Eberhard Karls Universität Tübingen
Wilhelm Schickard Institut Tübingen

Fachbereich Informatik

On Maximizing the Euclidian Distance Between Adjacent Vertices in Drawings of Small Area

Arbeitsbereich Algorithmik

zur Erlangung des akademischen Grades
Master of Science

Autor:	Benjamin Çoban MatNr. 3526251
Version vom:	18. Juli, 2022
ErstprüferIn:	Prof. Dr. Michael Kaufmann
ZweitprüferIn:	Prof. Dr. Ulrike von Luxburg

Zusammenfassung

Graphenzeichnungen sind vielfach anwendbar - für die Analyse der Molekularstrukturen oder Sensornetzen sind Zeichnungen mit vorbestimmten Distanzen von besonderer Bedeutung. Das *Graph Drawing Symposium* fordert im Jahre 2022 die Teilnehmer des dort stattfindenden Wettbewerbs auf, Zeichnungen auf kleiner Fläche anzufertigen und dabei die Distanzen der paarweise verbundenen Knoten zu maximieren. In *geradlinigen Zeichnungen* werden Knoten mit einem einzigen Liniensegment verbunden. Um die jeweiligen Distanzen zwischen Knoten zu maximieren, ist eine Umpositionierung der Knoten erforderlich. Dies erweist sich bei geradlinigen Zeichnungen als bedingt möglich. Lässt man *Polygonzüge* zu, erweitert dies die Möglichkeiten der Knotenrepositionierung. Bei einer *Polygonzugzeichnung* wird eine Verbindung zwischen zwei Knoten als eine Sequenz von Liniensegmenten mit unterschiedlicher Steigung, welche sich in einem Punkt schneiden, dargestellt. Dieser Schnittpunkt wird im Allgemeinen *Knick* genannt. Die Optimierungsfunktion wird als das Verhältnis zwischen der kürzesten euklidischen Distanz zwischen zwei verbundenen Knoten und dem gesamten Flächenverbrauch beschrieben. Dieses Verhältnis wird die *Ratio* genannt.

Anlässlich der diesjährigen Symposium Challenge untersucht dieses Ausarbeitung die Maximierung der euklidischen Distanz für die Klasse der *verwurzelten Bäume* und der *maximalen seriell-parallelen Graphen*. Im Allgemeinen ist die Klasse der Bäume durch ihre überschaubaren Eigenschaften eine gute erste Anlaufstelle zur Untersuchung eines Optimierungsproblems. Da die Klasse der maximal seriell-parallelen Graphen um einiges vielseitiger in ihren Eigenschaften ist, ist die Optimierung von dessen Zeichnungen mit geringem Platzverbrauch von hohem Interesse.

Zuerst wird ein Zeichenalgorithmus für verwurzelte Bäume präsentiert, welcher geradlinige Zeichnungen mit einer optimalen Ratio erstellt.

Darauffolgend wird eine Subklasse der maximal seriell-parallelen Graphen, die sogenannten *maximal outerplanaren* Graphen untersucht. Es wird gezeigt, dass die Ratio *unbeschränkt* ist, was heißt, dass es für jeden maximal outerplanaren Graphen einen größeren gibt, der das Ratio der dementsprechenden Zeichnung signifikant erhöht.
Ein erster Ansatz für eine Polygonzugzeichnung setzt sich mit maximal outerplanaren Graphen hoher Dichte auseinander. Die Begrenzungen dieses Ansatzes dienen zur Inspiration für einen zweiten, allgemeineren Ansatz. Dieser zweite Ansatz dient als eine Grundlage für die Erstellung von Polygonzugzeichnungen von maximal seriell-parallelen Graphen. Die Ansatzerweiterung resultiert in einem Zeichenalgorithmus, welcher Polygonzugzeichnungen von maximalen seriell-parallelen Graphen auf vertretbar kleiner Fläche produziert, sodass die Ratio in einem angemessenen Rahmen bleibt.

Zusätzlich wurde eben dieser Ansatz auf eine restriktivere Klasse an kreuzungsfreien Graphen, den sogenannten *planaren 3-Trees*, angewandt. Es wird illustriert, dass dieser Ansatz bezüglich der Ratiooptimierung für die Klasse der 3-Trees unzureichend ist.

Abstract

Graph drawings are diverse in their applications - regarding molecular structure analysis or sensor networks, constructing a drawing with certain properties of distance is of interest. In the year 2022, the *Graph Drawing Symposium* challenges its participants to create small area drawing with maximizing the distances between two adjacent vertices. In a *straight-line drawing*, adjacent vertices are connected by a single line segment. In order to increase the distance between vertices, the vertices are repositioned in a fixed area. In straight-line drawings, the repositioning might be limited in its possibilities. Allowing *polylines* loosens this limitation. In a *polyline drawing*, every edge is illustrated as a sequence of line segments which intersect in a point, so-called *bends*. The optimization problem is described as the proportion between the length of the Euclidian distance to the total area consumption. This will be called the *ratio* of a drawing.
On occasion of this Symposium challenge, this thesis will examine the maximization of the *Euclidian distance* on *rooted trees* and small area drawings of the class of *maximal series-parallel graphs*. For a given drawing problem, the class of trees are rather manageable due to their straightforward properties. The graph class of maximal series-parallel graphs is more complex in its properties than rooted trees and investigating the distance maximization of those small area drawings is of particular interest.

