ω with is a subgraph of G_κ.

Lemma 8.3.8. *If G is a connected graph that contains a subgraph homeomorphic to G_ω, but does not contain a subgraph homeomorphic to G_6, G_κ, or T_8, then G is an extended 3-spider.*

Proof. Suppose that the antecedent holds where H is a subgraph of G homeomorphic to G_ω and C is the cycle in H. If C is a k-cycle for some $k \geq 4$, then H would contain a subgraph homeomorphic to G_6. Thus, C is a 3-cycle.

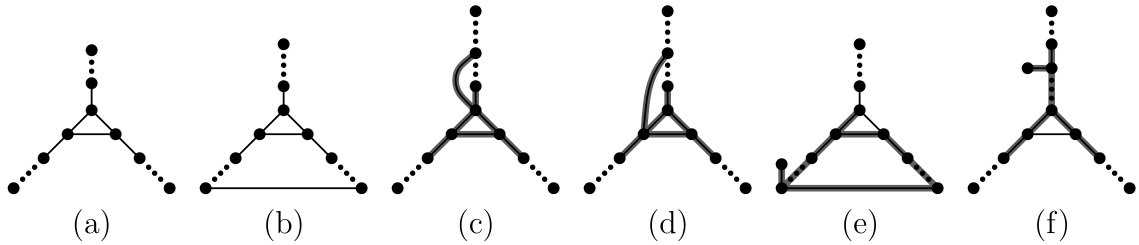


Figure 8.10: Cases for when an ULP graph can contain a subgraph homeomorphic to G_ω

We define a sequence of graphs $H_1, \ldots, H_m$, where $H_1 = H$, $H_m = G$, and $H_{i+1} = H_i + e_i$, where e_i is an edge in $G - H_i$ that is incident to H_i. We contend that the following two invariants are maintained:

(1) H_i is an E3S that contains three degree-3 vertices and either one or three pendant vertices as in Fig. 8.10(a)–(b) respectively; and

(2) e_i is a pendant edge incident to a pendant vertex of H_i or e_i is an edge connecting two of the pendant vertices of H_i.

Suppose that the invariants holds for some $i \geq 1$. We show that they also hold for $i + 1$. If e_i is incident to a degree-3 vertex of H_i, then e_i must be incident to C so that $H + e_i$ would be isomorphic to G_κ if one or both endpoints of e_i are incident to H_i as in Fig. 8.10(c)–(d). If e_i is incident to a pendant vertex of H_i, then the invariant would be maintained. The only other possibility is for e_i to be incident to a degree-2 vertex v of H_i. If v is a vertex along a cycle, then $H_i + e_i$ would contain a subtree homeomorphic to G_6 as in Fig. 8.10(e). Otherwise, v is a vertex along a path to a pendant vertex, and $H_i + e_i$ would contain a subgraph homeomorphic to T_8 as in Fig. 8.10(f). $\square$

Any graph with a cycle of length k has a C_k subdivision. We now determine which graphs with a C_5 subdivision can be ULP with distinct labels.

Lemma 8.3.9. *If G is a connected cyclic graph with a k-cycle for some $k \geq 5$ where G does not contain a subgraph homeomorphic to G_5, G_6, or T_8, then G is either an extended 3-spider or an extended K_4 subgraph.*

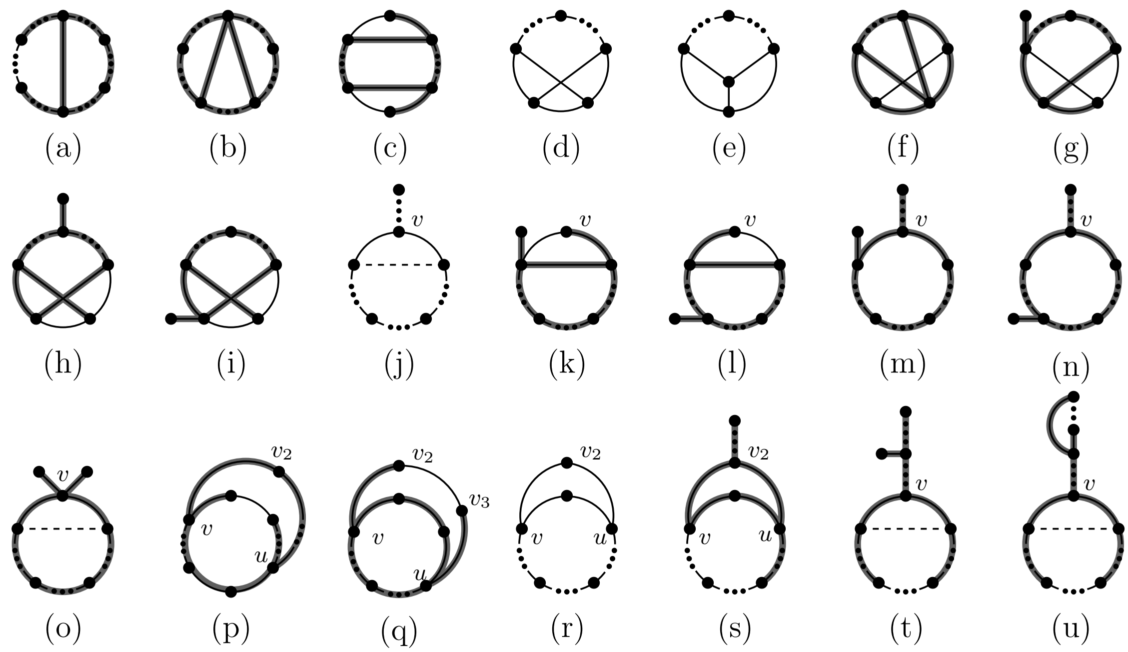


Figure 8.11: Cases for when an ULP graph can contain a subgraph homeomorphic to C_5

Proof. Suppose that the antecedent holds where C is a k-cycle for some $k \geq 5$ such that C is a longest cycle in G. If G contains a chord e whose endpoints are not adjacent along C, then $C + e$ contains a subgraph homeomorphic to G_6; see Fig. 8.11(a). Hence, any chord of C must have endpoints that are adjacent along C. If C has two chords e_1 and e_2 that are incident, then $C + e_1 + e_2$ is homeomorphic to G_5; see Fig. 8.11(b).

Assume then that G has two non-incident chords e_1 and e_2. Let p_1 and p_2 be the two paths that form C whose endpoints are the same as e_1. If both endpoints of e_2 lie along p_1 or lie along p_2, then $C+e_1+e_2$ contains a subgraph homeomorphic to G_6; see Fig. 8.11(c). If an endpoint of e_1 is adjacent to both endpoints of e_2 along C, and vice versa, as in Fig. 8.11(d), then $C+e_1+e_2$ forms a diamond-cycle; see Fig. 8.11(e). Here the chords e_1 and e_2 "cross" inside of C. There cannot be a third such chord that crosses one chord without being incident to the other; see Fig. 8.11(f). Hence, C has at most two chords. Moreover, if C has two chords where $V(G) \neq V(C)$, (i.e., $G - G[V(C)]$ is non-empty), then G has an edge incident to a vertex of C as in Fig. 8.11(g)–(i) where G would then have a subgraph homeomorphic to G_6.

Otherwise, G contains at most one chord e whose endpoints have a common neighbor v along C. We define a sequence of graphs $H_1, \ldots, H_m$, where either $H_1 = C + e$ (if C has a chord in G) or $H_1 = C$ (otherwise), $H_m = G$, and $H_{i+1} = H_i + e_i$, where e_i is an edge in $G - H_i$ that is incident to H_i such that $e_i \neq e$. We contend that the following two invariants are maintained:

(1) H_1 is an EK4 and H_i is an E3S that contains either one or three degree-3 vertices and one pendant vertex for $2 \leq i < m$ as in Fig. 8.11(j); and

(2) e_i is a pendant edge incident to the pendant vertex of H_i.