First, a drawing algorithm for rooted trees is presented which produces straight-line drawings with uniform distances between adjacent vertices.

Next, the *maximal outerplanar graphs*, a subclass of maximal series-parallel graphs, are investigated for Euclidian distance maximization on small area polyline drawings. It will be proved that they are unbounded, meaning that for every maximal outerplanar graph there exists a larger graph which exceeds the ratio in the respective drawing.
A first approach of a polyline drawing algorithm deals with maximal outerplanar graphs of high density. Its limitations will insprire a new idea for dealing with more general occasions. Then, a second approach will serve as a foundation to be extended for the class of maximal series-parallel graphs. An extension of this approach will produce small area polyline drawings for maximal series-parallel graphs with a reasonable ratio.

In the extensional work, the previous approaches and results will be applied to a more restrictive class of graphs, called the *planar 3-trees*. It will be demonstrated that the approach for maximal series-parallel graphs is not applicable to the class of 3-trees regarding ratio optimization.

Erklärung

Hiermit erkläre ich, dass ich diese schriftliche Abschlussarbeit selbstständig verfasst habe, keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe.

Datum, Ort, Unterschrift

Contents

1 Introduction

The topic of visualization of information relationships occur in various areas of work. Examples of the fields include circuit design, architecture, web science, social sciences, biology, geography, information security and software engineering. Over the last decades, many different efficient algorithms were developed for graph drawings in the Euclidean plane. Different quality measures for graph drawings have been considered, including area consumption, angular resolution, slope number, average edge length, and total edge length, addressing the readability and aesthetics [6].
Graphs admitting a *crossing-free* drawing are called *planar* graphs and have a long history in graph theory. This well-developed theory can be used to simplify topological concepts that otherwise would prove cumbersome [3, P. 9]. In

context of this thesis, drawings with prescribed edge lengths is of particular interest. Its relevance addresses aspects of computational geometry, rigidity theory, structural analysis of molecules and sensor networks [1, P. 1]. In general, it is hard to decide [1, P. 2] whether a given graph admits a drawing with prescribed edge lengths under the condition that any edge is drawn with a straight-line and no two edges cross each other, even in the case of uniform lengths for all edges.

Starting from a workshop in 1994, the first international conference for *Graph Drawing* was held in Passau in 1995 [16]. The annual symposium covers topics of combinatorical and algorithmic aspects of graph drawing as well as the design of network visualization systems and interfaces.
One part of the symposium is the *Graph Drawing Contest*. The contest consists of two parts - the *Creative Topics* and the *Live Challenge*. The main focus for the Creative Topics lies on the creation of drawings of two given graphs. Aspects to consider for the visualization are clarity, aesthetic appeal and readability.
On the other hand, the Live Challenge is held similar to a programming contest. Participants, usually teams, will get a task and will have one hour of processing a set of exemplary graphs. The results will be ranked and the team with highest score wins the competition. The teams will be allowed to use any combination of software and human interaction systems in order to produce the best results. Usually, the challenge is derived from a theoretical optimization problem [10].

In 2021, the Live Challenge during the 29th International Symposium on Graph Drawing and Network Visualization held in Tübingen, Germany addressed the optimization of graph drawing edge lengths. An *edge length ratio* of a drawing describes the proportion between the *minimal and maximal edge lengths*. The size of the total area of a drawing affects the maximum edge length. When considering *straight-line drawings*, where edges are a single straight line segment, the ratio scales in proportion of the total area size.
For the *Live Challenge*, the goal was to produce a *polyline graph drawing*, where edges are line segments joined together, with uniform edge lengths. The difficulty of this challenge was intensified by constraints on the drawing area and the amount of line segments per edge [11].

In 2022, the 30th International Symposium of Graph Drawing and Network Visualization held in Tokio, Japan [12] addresses an alternation of the Live Challenge from previous year. In contrast to the edge length ratio from 2021, this years ratio describes the proportion of the *maximal polyline edge length* to the *minimal Euclidian distance* between two adjacent vertices.

This thesis will address the topic of minimizing the difference between the longest and shortest edge length for certain graph classes. The main goal is to maximize the distances between two adjacent vertices while keeping the area consumption of the resulting drawing at a low level. Allowing to reroute edges through so-called *bend points* in a drawing without causing crossings between edges increases the flexibility of finding an approach for the edge length difference minimization approach.
In Section 2, the preliminaries and terminology are defined. In Section 3, the general problem considering the edge length ratio is formalized. Furthermore, the potential for ratio improvement is illustrated by allowing polyline edges. Section 4 describes a drawing algorithm for the graph class of *k-ary trees* which guarantees a satisfying edge length ratio. Section 5 examines the properties of *maximal outerplanar graphs*, a subclass of *maximal SP-graphs*, and demonstrates two drawing algorithms. Section 6 contains an extension of a drawing algorithm for maximal outerplanar graphs in order to draw any *maximal SP-graph* with a reasonable ratio. Then, it addresses the applicability of the main approach for other classes of planar graphs. It will be demonstrated that the approach for maximal SP-graphs does not suffice for a more restriced class of planar graphs, the planar *3-trees.*

2 Preliminaries

2.1 Definitions and Terminology

A *graph* $G = (V, E)$ is a tuple consisting of two sets - the set of vertices $V = V(G)$ and the set of edges $E = E(G)$. An *edge* $e = (v, w), v, w \in V$ is a tuple and describes a connectivity relation between two vertices. If $V' \subseteq V, E' \subseteq E$, then $G' = (V', E')$ is a *subgraph* of G. The *degree* of a vertex states the amount of edges incident to the vertex.
A *path* of length k from a vertex v_1 to v_{k+1} is a sequence of vertices $(v_1, ..., v_{k+1})$ such that (v_i, v_{i+1}) is an edge in G. A path is *simple* if all the vertices in the sequence are distinct. A *cycle* is a path where $v_1 = v_{k+1}$ and has at least one edge. A graph with no cycles is called *acyclic* [7, P. 1170].
Unless otherwise mentioned, the graphs are *undirected*, meaning that the edge (u, v) is identical to the edge (v, u). An undirected graph is *connected* if every vertex is reachable from all the other vertices [7, P. 1170]. A graph is *biconnected* if the removal of any vertex still leaves the graph connected [9, P. 224]. A graph is *simple* if it does contain neighter multiple edges nor self loops. On the other hand, if a graph contains multiple edges or self loops, it is called a *multigraph* [7, P. 1172]. Unless otherwise mentioned, a graph is presumed to be simple.
The *depth-first search, DFS* in short, is a strategy to explore edges out of the most recently discovered vertex v that still has unexplored edges leaving it. Once of all v's edges have been explored, the DFS backtracks in order to explore edges leaving the vertex from which v was discovered. The DFS terminates when all the reachable vertices from the original source vertex were discovered [7, P.603].

2.2 Graph Drawing Models and Representations

An $h \times w$ *grid* is a graph consisting of h rows and w columns of vertices. The vertex in the i-th row and j-th column is denoted as (i, j) and is called a *grid point*. All vertices in a grid have exactly four neighbours, except for the boundary vertices [7, P. 760]. One *unit length* values the distance between two adjacent vertices on the grid and is denoted as `UL` . A *drawing* Γ of a graph G is a function, where each vertex is mapped on a unique point $\Gamma(v)$ in the plane and each edge is mapped on an open Jordan curve $\Gamma(e)$ ending in its vertices [9, P. 225]. In this context, a graph will be drawn on an underlying grid.
A drawing is called *planar* if no two distinct edges intersect. A graph is *planar* if it admits a planar drawing[7, Page 100]. A drawing partitions the plane into topologically connected regions [3, P. 7]. A *face* is a maximal open region of the plane bounded by edges. The *outer face* is the unbounded face. A bounded face is called *inner face* [8, S. 86].
A multigraph G^* is the *dual graph of G* if and only if there exists a bijective function between G^* and G such that:

1. Every face f in G corresponds to a vertex v_f in G^*
2. For every edge e of G, the corresponding vertices of the faces in G^* incident to e get an edge

3. If e is incident to only one face, a loop is attached to the corresponding vertex in G^*

[8, P. 103] The *weak dual graph* of G is the dual graph of G without considering the outer face.

An *embedding* of G is the collection of counter-clockwise circular orderings of edges around each vertex of V, denoted as a sequence of edges.
In a *straight-line drawing*, verticees are points on the grid and edges are straight-line segments.
In a *polyline drawing*, vertices are points on the grid, edges are sequences of contiguous straight-line segments. The transition point between two edge segments with different slopes is called a *bend*. Like vertices, bends are placed on points on the underlying grid.
A *box* is an axis-parallel rectangle, overlapping vertical and horizontal grid lines. The *width* of a box is one unit smaller than the number of vertical grid lines that are overlapped by it. The *height* of a box is one unit smaller than the number of horizontal grid lines that are overlapped by it. In an *orthogonal box drawing*, vertices are axis-aligned boxes (possibly degenerated to a line segment or a point), edges are sequences of contiguous horizontal or vertical line segments. [5, P. 144ff]
A *layering* is a mapping $L : V \to \mathbb{N}$ and determines the horizontal grid line placement of a vertex. A layering is *valid*, if $|L(u) - L(v)| \geq 1$ for any edge (u, v) [15, P. 4].
A drawing whose minimum enclosing box has width w and height h is called a $w \times h$-drawing and inherits area $w \cdot h$ [5, P. 145].
The <u>*euclidian distance*</u> between two grid points (x_1, y_1) and (x_2, y_2) is defined as $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. The *length of a line segment* is defined as the euclidian distance between the end points. The *length of a polyline* is defined as the sum of the individual line segment lengths.