Define $v_1 = v$ and v_i to be the pendant vertex of H_i for $i \geq 2$, and define p_i to be the path from $v \rightsquigarrow v_i$. Clearly, H_1 is either a cycle or a K_3-cycle, which are both EK4s. Since G can have at most one chord and $e_i \neq e$, e_1 must be a pendant edge incident to C. If G has the chord e, then e_1 must be incident to v, since otherwise $H_2 = H_1 + e_1$ would contain a subgraph homeomorphic to G_6; see Fig. 8.11(k)–(l). If C has no chord, we can assume w.l.o.g. that e_1 is incident to v. Hence, H_2 is an E3S with one pendant vertex and contains one or three degree-3 vertices, which are v and the endpoints of chord e (if present) in G; see Fig. 8.11(j).

We assume that the invariants holds for some $i \geq 2$ and will show that they also hold for $i + 1$. If e_i is a pendant edge incident to a pendant vertex of H_i, then both invariants hold. Edge e_i cannot be a chord of C since C has at most one chord and $e_i \neq e$. If e_i is a pendant edge incident to some vertex on C other than v, then $C + p_2 + e_i$ would be homeomorphic to G_6; see Fig. 8.11(m)–(n). If e_i is incident to v, then H_{i+1} would contain a subgraph homeomorphic to G_α; see Fig. 8.11(o).

Suppose that e_i connects v_j for some $j \geq 2$ to some vertex u in C other than v. Let p and q be the two paths with common endpoints u and v that form C where $|p| \leq |q|$ so that $|p| + |q| = |C| = k$. Let C' be the cycle formed by paths p_j and q and the edge e_i. As a result,

$$|C'| = |p_j| + |q| + |e_1| = (j - 1) + (k - |p|) + 1 = k + j - |p|.$$

Since k is the length of a longest cycle, $|p| \geq j$. If $|p| > j$, then $|p| \geq 3$ and $|q| \geq 3$ (since $|q| \geq |p| > j \geq 2$) so that q, p_j, e_i, and the two pendant edges of p form a subgraph homeomorphic to G_6; see Fig. 8.11(p). Therefore, $|p| = j$. If $j > 2$, then $C+p_2+e_2$ would be homeomorphic to G_6; see Fig. 8.11(q). The only remaining possibility is if $j = 2$ as in Fig. 8.11(r). If $i = j = 2$, then $H_{i+1} = H_3$ would be a C_4-cycle. Otherwise, $i > j$ and H_{i+1} would contain subgraph homeomorphic to G_6; see Fig. 8.11(s).

Thus, we can assume that e_i is not incident to any vertex in C, but is incident to v_j for some $j \geq 2$. Either e_i is a pendant edge, or is incident to some v_l for some $j \neq l$. Regardless, H_{i+1} would contain a subgraph homeomorphic to T_8; see Fig. 8.11(t)–(u). □

Finally, we determine which graphs that contain a T_7 subgraph can be ULP with distinct labels.

Lemma 8.3.10. *If G is a connected graph that contains a subgraph isomorphic to T_7, but does not contain a subgraph homeomorphic to G_6, G_α, G_κ, G_δ, T_8, or T_9, then G is either an E3S or a R2S.*

Proof. Suppose that the antecedent holds where H is a subgraph isomorphic to T_7 with root r. There are two cases: either $deg(r) = 3$ or $deg(r) \geq 4$. In the first case, we will show that G must be an E3S as in Fig. 8.12(a), and in the second case, we will show that G must be an R2S as in Fig. 8.12(b).

In either case, we define a sequence of graphs $H_1, \ldots, H_m$, where either $H_1 = H$, $H_m = G$, and $H_{i+1} = H_i + e_i$, where e_i is an edge in $G - H_i$ that is incident to a vertex in H_i with the following constraint: if there is a vertex $u \in V(G) \setminus V(H_i)$ that is adjacent to some vertex $v \in V(H_i)$, then e_i cannot be an internal edge in $G[V(H_i)]$. This will ensure that whenever possible $|V(H_{i+1})| = |V(H_i)| + 1$.

In both cases, edge e_i cannot be any of the following three types of edges:

(i) an edge (x, y) where $x \in N(r)$, but $y \notin N(r)$ as in Fig. 8.12(c) since G would then have a subgraph homeomorphic to G_6;

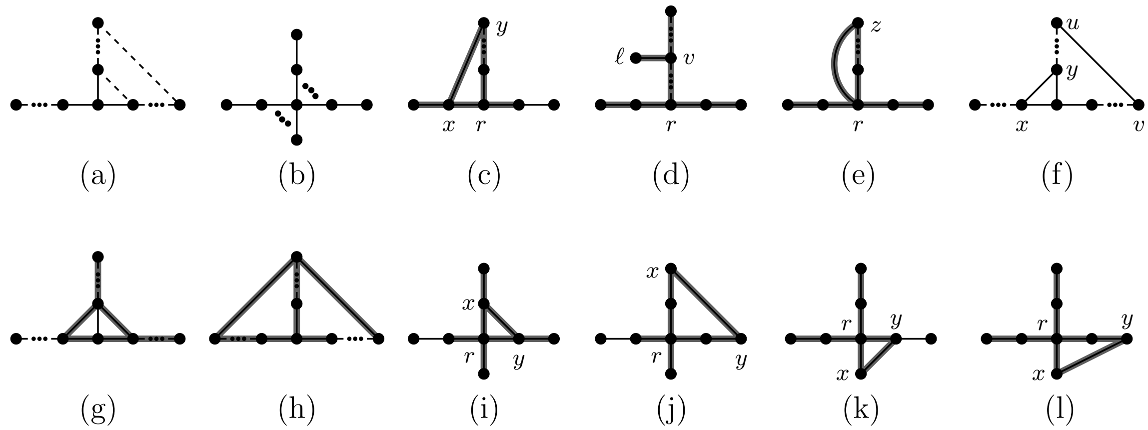


Figure 8.12: Cases for when an ULP graph can contain a subgraph isomorphic to T_7

(ii) a pendant edge (v, ℓ) incident to an internal vertex $v \neq r$ as in Fig. 8.12(d) since G would then have a subgraph homeomorphic to T_8; or

(iii) an edge (r, z) where $z \notin N(r)$ as in Fig. 8.12(e) since G would then have a subgraph homeomorphic to G_δ.

Case 1: Assume that $deg(r) = 3$ in G. We contend that the following two invariants are maintained:

(1) H_i is a degree-3 spider for $i \in [1..m-2]$ and H_{m-1} and H_m are E3Ss as in Fig. 8.12(a); and

(2) e_i is a pendant edge incident to the pendant vertex of H_i for $i < m-1$.

As long as e_i is a pendant edge incident to a pendant vertex of H_i, the invariants are maintained, and H_i is a degree-3 spider for $i < m-1$. Suppose that e_j is the first edge, where this is not the case. Edge types (i), (ii), and (iii) only allow two remaining types of edges for e_j:

(a) the edge (x, y) where $x, y \in N(r)$; and

(b) the edge (u, v) where u and v are pendant vertices of H_j.

In either case, H_{j+1} would be $\mathsf{E3S}$; see Fig. 8.12(f). If there are two edges of type (a) or two edges of type (b) remaining in $G - H_j$ as in Fig. 8.12(g)–(h), then G would have a subgraph homeomorphic to G_6. Hence, there can be at most two edges remaining in $G - H_j$ and have at most one of each type so that invariant (1) is maintained for $i = m - 1$ and $i = m$.