2.3 The Relationship between Drawing Models

Box drawings, straight-line and polyline drawings are the drawings of interest in this thesis. They stand in relation to each other in the following way:
A straight-line drawing is a polyline drawing with no bends by definition. Every sequence of line segments is of length 1.
A box drawing can be transferred to a polyline drawing with asymptotically the same area consumption. For this to happen, add empty grid lines until every segment of every edge has length at least 2. This will at most double the width and height. For any vertex, replace the box by an arbitrary grid point inside the box and reroute the incident edges to that vertex locally. Every box introduces a bend per edge, resulting in a polyline drawing with two bends and the asymptotically same area bound. These relationships are illustrated in figure 1. [5, P. 145]

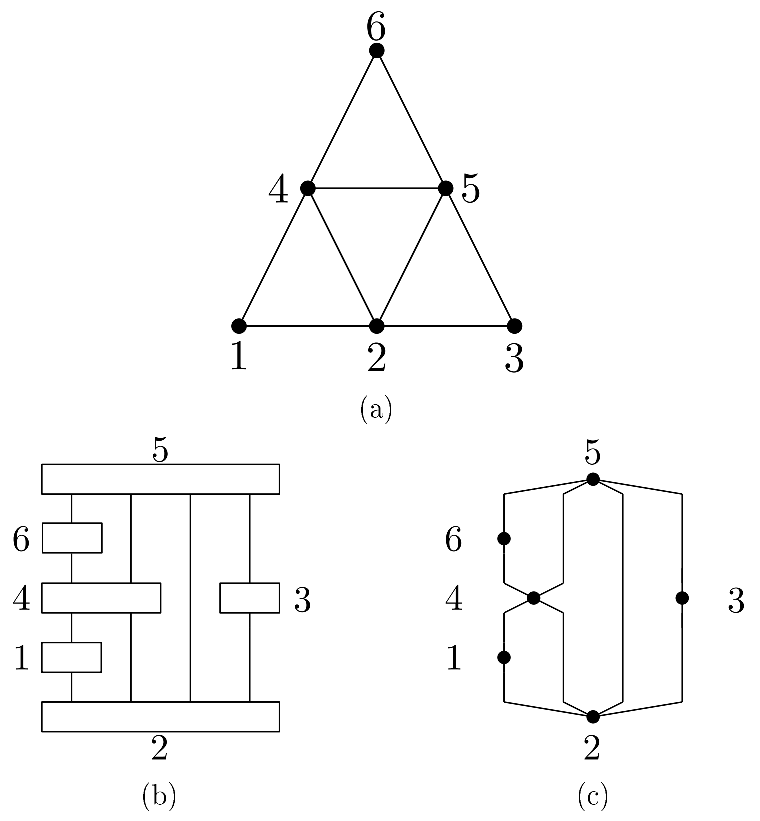


Figure 1: 1a is a straight-line drawing of a maximal outerplanar graph G, 1b is a box drawing of G and 1c is a polyline drawing derived from 1b

2.4 Graph Classes

2.4.1 Tree

A graph T is called *tree* if and only if it is connected, acyclic and undirected [7, P. 1172].
A tree is *rooted* if one of its vertices is distinguished from the other ones, called the *root*. Unless otherwise mentioned, it is assumed that every tree is rooted at the root vertex r.

When considering a path $(r, v_1, ..., v_k)$ from r to any other vertex v_k, then, the following holds:

1. Besides w, every vertex of this path is an *ancestor* of w
2. For an edge (v_i, v_{i+1}), v_i is called the *parent* of v_{i+1}, and v_{i+1} is a child of v_i.
3. The *height* of the root r is set to 0. For a vertex v of T it holds that its height is larger by one unit than its parent.

A vertex with no children is called a *leaf.* Any vertex which is not a leaf is called an *internal node.* A *level* of T consists of all vertices of the same height. The *height* of T is equal to the largest height of any vertex of T. [7, P. 1176ff]
A *k-ary tree* is a rooted tree in which for every vertex has at most k children. A *complete k-ary tree* is a k-ary tree in which all leaves have the same height and all internal nodes have k children.