Case 2: Now assume that $deg(r) \geq 4$ in G where e' is some edge incident to r not in H. We contend that the following two invariants are maintained:

(1) H_i is a radius-2 star for $i \in [1..m]$ as in Fig. 8.12(b); and

(2) e_i is a pendant edge incident either to r or to a leaf v of H_i in $N(r)$.

If e_i is an edge such that invariant (2) holds, then invariant (1) is maintained. Suppose that e_j is the first edge, where invariant (2) does not hold. If e_j is a pendant edge incident to a pendant vertex $u \notin N(r)$, then G would contain a subgraph isomorphic to T_9. Edge type (ii) precludes any other type of pendant edge that would violate invariant (2). Edge types (i) and (iii) only allow for four remaining types of internal edges for e_j:

(a) edge (x, y) where $x, y \in N(r)$ and $deg(x) = deg(y) = 3$ as in Fig. 8.12(i);

(b) edge (x, y) where $x, y \notin N(r)$ and $deg(x) = deg(y) = 2$ as in Fig. 8.12(j);

(c) edge (x, y) where $x, y \in N(r)$, $deg(x) = 2$, and $deg(y) = 3$ as in Fig. 8.12(k); and

(d) edge (x, y) where $x \in N(r)$, $y \notin N(r)$, and $deg(x) = deg(y) = 2$ as in Fig. 8.12(l).

However, none of these types of edges are possible. In edge types (a) and (b), $H + e_j$ would either contain a subgraph isomorphic to G_κ and G_α, respectively. In edge types (c) and (d), $H + e_j$ would contain a subgraph homeomorphic to G_δ. Therefore, there can be no such edge e_j that would violate the two invariants. $\square$

We next show that $\mathcal{F}_{\mathsf{ULP}}$ forms a set of forbidden graphs for $\mathcal{G}_{\mathsf{ULP}}$.

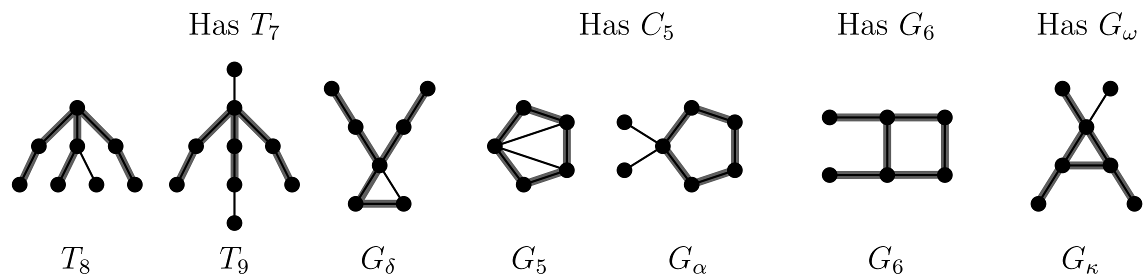


Figure 8.13: Forbidden ULP graphs containing T_7, G_5, G_6, or G_ω subgraphs

Lemma 8.3.11. *The set of ULP graphs with distinct labels $\mathcal{G}_{\mathsf{ULP}}$ does not contain a subgraph homeomorphic to any of the forbidden graphs in $\mathcal{F}_{\mathsf{ULP}}$.*

Proof. If a G is a GC, then according to Lemma 8.3.7, G does not contain a subgraph homeomorphic to (i) T_7, which prevents T_8, T_9, and G_δ from being homeomorphic subgraphs of G; (ii) C_5, which prevents G_5 and G_α from being homeomorphic subgraphs of G; (iii) G_6; or (iv) G_ω, which prevents G_κ from being a homeomorphic subgraph of G; see Fig. 8.13.

If G is a R2S, then by Theorem 5.3.2, G does not contain a subtree homeomorphic to T_8 or T_9. Nor can G contain subgraphs homeomorphic to G_5, G_6, G_α, G_λ, or G_κ, which are all cyclic graphs.

Lastly, if G is either an E3S or an EK4, then by Lemma 8.3.6, every spanning tree of G is either a degree-3 spider or a path, which means that G cannot contain a degree-4 vertex or two degree-3 vertices x and y such that $N(x) \cap N(y) = \varnothing$. Thus, G cannot contain subgraphs homeomorphic to G_5, G_α, G_δ, G_κ, and T_9, which each have a degree-4 vertex. Neither can G contain a subgraph homeomorphic to G_6 or T_8, since both graphs each have two degree-3 vertices with no common neighbors. $\square$

Now that we have assembled the necessary tools, we can prove that the forbidden graphs in $\mathcal{F}_{\mathsf{ULP}}$ fully characterize the ULP graphs $\mathcal{G}_{\mathsf{ULP}}$ with distinct labels.

Theorem 8.3.12. *Every connected graph G is a generalized caterpillar, a radius-2 star, an extended 3-spider, or an extended K_4 subgraph if and only if G does not contain a subgraph homeomorphic to a graph in $\mathcal{F}_{\mathsf{ULP}}$.*

Proof. Lemma 8.3.11 gives necessity. To show sufficiency, assume that G does not contain a subgraph homeomorphic to a graph in $\mathcal{F}_{\mathsf{ULP}}$. By Lemma 8.2.1, the graphs in $\mathcal{F}_{\mathsf{ULP}}$ are minimal examples of graphs that are not GCs, R2S's, E3Ss, or EK4s.

Suppose that G is a connected graph that does not contain a subgraph homeomorphic to a graph in $\mathcal{F}_{\mathsf{ULP}}$. By Lemma 8.3.7, every graph either is a GC or contains a subgraph homeomorphic to G_6, G_ω, C_5, or T_7. Since G does not contain a subgraph homeomorphic to G_6 by assumption, G must then either be a GC or contain a subgraph homeomorphic to G_ω, C_5, or T_7. In the first case, G must be an E3S by Lemma 8.3.8, since G does not contain a subgraph homeomorphic to G_6, G_κ, or T_8. In the second case, G must be either an E3S or an EK4 by Lemma 8.3.9, since G does not contain a subgraph homeomorphic to G_5, G_6, or T_8. In the third case, G must be either an E3S or a R2S by Lemma 8.3.10, since G does not contain a subgraph homeomorphic to G_6, G_α, G_κ, G_δ, T_8, or T_9. $\square$

Corollary 8.1.3 states that any graph containing a subgraph homeomorphic to a forbidden ULP graph in $\mathcal{F}_{\mathsf{ULP}}$ has a labeling that forces a crossing. Theorem 7.3.3 states that each of the four types of ULP graphs in $\mathcal{G}_{\mathsf{ULP}}$ have level planar drawings for any distinct labeling. Theorem 8.3.12 completes the characterization by stating that all graphs either contain a subgraph homeomorphic to a forbidden graph in $\mathcal{F}_{\mathsf{ULP}}$ or belong to one of the four classes of ULP graphs in $\mathcal{G}_{\mathsf{ULP}}$.

Together these results are summarized by our main theorem characterizing ULP graphs with distinct labels.

Theorem 8.3.13. *For graph G, the following three statements are equivalent:*

1. *G does not contain a subgraph homeomorphic to T_8, T_9, G_5, G_6, G_α, G_κ, or G_δ.*

2. *G is a generalized caterpillar, a radius-2 star, an extended 3-spider, or an extended K_4 subgraph.*

3. *G is ULP with distinct labels.*

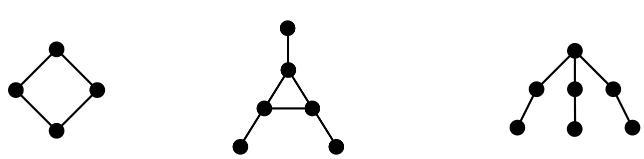
8.4 Characterizing ULP Graphs with Duplicate Labels

By Theorem 7.5.2, K_3-caterpillars and graphs isomorphic to G_ω are both ULP with duplicate labels. In this section, we show that these are the only graphs that do not contain C_4, G_κ, or T_7 as a subdivision. However, before we fully characterize both classes of graphs in $\mathcal{G}^*_{\mathsf{ULP}}$, we characterize the first class of K_3-caterpillars in terms of the three forbidden graphs given in Fig. 8.14.

Lemma 8.4.1. *A connected graph G is a K_3-caterpillar if and only if G does not contain a subgraph homeomorphic to C_4, G_ω, or T_7.*

Proof. To show necessity, suppose that G is a K_3-caterpillar. By Definition 2.3.5, G is only composed of P_2 or K_3 blocks, and hence, G cannot contain a subgraph homeomorphic to C_4. Since every non-trivial block in G is a 3-cycle that is either a 1-block or a 2-block, G cannot contain a subgraph homeomorphic to G_ω that has a 3-block.

Suppose that G has a subgraph H isomorphic to T_7 where the vertex $r \in V(H)$ corresponds to the root of H. Since $deg(r) \geq 3$ in G and the only non-trivial blocks in G are K_3 blocks, r must be a connector in G. By Definition 2.3.4, r only can be the connector of at most two K_3 blocks B_1 and B_2 and the connector of pendant edges. Hence, we can assume w.l.o.g. that two of the length-2 paths p_1 and p_2 in H (with common endpoint r) each have their first edges (r, u) and (r, v) in B_1 where $V(B_1) = \{r, u, v\}$. Since either $deg(u) = 2$ or $deg(v) = 2$ given that B_1 is not a 3-block, then the edge (u, v) must be an edge of p_1 or p_2. Therefore, p_1 and p_2 are not internally disjoint. As a result, G does not have three internally disjoint paths of length-2, and hence, G cannot contain a subgraph homeomorphic to T_7.



C_4 $\quad$ G_ω $\quad$ T_7

Figure 8.14: The three forbidden graphs of K_3-caterpillars

To show sufficiency, next suppose that G does not contain a subgraph homeomorphic to C_4, G_ω, or T_7; see Fig. 8.14. For G to have no k-cycle for some $k \geq 4$ (which would give a subgraph homeomorphic to C_4), each block must be isomorphic to either P_2 or K_3. Additionally, every block in G must either be a 1-block or a 2-block, or G would contain a subgraph homeomorphic to G_ω by Lemma 8.3.3.

Suppose B is a 1-block with connector v. If B is isomorphic to K_3, then B is an ULP K_3 block ending on v. Otherwise, B must be isomorphic to P_2, and hence, is a pendant edge incident to v. Suppose B is a 2-block with connectors v_1 and v_2. Since B is either a P_2 or K_3 block, B is an ULP $(K_3)^m$ block that joins v_1 to v_2 where $m = 0$ (if B is a P_2 block) or $m = 1$ (if B is a K_3 block).

Hence, every block in G is a pendant edge or a K_3 joining or ending block. Since G does not contain a homeomorphic subgraph of T_7, G cannot contain a subgraph isomorphic to T_7. As a consequence, G must be a caterpillar generalized with K_3 blocks by Lemma 8.3.4. Therefore, G is a K_3-caterpillar by Definition 2.3.5. $\square$

We next show that $\mathcal{F}^*_{\mathsf{ULP}}$ forms a set of forbidden graphs for $\mathcal{G}^*_{\mathsf{ULP}}$.

Lemma 8.4.2. *The set of ULP graphs with duplicate labels $\mathcal{G}^*_{\mathsf{ULP}}$ does not contain a subgraph homeomorphic to any of the forbidden graphs in $\mathcal{F}^*_{\mathsf{ULP}}$.*

Proof. If a G is a K_3-caterpillar, then by Lemma 8.4.1, G does not contain a subgraph homeomorphic to either of the graphs T_7 or C_4 in $\mathcal{F}^*_{\mathsf{ULP}}$ or to G_ω, which also prevents G from having a subgraph homeomorphic to G_κ.

If G is isomorphic to G_ω, then G cannot contain a G_κ homeomorphic subgraph (since G_ω is proper subgraph of G_κ). Nor can G contain a subgraphs homeomorphic to T_7 (since $n(G) = 6$ while $n(T_7) = 7$) or C_4 (since G_ω only has a 3-cycle). $\square$

Now that we have the means, we can proceed to prove that the forbidden graphs in $\mathcal{F}^*_{\mathsf{ULP}}$ fully characterize the ULP graphs $\mathcal{G}^*_{\mathsf{ULP}}$ with duplicate labels.

Theorem 8.4.3. *Every connected graph G is a K_3-caterpillar or is isomorphic to G_ω if and only if G does not contain a subgraph homeomorphic to T_7, C_4, or G_κ.*

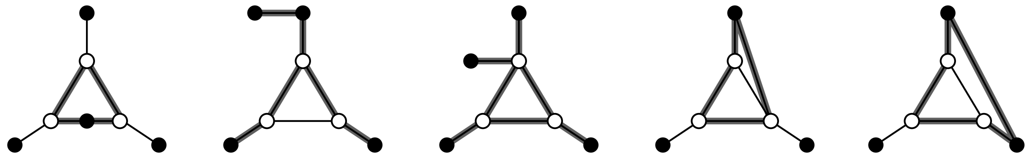


Figure 8.15: Cases for Theorem 8.4.3 for when a graph containing a G_ω subdivision may also contain a C_4, T_7, or G_κ subdivision.

Proof. Lemma 8.4.2 gives necessity. To show sufficiency, assume that G does not contain a subgraph homeomorphic to a graph in $\mathcal{F}^*_{\mathsf{ULP}}$. By Lemma 8.2.2, all the graphs in $\mathcal{F}^*_{\mathsf{ULP}}$ are minimal. By Lemma 8.4.1, every graph either is a K_3-caterpillar or contains a subgraph homeomorphic to C_4, G_ω, or T_7. Since G does not contain a subgraph homeomorphic to C_4 or T_7 by assumption, G must then either be a K_3-caterpillar or contain a subgraph homeomorphic to G_ω.

Suppose G contains a subgraph homeomorphic to G_ω. Either G also contains a subgraph H isomorphic to G_ω (if the 3-cycle is not subdivided) or G would contain a k-cycle for some $k \geq 4$, and hence, a subgraph homeomorphic to C_4; see Fig. 8.15(a). Let e be an edge in $G - H$ that is incident to H. If e is a pendant edge incident to a pendant vertex in H, then $H + e$ contains three internally disjoint length-2 paths that share a common endpoint, and hence, G would contain a subgraph isomorphic to T_7; see Fig. 8.15(b). If e is a pendant edge incident to the 3-cycle of H, then $H+e$ would be isomorphic to G_κ; see Fig. 8.15(c). Finally if e is an internal edge incident to any two vertices of H, then $H + e$ would contain a k-cycle for some $k \in \{4, 5\}$, and hence, G would contain a subgraph homeomorphic to C_4; see Fig. 8.15(d)–(e). Therefore, there can be no edge e in $G - H$, and G must be isomorphic to G_ω. □

Corollary 8.1.4 states that any graph containing a subgraph homeomorphic to a forbidden ULP graph in $\mathcal{F}^*_{\mathsf{ULP}}$ has a labeling that forces a crossing. Theorem 7.5.2 states that both types of ULP graphs in $\mathcal{G}^*_{\mathsf{ULP}}$ have level planar drawings for any duplicate labeling. Theorem 8.4.3 completes the characterization by stating that all graphs either contain a subgraph homeomorphic to a forbidden graph in $\mathcal{F}^*_{\mathsf{ULP}}$ or belong to one of the two classes of ULP graphs in $\mathcal{G}^*_{\mathsf{ULP}}$.