2.4.2 Outerplanar Graphs, Series Parallel Graphs and 2-Trees

An *outerplanar graph* is a planar graph that can be drawn without crossing in a way so that all vertices are on the outer face. A *maximal outerplanar graph* is an outerplanar graph to which it is not possible to add an edge without destroying the simplicity, planarity or outerplanarity property. A *2-terminal series-parallel graph* with terminals s, t is a recursively defined graph with one of the following three rules:

1. An edge (s, t) is a 2-terminal series-parallel graph
2. If $G_i, i = 1, 2$, is a 2-terminal series-parallel graph with terminals s_i, t_i, then in the serial composition t_1 is identified with s_2 to obtain a 2-terminal series-parallel graph with s_1, t_2 as terminals
3. If $G_i, i = 1, ..., k$, is a 2-terminal series-parallel graph with terminals s_i, t_i, then in a parallel composition we identify all s_i into one terminal s and all t_i into the other terminal t and the result is a 2-terminal series-parallel graph with terminals s, t.

A *series-parallel graph*, *SP-graph* in short, is a graph for which every biconnected component is a 2-terminal series-parallel graph. A SP-graph is *maximal* if no edge can be further added while maintaining a SP-graph. [5, P. 143ff]
A *k-tree* is a recursively defined graph with at least $k+1$ vertices. If $n = k+1$, then the k-tree is the complete graph K_{k+1}. If $n > k+1$, start with a K_{k+1} and every vertex added is adjacent to exactly k adjacent neighbours. The class of *2-trees* correspond to the class of maximal SP-graphs [1, Page 2].

2.5 Tools

2.5.1 $SPQR$-Tree

A *cut vertex* in a graph G is a vertex whose removal disconnects G. A *separation pair* in G is a pair of vertices whose removal disconnects G. A *biconnected component* of G is a maximal (in terms of vertices and edges) biconnected subgraph of G. If G contains vertices s, t, then G is *st-biconnectable* if $G \cup \{s, t\}$ is biconnected. A *split pair* of G is either a separation pair or a pair of adjacent vertices of G. A *maximal split component* of G in regard to a split pair $\{u, v\}$ is either an edge (u, v) or a maximal subgraph G' of G such that G' contains u and v and $\{u, v\}$ is not a split pair of G'. A vertex w aside from u and v belongs to exactly one maximal split component. A *split component* of $\{u, v\}$ is defined as the union of any number of maximal split components of $\{u, v\}$. A split pair $\{u, v\}$ is *maximal*, if there is no split pair $\{w, z\}$ in G such that $\{u, v\}$ is contained in a split component of $\{w, z\}$.
The *SPQR-Tree* $\mathcal{T}$ of G is a recursively defined composition of G with respect to its split pairs. $\mathcal{T}$ is a rooted tree with four types of nodes: S, P, Q and R. Any node μ of $\mathcal{T}$ is related to a planar uv-biconnectible multigraph, the so-called *skeleton* of μ, denoted as $sk(\mu)$. $\mathcal{T}$ is recursively defined - Let (s, t) be an edge of G, called the *reference edge*. $\mathcal{T}$ is initialized with a Q node ϕ as root, representing the edge (s, t). The skeleton of ϕ consists of two parallel edges (s, t). One is a *real* edge, one is a *virtual* edge.
After the initialization of $\mathcal{T}$ with an arbitrary reference edge, consider a node

ψ of $\mathcal{T}$, $G_\psi = (V(G), E(G) \setminus \{(s,t)\})$ and a pair of vertices $\{u, v\}$ of G_ψ, called the *poles* of ψ.

Trivial case If G_ψ consists of a single edge (u, v), then ψ is a Q-node. In this thesis, this case will be denoted as $Q(u, v)$.

Series case If G_ψ is not a single edge and not biconnected, then ψ is a S-node with at least one cut vertex c between the path between u and v. In this thesis, this case will be denoted as $S(u, c, v)$.

Parallel case If G_ψ is not a single edge, but biconnected with $\{u, v\}$ as a split pair of G_ψ, then ψ is a P-node. In this thesis, this case will be denoted as $P(u, v)$.

Rigid case If G_ψ is not a single edge, biconnected with $\{u, v\}$ not being a split pair of G_ψ, then ψ is a R-node. Due to the properties of the graph classes considered in this thesis, this case will not be covered in application and is not of further interest.

[2, P. 7-8]

2.5.2 Tree decomposition

Let G be a graph, T a tree, and let $\mathcal{W} = (W_t)_{t \in T}$ be a family of vertex sets $W_t \subseteq V_G$ indexed by the vertices t of W. The pair (T, W) is called a *tree decomposition* of G if it satisfies the following three conditions:

1. $V_G = \bigcup_{t \in T} W_t$
2. For every edge $e \in E_G$ there exists a $t \in T$ such that both ends of e lie in W_t
3. For all $v \in V$, there exists a connected subtree T' in T such that $v \in W_{t'}, t' \in V_{T'}$

[8, P. 319]
The *width* of a tree decomposition is defined as

$$tw((T, W)) = \max\{|W_t|, t \in V_T\} - 1 \tag{1}$$

The *treewidth* of a graph is the least width of any tree decomposition of G [8, P. 321]. A graph with a treewidth bound by k is a k-tree [13, P. 5].

3 The Problem

3.1 The Fixed Edge-Length Planar Realization Problem

The *Fixed Edge-Length Planar Realization Problem*, *FEPR* problem in short, asks whether there exists a planar straight-line drawing of a given graph where the Euclidian length of an edge is given by a function $\lambda : E(G) \to \mathbb{R}^+$.
It was shown, that the *FEPR* problem is generally $\mathcal{NP}$-hard for triconnected graphs with unit lengths as well as biconnected graphs with unit lengths, with λ as a constant function. [1, P. 2]

3.2 Formalization of the Problem

The Edge-Length Ratio

Let Γ_G be a given planar polyline drawing. The length of an edge is defined as the sum of $k+1$ line segments, induced by k bends. $l_{\max}$ is the length of the longest edge in Γ_G, $\delta_{\min}$ is the minimal Euclidian distance between two adjacent vertices in Γ_G. Then, the edge-length ratio r of Γ_G is defined as:

$$r_{\Gamma_G} = \frac{l_{\max}}{\delta_{\min}} \tag{2}$$

It trivially holds, that $r \geq 1$, since the length of every polyline with at least one bend between two vertices is naturally longer than their respective Euclidian distance. r is said to be *optimal* if $r = 1$. Then, all the edges in a drawing are straight-lines and of the same length. This corresponds to the *FEPR* problem with λ being a constant function.

Upper Bound of the Ratio

There exist multiple straight-line drawing algorithms which produce a drawing for a planar graph in area $\mathcal{O}(n) \times \mathcal{O}(n)$. Prominent examples are the Shift Method by Fraysseix, Pach and Pollack [17, P. 202ff] and Schnyder drawings [4, P. 3].
The area consumption of a straight-line drawing directly induces the bounds for the ratio. Let $k \times k$ be the area consumption of a square bounding Γ_G, $k \in \mathcal{O}(n)$. The maximal length of a straight-line is then bound by $\sqrt{2}k$, from one corner of the grid to the diagonal opposing one, while $l_{\min}$ might value 1 `UL` . The ratio therefore values $\sqrt{2}k \in \mathcal{O}(n)$ in the worst case.
This automatically gives an upper bound for any poly-line drawing Γ_G since a straight-line drawing can be seen as a polyline drawing with zero bends. Including bends in a straight-line drawing enables the possibility to reposition vertices in order to maximize the Euclidian distances even further.

3.3 On the Edge-Length Ratio of Maximal Series-Parallel Graphs

The class of maximal SP-graphs is of particular interest in this thesis due to their balance in restricted properties on the one hand, and having non-trivial

approaches and results for general problems on the other hand. Maximal SP-graphs are biconnected, but not triconnected. This property implies a high amount of possible embeddings for a given maximal SP-graph G, since parallel subgraphs can be permuted and flipped. Therefore, finding drawings with an optimization regarding a specific problem require combinatorial and algorithmic approches. This effect is pointed out by previous results regarding the edge-length ratio. It was shown that the *FEPR* problem is $\mathcal{NP}$-hard for straight-line drawings for maximal SP-graphs with up to four distinct edge lengths while it is solvable in linear time for uniform edge lengths [1, P. 1].
Also, it was proven that the ratio of straight-line drawings of maximal SP-graphs is unbounded, meaning, that for a given constant r, there exists a sufficiently large maximal SP-graph G with $r_G > r$ over all its straight-line drawings. One result states that the ratio lies in the gap between the lower bound $\Omega(\log n)$ and the upper bound $\mathcal{O}(n^{0.695})$ [6, P. 2].

3.4 The Symposium Challenge

The *Live Challenge* takes place during the Graph Drawing Symposium in 2022. During one hour, teams will compete either manually or automatically with their own tools and implementation. The goal is to optimize the ratio for given graphs on a fixed grid. The team with the highest score achieved regarding the edge-length ratio wins. For teams participating with their own tools, an embedding might not be given with the input. For participants working manually, an embedding is already given beforehand.
For the automatic section of the Live Challenge, the input consists of a JSON file with the following entries:

nodes Every node has an unique ID value between 0 and the amount of nodes - 1, a value for the x and y coordinate each, delimited by the width and height

edges Every edge has an ID for source and destination each and an optional list of bend points, specified in x and y coordinate

width (optional) The maximum x-coordinate of the grid. If unspecified, the width is set to 1,000,000.

height (optional) The maximum y coordinate of the grid. If unspecified, the height is set to 1,000,000.

bends The maximum number of bends allowed per edge

The results of the optimization are also JSON files. The planarity of the graph shall be preserved and the ratio minimized by relocation of the nodes and using suitable bend points.