Together these results are summarized by our main theorem characterizing ULP graphs with duplicate labels.

Theorem 8.4.4. *For graph G, the following three statements are equivalent:*

1. *G does not contain a subgraph homeomorphic to T_7, C_4, or G_κ.*
2. *G is a K_3-caterpillar or is isomorphic to G_ω.*
3. *G is ULP with duplicate labels.*

Chapter 9

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 each vertex 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 and graphs:

(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 and certified?

(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 a subdivision of the two forbidden graphs T_8 and T_9. This is akin to Kuratowski's characterization of planar graphs in term of the forbidden subdivisions K_5 and $K_{3,3}$ [68]. Similarly, ULP graphs with distinct labels consist of generalized caterpillars, radius-2 stars, extended 3-spiders, and extended K_4 subgraphs. Every other graph contains a subdivision of one the seven forbidden graphs: T_8, T_9, G_5, G_6, G_α, G_κ, or G_δ.

(2) Each type of ULP tree and graph can be drawn in linear-time and space on an integer grid for any type of 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 and graphs can be recognized by determining in linear-time whether the given tree or graph belongs to one of the classes of ULP trees or graphs. ULP trees can also be certified in linear time by determining if the tree contains a subtree homeomorphic to one of the forbidden graphs.

For ULP trees, Estrella-Balderrama *et al.* provide 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 [39]. 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 while K_3-caterpillars and graphs isomorphic to G_ω are the only graphs that are also ULP when multiple vertices can have the same label. This implies that level caterpillars and K_3-caterpillars are the only trees and graphs that are always level planar, with the lone exception of G_ω.

In the conference paper on ULP trees [37], only the first question was fully addressed for trees, while the second and third questions were only partially addressed, and the fourth question was not considered. In the journal article on ULP trees [40], all four questions were fully addressed for trees. For general planar graphs, only the first two questions were partially addressed in the conference paper on ULP graphs [43]. In this work, we fully addressed all four questions for both ULP trees and graphs in detail, with the one exception: we did not provide a full set of certification algorithms for all ULP graphs that can find a forbidden ULP subdivision (if it exists) in a given planar graph. This last open problem for unlabeled level planarity remains as future work.

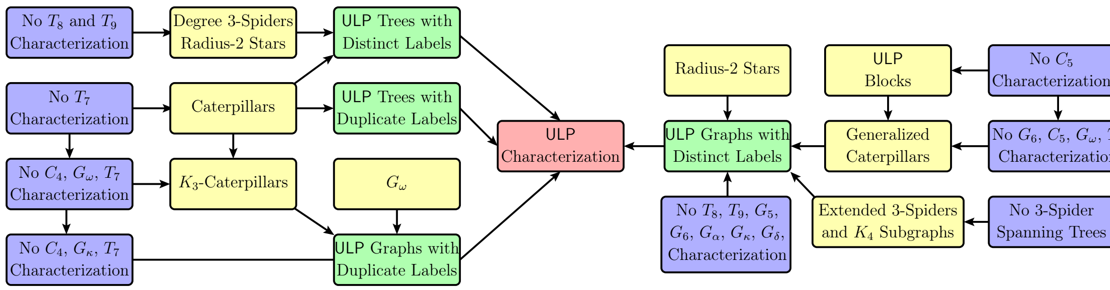


Figure 9.1: An overview of the overall ULP characterization

Figure 9.1 gives an overview of the ULP characterization and each of the other characterizations upon which it rests. In section 5.3, we showed that all trees not containing a T_7 subtree were caterpillars, and all trees that had a caterpillar, but not a T_8 or a T_9 subdivision, were either a radius-2 star or a degree-3 spider giving the Venn diagram of Fig. 1.5. This lead to both of the characterizations for ULP trees with distinct and duplicate labels. In section 8.3, we showed that the three forbidden graphs of C_4, G_ω, and T_7 from Fig. 8.14 form the set of forbidden graphs that fully characterize K_3-caterpillars. Slightly modifying the set of forbidden graphs to include G_κ instead of G_ω allowed us to characterize ULP graphs with duplicate labels, which also includes graphs isomorphic to G_ω.

In section 8.3, we also showed that graphs without a C_5 subdivision must consist of the ULP blocks from Fig. 2.2. We used this to show that generalized caterpillars are fully characterized by the four forbidden graphs of G_6, C_5, G_ω, and T_7 from Fig. 8.8. We also showed that extended 3-spiders and extended K_4 subgraphs only have degree-3 spiders for spanning trees, which is a partial characterization. All these characterizations were combined to give the characterization for ULP graphs with distinct labels illustrated by the Venn diagram in Fig. 1.6.

Future work also includes extending these results for other types of planarity, such as radial level planarity [2, 3] or cyclic level planarity [4, 5]. Both use radial coordinates in lieu of standard Cartesian coordinates. In radial level planarity, the radial distance for each vertex is fixed instead of the y-coordinate as in level planarity where vertices are confined on concentric circles instead of horizontal tracks; see

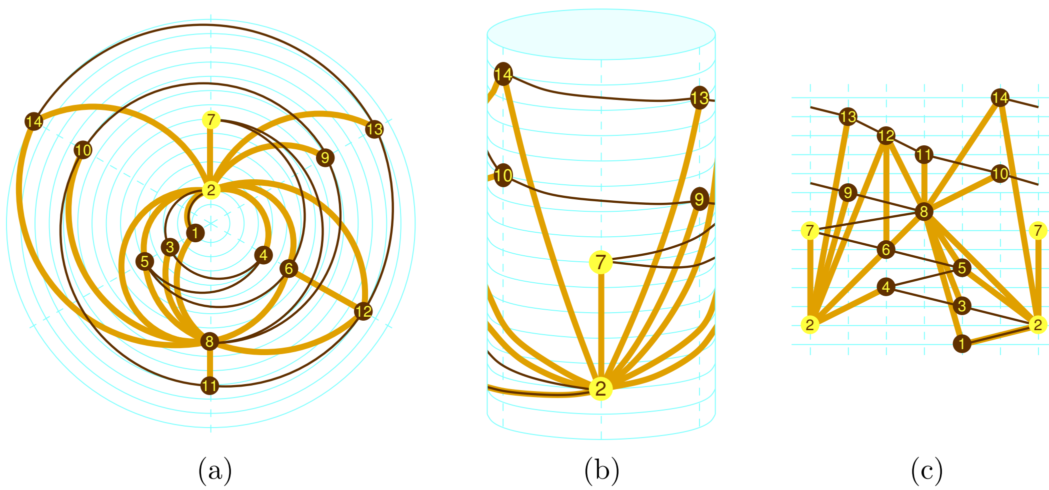


Figure 9.2: Three equivalent representations of radial level planar graphs

Fig. 9.2(a). One can visualize radial level graphs as level graphs wrapped around a cylinder as in Fig. 9.2(b). This is also equivalent to edges "wrapping" around from the right to the left as in Fig. 9.2(c). In cyclic level planarity, the radial angle for each vertex is fixed where vertices are confined to rays emanating from the origin.