The Live Challenge addresses the practical aspects of a specific optimization task, including implementing heuristics and testing them beforehand. For the task at hand, the grid is fixed in order to keep a specific area consumption for given planar graphs while maximizing the Euclidian distance between adjacent vertices. In contrast to the Live Challenge, this thesis as a theoretical piece of work addresses approaches to guarantee asymptotic upper bounds for the ratio of drawings regarding certain graph classes. The area consumption will be investigated in asymptotic behaviour while maximizing the Euclidian distances between vertices.

4 Trees

When analyzing a general problem of graph drawings, the class of trees is of interest due to their assessable properties. The first graph class considered for minimizing the ratio in a drawing is therefore the class of k-ary trees. The results of this class of trees will be applicable to any rooted tree. This section describes a drawing algorithm for a k-ary tree T, guaranteeing a nearly optimal Euclidian ratio in the resulting drawing Γ_T. The total area of Γ_T will depend on the height of T.
Trees are connected and acyclic per definition, inheriting exactly $n-1$ edges. These properties make trees accessible for a straightforward solution for a given problem.
When working on other graph classes, it may be possible to find an approach by reducing a graph G to an instance of a tree. In fact, the resulting drawing algorithm for k-ary trees in this section will serve as a first approach for drawing maximal outerplanar graphs in the succeeding section.

4.1 Properties Of Complete k-ary Trees

Lemma 1. *Any tree T can be interpreted as a k-ary tree.*

A tree T does not necessarily inherit a root but any vertex of T can be flagged as such. The height of any vertex is then described as the path length starting from the root and k is set to the maximum amount of children of all the inner nodes. T is now rooted and k fixed, hence a k-ary tree. When describing a tree T as a k-ary tree, the root will be chosen so that the resulting height of T is minimal.

Lemma 2. *The* height *h of a complete k-ary tree T is in $\mathcal{O}(\log n)$.*

Proof.

$$n = \sum_{i=0}^{h} k^i = \frac{k^{h+1}-1}{k-1} \tag{3}$$

$$\Leftrightarrow h = \log_k((k-1)n+1) - 1 \tag{4}$$

$$= \frac{\log((k-1)n+1)}{\log k} - 1 \tag{5}$$

$$\underbrace{\Rightarrow}_{k \text{ fixed}} h \in O(\log n) \tag{6}$$

□

4.2 Drawing Algorithm For Complete k-ary Trees

Theorem 1. *There exists a linear-time drawing algorithm which produces a planar straight-line drawing for complete k-ary trees with a nearly optimal ratio, apart from a rounding error, on area $\mathcal{O}(n^2 \log n)$.*

Proof. The following drawing will be constructed from top to bottom, meaning that the y-coordinates of the children of any vertex v are smaller than the y coordinate of v.

Let $r := k^h$. For a vertex v in height i, consider k equidistant columns with x-coordinates between $x(v) - (k-1) \cdot k^{h-h'-1}$ and $x(v) + (k-1) \cdot k^{h-h'-1}$. These are integer coordinates since the distance between two columns next to each other equals $2 \cdot \frac{k^{h-i}}{k}$. Draw a circle around v with radius r. Choose the grid points v_i on the column closest to the resulting intersections with the constraint that $y(v_i) \leq y(v)$ and connect v with its k children with a straight-line, as illustrated in figure 2.

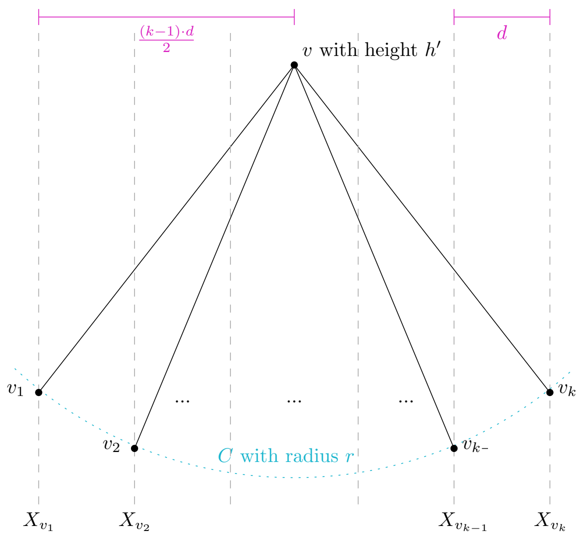


Figure 2: Illustration of drawing algorithm at a vertex v with height h

The distance between two neighbouring columns in height i suffices for the remaining drawing since it holds for the remaining heights:

$$\underbrace{2 \cdot \sum_{j=i}^{h-1} k^{h-j-1}}_{\text{drawn from both columns}} = 2 \cdot \sum_{z=0}^{h-i-1} h^z \tag{7}$$