Figure 9.3 shows how to construct a *generalized circular caterpillar* from a circular caterpillar, which is a graph in which the removal of all endpoints leaves a cycle. Clearly, generalized circular caterpillars are *unlabeled radial level planar* (URLP) with distinct labels and K_3-circular caterpillars are URLP with duplicate labels. Simply "divide" the graph at any cut-vertex to obtain a generalized caterpillar or K_3-caterpillar, and apply the drawing algorithms from chapter 7 to draw the radial level graph on a cylinder and then "reattach" the ends.

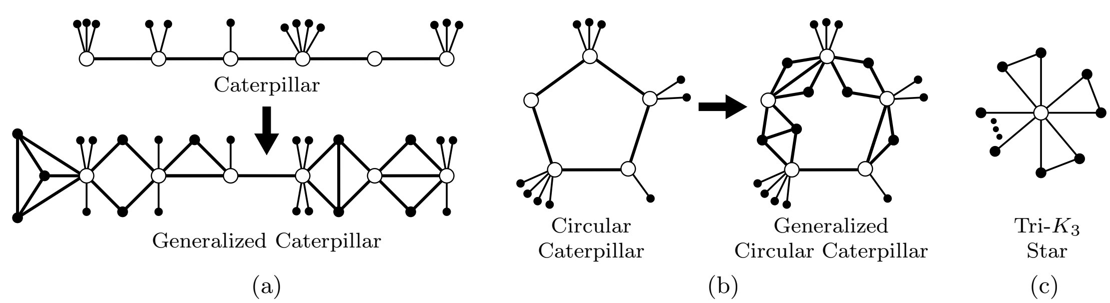
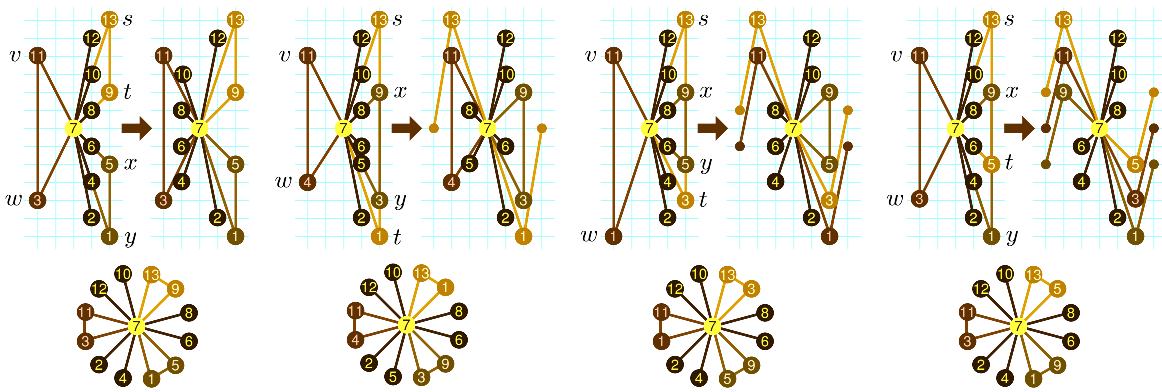


Figure 9.3: As a caterpillar can be generalized using ULP blocks as in (a), so too can a circular caterpillar be generalized as in (b).



(a) (b) (c) (d)

Figure 9.4: Four different cases of drawing a tri-K_3 star on radial levels

The proofs given in section 5.1 for the forbidden labelings of T_7, T_8, and T_9 also apply in the case of radial level planarity, where the ability to "wrap" edges is insufficient to prevent a crossing. Hence, T_7 is one of the forbidden URLP graphs with duplicate labels, and T_8 and T_9 are two of the forbidden URLP graphs with distinct labels.

However, the other forbidden ULP graphs from Fig. 8.1 are all URLP since they are examples of K_3-circular caterpillars (in the case of C_4 and G_ω) and generalized circular caterpillars (in the case of G_5,G_6, G_α, G_κ). The only exception is G_δ, which is a subgraph of a tri-K_3 star shown in Fig. 9.3(c), which consists of three K_3's that share a common vertex that can have any number of pendant edges. Tri-K_3 stars are also URLP as shown by the drawing algorithm outlined in Fig. 9.4.

If one takes any of the forbidden ULP graphs with distinct labels that is URLP, and adds a non-incident edge e as a separate component, then the graph becomes a forbidden URLP graph since the endpoints of e can be given minimum and maximum labels preventing any other edge from "wrapping" around as in Fig. 9.5(a)–(e). For each of the five forbidden cyclic ULP graphs G_5,G_6, G_α, G_δ, G_κ, one can either add an incident edge to prevent an edge from wrapping around as in Fig. 9.5(f)–(j), or add additional internal edge as in Fig. 9.5(k)–(o). Hence, there are at least 17 forbidden URLP graphs as shown in Fig. 9.6.

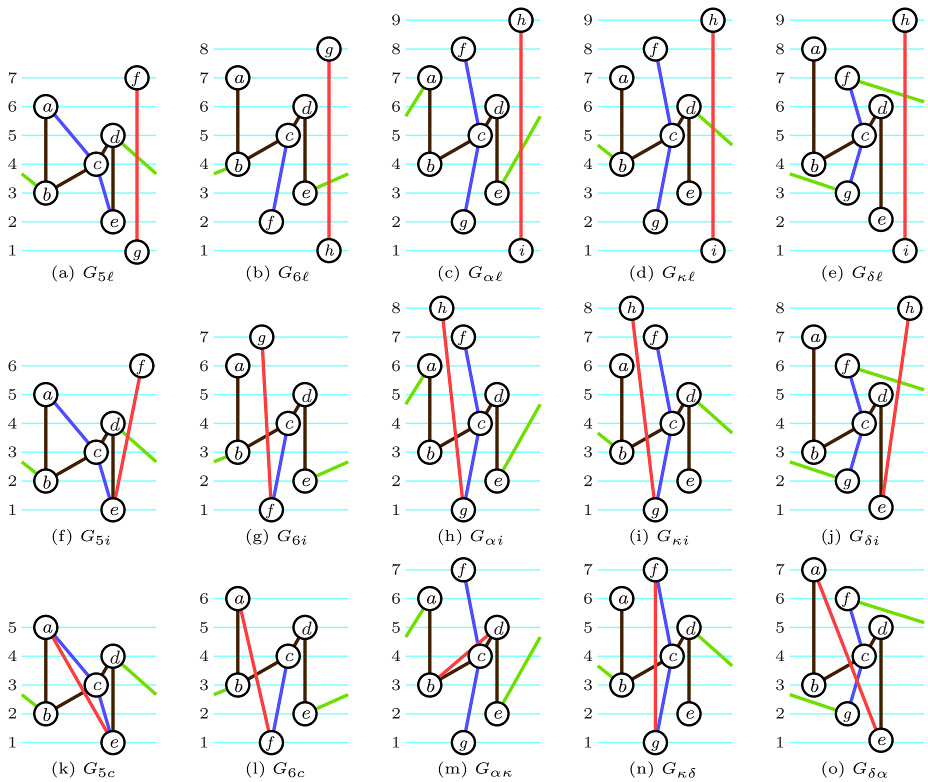


Figure 9.5: Distinct labelings for fifteen forbidden URLP graphs

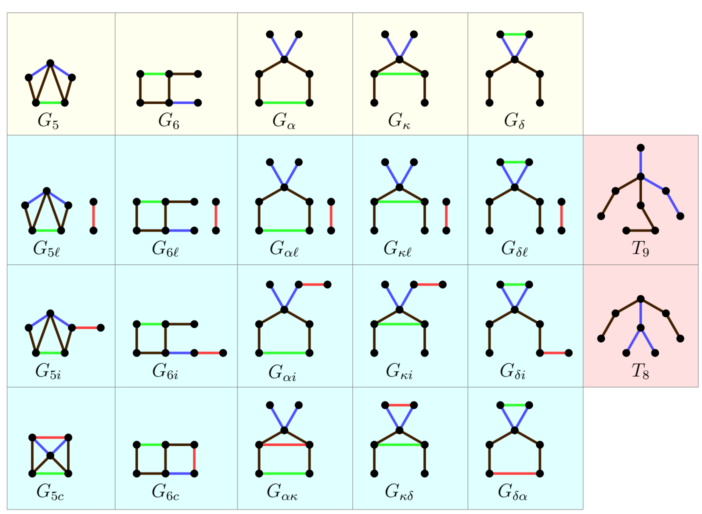


Figure 9.6: Seventeen forbidden URLP graphs

Determining any remaining classes of URLP graphs and providing their drawing algorithms, as well as finding any additional forbidden URLP graphs is the next most logical progression of this work.

References

[1] T. Adachi, H. Kitazawa, M. Nagatani, and T. Sudo. Hierarchical top-down layout design method for VLSI chip. In *19th Conference on Design Automation, (DAC 1982)*, pages 785–791, Piscataway, NJ, 1982. IEEE Press.

[2] C. Bachmaier. *Circle Planarity of Level Graphs*. PhD thesis, Fakultät für Mathematik und Informatik, Universität Passau, 2004.

[3] C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time. *Journal of Graph Algorithms and Applications*, 9(1):53–97, 2005.

[4] C. Bachmaier and W. Brunner. Linear time planarity testing and embedding of strongly connected cyclic level graphs. In D. Halperin and K. Mehlhorn, editors, *16th European Symposium on Algorithms (ESA 2008)*, LNCS, pages 136–147, Heidelberg, 2008. Springer.

[5] C. Bachmaier, W. Brunner, and C. König. Cyclic level planarity testing and embedding. In S. Hong and T. Nishizeki, editors, *15th Symposium on Graph Drawing (GD 2007)*, volume 4875 of *LNCS*, pages 50–61, Heidelberg, 2008. Springer.

[6] D. Batory and S. O'Malley. The design and implementation of hierarchical software systems with reusable components. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 1(4):355–398, 1992.

[7] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, Reading, Massachusetts, 1999.

[8] K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.

[9] U. Brandes. Drawing on physical analogies. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs*, volume 2025 of *LNCS*, pages 71–86. Springer, Heidelberg, 2001.

[10] 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.

[11] U. Brandes, J. Raab, and D. Wagner. Exploratory network visualization: Simultaneous display of actor status and connections, 2001.

[12] 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. In *8th Workshop on Algorithms and Data Structures (WADS 2003)*, pages 243–255, 2003.

[13] 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.

[14] R. Brockenauer and S. Cornelsen. Drawing clusters and hierarchies. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs: Methods and Models*, number 2025, pages 193–227. Springer, Heidelberg, 2001.

[15] C. Buchheim, M. Chimani, D. Ebner, C. Gutwenger, M. Jünger, G. W. Klau, P. Mutzel, and R. Weiskircher. A branch-and-cut approach to the crossing number problem. 5(2):373–388, 2008.

[16] C. Buchheim, D. Ebner, M. Jünger, G. W. Klau, P. Mutzel, and R. Weiskircher. Exact crossing minimization. In P. Healy and N. S. Nikolov, editors, *13th International Symposium on Graph Drawing (GD 2005)*, volume 3843 of *LNCS*, pages 37–48, Heidelberg, 2006. Springer.

[17] M. J. Carpano. Automatic display of hierarchized graphs for computer-aided decision analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(11):705–715, 1980.

[18] M. Chimani, C. Gutwenger, and P. Mutzel. Experiments on exact crossing minimization using column generation. In C. Àlvarez and M. J. Serna, editors, *5th International Workshop on Experimental Algorithms (WEA 2006)*, volume 4007 of *LNCS*, pages 303–315, Heidelberg, 2006. Springer.

[19] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong. Layer-free upward crossing minimization. In *7th International Workshop on Experimental Algorithms (WEA 2008)*, volume 5038 of *LNCS*, pages 55–68, Heidelberg, 2008. Springer.

[20] M. Chimani, P. Mutzel, and I. M. Bomze. A new approach to exact crossing minimization. In D. Halperin and K. Mehlhorn, editors, *16th Annual European Symposium on Algorithms (ESA 2008)*, volume 5193 of *LNCS*, pages 284–296, Heidelberg, 2008. Springer.

[21] R. S. Corporation. Rational rose, 2002.

[22] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry: Theory and Applications*, 4:235–282, 1994.

[23] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Englewood Cliffs, NJ, 1999.

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

[25] G. Di Battista, E. Pietrosanti, R. Tamassia, and I. G. Tollis. Automatic layout of PERT diagrams with XPERT. In *IEEE Workshop on Visual Languages (VL 1989)*, pages 171–176, 1989.

[26] P. Dietz and D. Leigh. DiamondTouch: a multi-user touch technology. In *14th ACM Symposium on User Interface Software and Technology*, pages 219–226, 2001.

[27] 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.

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

[29] P. Eades, Q. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In *4th Symposium on Graph Drawing (GD 1996)*, volume 1190 of *LNCS*, pages 113–128, Heidelberg, 1997. Springer.

[30] 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.

[31] P. Eades, W. Lai, K. Misue, and K. Sugiyama. Preserving the mental map of a diagram. In *Proceedings of Compugraphics 91*, pages 24–33, 1991.

[32] P. Eades and K. Sugiyama. How to draw a directed graph. *Journal on Information Processing*, 13:424–437, 1991.

[33] P. Eades and L. Xuemin. How to draw a directed graph. In *IEEE Workshop on Visual Languages (VL 1989)*, pages 13–17, 1989.

[34] H. Eichelberger. Sugibib. In *9th International Symposium on Graph Drawing (GD 2001)*, volume 2265 of *LNCS*, pages 581–584, Heidelberg, 2002. Springer.

[35] D. Eppstein. Separating thickness from geometric thickness. In M. Goodrich and S. Kobourov, editors, *10th Symposium on Graph Drawing (GD 2002)*, volume 2528 of *LNCS*, pages 150–161, Heidelberg, 2003. Springer.

[36] C. Erten and S. G. Kobourov. Simultaneous embedding of planar graphs with few bends. In J. Pach, editor, *12th Symposium on Graph Drawing (GD 2003)*, volume 3383 of *LNCS*, pages 195–205, Heidelberg, 2004. Springer.

[37] A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. Characterization of unlabeled level planar trees. In M. Kaufman and D. Wagner, editors, *14th Symposium on Graph Drawing (GD 2006)*, volume 4372 of *LNCS*, pages 367–379, Heidelberg, 2007. Springer.

[38] A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. Simultaneous graph embedding with bends and circular arcs. In M. Kaufman and D. Wagner, editors, *14th Symposium on Graph Drawing (GD 2006)*, volume 4372 of *LNCS*, pages 367–379, Heidelberg, 2007. Springer.

[39] A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. Graph simultaneous embedding tool, graphset. In I. Tollis and M. Patrignani, editors, *16th Symposium on Graph Drawing (GD 2008)*, volume 5417 of *LNCS*, pages 169–180, Heidelberg, 2008. Springer.

[40] A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. Characterization of unlabeled level planar trees. *CGTA: Computational Geometry: Theory and Applications*, 42(7):704–721, 2009.

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

[42] I. Fáry. On straight line representation of planar graphs. *Acta Univ. Szeged. Sect. Sci. Math.*, 11:229–233, 1948.

[43] J. J. Fowler and S. G. Kobourov. Characterization of unlabeled planar graphs. In S. H. Hong and T. Nishizeki, editors, *15th Symposium on Graph Drawing (GD 2007)*, volume 4875 of *LNCS*, pages 37–49, Heidelberg, 2008. Springer.

[44] J. J. Fowler and S. G. Kobourov. Minimum level nonplanar patterns for trees. In S. H. Hong and T. Nishizeki, editors, *15th Symposium on Graph Drawing (GD 2007)*, volume 4875 of *LNCS*, pages 69–75, Heidelberg, 2008. Springer.

[45] F. Frati. Embedding graphs simultaneously with fixed edges. In M. Kaufman and D. Wagner, editors, *14th Symposium on Graph Drawing (GD 2006)*, volume 4372 of *LNCS*, pages 108–113, Heidelberg, 2007. Springer.

[46] M. Geyer, M. Kaufmann, and I. Vrt´o. Two trees which are self-intersecting when drawn simultaneously. In P. Healy and N. S. Nikolov, editors, *13th Symposium on Graph Drawing (GD 2005)*, volume 3843 of *LNCS*, pages 201–210, Heidelberg, 2006. Springer.

[47] C. Gutwenger, M. Jünger, K. Klein, J. Kupke, S. Leipert, and P. Mutzel. A new approach for visualizing UML class diagrams. In S. Diehl, J. T. Stasko, and S. N. Spencer, editors, *2003 ACM Symposium on Software Visualization (SoftVis 2003)*, pages 179–188, New York, NY, USA, 2003. ACM.

[48] C. Gutwenger and P. Mutzel. An experimental study of crossing minimization heuristics. In G. Liotta, editor, *11th International Symposium on Graph Drawing (GD 2003)*, volume 2912 of *LNCS*, pages 13–24, Heidelberg, 2004. Springer.

[49] M. Harrigan and P. Healy. Practical level planarity testing and layout with embedding constraints. In S. H. Hong and T. Nishizeki, editors, *15th Symposium on Graph Drawing (GD 2007)*, volume 4875 of *LNCS*, pages 62–68, Heidelberg, 2008. Springer.

[50] 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.

[51] P. Healy, A. Kuusik, and S. Leipert. A characterization of level planar graphs. *Discrete Mathematics*, 280(1-3):51–63, 2004.

[52] 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.

[53] 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.

[54] L. S. Heath and A. L. Rosenberg. Laying out graphs using queues. *SIAM Journal on Computing*, 21(5):927–958, 1992.

[55] P. Holme, M. Huss, and H. Jeong. Subnetwork hierarchies of biochemical pathways. *Bioinformatics*, 19(4):532–538, 2003.

[56] N. Jardine and R. Sibson. *Mathematical taxonomy*. John Wiley & Sons Ltd., London, 1971. Wiley Series in Probability and Mathematical Statistics.

[57] M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In G. D. Battista, editor, *5th International Symposium on Graph Drawing (GD 1997)*, volume 1353 of *LNCS*, pages 13–24, Heidelberg, 1998. Springer.

[58] M. Jünger and S. Leipert. Level planar embedding in linear time. In J. Kratochvíl, editor, *7th Symposium on Graph Drawing (GD 1999)*, volume 1731 of *LNCS*, pages 72–81, Heidelberg, 2000. Springer.

[59] M. Jünger and S. Leipert. Level planar embedding in linear time. *Journal of Graph Algorithms and Applications*, 6(1):67–113, 2002.

[60] M. Jünger, S. Leipert, and P. Mutzel. Pitfalls of using PQ-trees in automatic graph drawing. In S. C. North, editor, *4th Symposium on Graph Drawing (GD 1996)*, volume 1190 of *LNCS*, pages 193–204, Heidelberg, 1997. Springer.

[61] M. Jünger, S. Leipert, and P. Mutzel. Level planarity testing in linear time. In *6th Symposium on Graph Drawing (GD 1998)*, volume 1547 of *LNCS*, pages 224–237, Heidelberg, 1999. Springer.

[62] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1, 1997.

[63] A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

[64] A. B. Kahng, I. Mandoiu, P. Pevzner, S. Reda, and A. Zelikovsky. Engineering a scalable placement heuristic for dna probe arrays. In *7th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2003)*, pages 148–156, New York, NY, USA, 2003. ACM.

[65] A. B. Kahng, I. Mandoiu, S. Reda, X. Xu, and A. Z. Zelikovsky. Evaluation of placement techniques for dna probe array layout. In *2003 IEEE/ACM international Conference on Computer-Aided Design (ICCAD 2003)*, page 262, Washington, DC, 2003. IEEE Computer Society.

[66] G. Kant. *Algorithms for Drawing Planar Graphs.* PhD thesis, Dept. Comput. Sci., Univ. Utrecht, Utrecht, Netherlands, 1993.

[67] S. G. Kobourov and C. Pitta. An interactive multi-user system for simultaneous graph drawing. In J. Pach, editor, *12th Symposium on Graph Drawing (GD 2003)*, volume 3383 of *LNCS*, pages 492–501, Heidelberg. Springer. 2004.

[68] C. Kuratowski. Sur les problèmes des courbes gauches en Topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

[69] S. Leipert. *Level Planarity Testing and Embedding in Linear Time*. PhD thesis, Mathematisch-Naturwissenschaftlichen Fakultät der Universität zu Köln, 1998.

[70] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Visual Lang. Comput.*, 6(2):183–210, 1995.

[71] P. Mutzel. An alternative method to crossing minimization on hierarchical graphs. In S. C. North, editor, *4th International Symposium on Graph Drawing on Graph Drawing (GD 1996)*, volume 1190 of *LNCS*, pages 318–333, Heidelberg, 1997. Springer.

[72] P. Mutzel and R. Weiskircher. Two-layer planarization in graph drawing. In K. Chwa and O. Ibarra, editors, *9th International Symposium on Algorithms and Computation (ISAAC 1998)*, volume 1533 of *LNCS*, pages 69–78, Heidelberg, 1998. Springer.

[73] J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. In S. C. North, editor, *4th Symposium on Graph Drawing (GD 1996)*, LNCS, pages 263–274, Heidelberg, 1998. Springer.

[74] H. Purchase. Which aesthetic has the greatest effect on human understanding? In G. Di Battista, editor, *5th Symposium on Graph Drawing (GD 1997)*, volume 1353 of *LNCS*, pages 248–261, Heidelberg, 1998. Springer.

[75] H. C. Purchase, R. F. Cohen, and M. James. Validating graph drawing aesthetics. In F. J. Brandenburg, editor, *3rd Symposium on Graph Drawing (GD 1995)*, volume 1027 of *LNCS*, pages 435–446, Heidelberg, 1996. Springer.

[76] M. G. Reggiani and F. E. Marchetti. A proposed method for representing hierarchies. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):2–8, 1988.

[77] L. A. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis, and A. Tuan. A browser for directed graphs. Technical Report UCB/ERL M87/5, EECS Department, University of California, Berkeley, 1987.

[78] J. Seemann. Extending the Sugiyama algorithm for drawing UML class diagrams: Towards automatic layout of object-oriented software diagrams. In G. Di Battista, editor, *5th Symposium on Graph Drawing (GD 1997)*, volume 1353 of *LNCS*, pages 415–424.

[79] K. Sugiyama. *Graph Drawing and Applications for Software and Knowledge Engineers*, volume 11 of *Series on Software Engineering and Knowledge Engineering*. World Scientific Publishing Company, June 2002.

[80] 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.

[81] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[82] P. F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. *ACM SIGCOMM Computer Communication Review*, 18(4):35–42, 1988.