$$= 2 \cdot \frac{k^{h-i} - 1}{k - 1} < 2 \cdot k^{h-i} \tag{8}$$

Since for a complete k-ary tree, k^{h-1} leafs have to be assigned to a distinct x coordinate, thus the radius of the circles used lies in $\Omega(k^h)$ and therefore in $\Omega(n)$. The height of the drawing is bound by $h \cdot r = h \cdot k^h \in \mathcal{O}(n \cdot \log n)$. The width is bounded by $2 \cdot \sum_{i=0}^{h} k^i = 2 \cdot \frac{k^{h+1}-1}{k-1} \in \mathcal{O}(n)$, resulting in $\mathcal{O}(n^2 \log n)$ area.
Since the algorithm works from top to bottom and for height i, the area for every subtree is disjointedly reserved, the resulting drawing is planar. Furthermore, all straight-line edges inherit a length of approximately k^h. A rounding error bound by $\varepsilon, 0 \leq \varepsilon < 1$ occurs when placing the vertices on grid points. The rounding error does not accumulate since the length r is fixed and the vertices are placed in the grid independently from each other. No bends were used and the ratio is bound by $1 + \varepsilon$. $\square$

The following drawing algorithm sums up the approach described above.

```
Algorithm 1: Draw_$k$-ary_tree($T$)
  Input: complete $k$-ary tree $T$, $h$
  Output: Straight-line drawing of $T$ with nearly optimal ratio
1 $h \leftarrow$ height of $T$
2 $r \leftarrow k^h$
3 Draw $root(T)$ on any grid point
4 Draw_$k$-ary_Children($root(T), r, h$)
5 return $\Gamma$
```

```
Algorithm 2: Draw_$k$-ary_Children($v, r, h$)
  Input: Already drawn vertex $v$, radius and height $r, h \in \mathbb{N}$
  Output: Coordinates of all the children of $v$
1 if $v$ leaf then
2     return
3 else
4     $h' \leftarrow$ height($v$)
5     $d \leftarrow 2 \cdot k^{h-h'-1}$
6     $C \leftarrow \Gamma$.DrawCircle($r, v$)
      /* Draw circle with radius $r$ around $v$ */
7     for $i \in [1..k]$ do
8         $x(v_i) \leftarrow x(v) - \frac{(k-1)\cdot d}{2} + (i-1) \cdot d$
          /* $x$-coordinate of $i$-th child of $v$ */
9         $X \leftarrow \Gamma$.Column($x(v_i)$)
          /* Identify the column at position $x(v_i)$ */
10        $s \leftarrow \Gamma$.Intersection($I, X$)
          /* Calculate the intersection of the circle $C$ and the
             column $X$ */
11        $y(v_i) \leftarrow round(y(s))$
12        $\Gamma$.DrawStraightLine($v, v_i$)
13        $\Gamma$.Draw_$k$-ary_Children($v_i, r, h$)
```

4.3 Drawing Algorithm For General Trees

Theorem 2. *Any tree T admits a optimal straight-line drawing, apart from a rounding error, on exponential area.*

Proof. By Lemma 1, any tree T can be interpreted as a k-ary tree. The algorithm 1 can be reused in order to produce a nearly-optimal straight-line drawing. Set the root $r \in V(T)$ so that the height of T is minimal. Set k as the maximum amount of children of all inner nodes of T. The drawing algorithm 1 is slightly modified which checks the existence of any vertex since any k-ary tree with height h is a subtree of a complete k-ary subtree of the same height. For a smaller radius than k^h, consider the following figure of a tree.

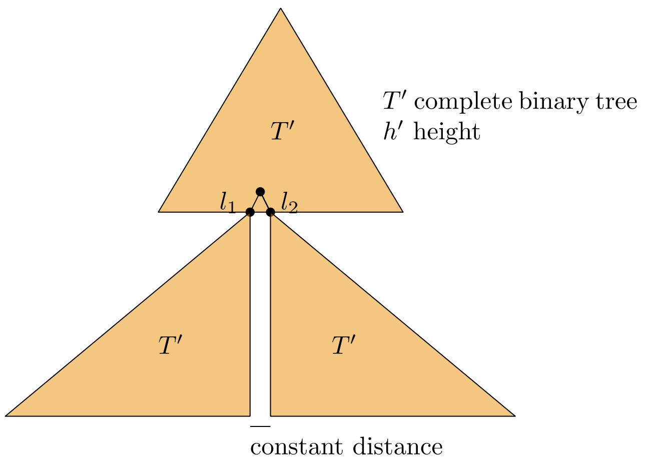


Figure 3: A tree T consisting of three complete binary trees T' in total. Two complete binary trees T' were added to leaves l_1 and l_2 of T'.

In figure 3 a tree consisting of three connected complete binary trees is illustrated. Consider algorithm 3 to create a straight-line drawing of T with nearly-optimal ratio. The distance between l_1 and l_2 is a constant since they share the same parent. Thus with a radius of $k^{h'}$ it is not possible to draw T since there are not enough grid points for the placements of all the leaves of T availiable. The radius must value $k^{2\cdot h'}$.
The height of the drawing is computed by $h \cdot r = h \cdot k^h$. When the height of the tree is in $\mathcal{O}(n)$, then the radius and the area consumption get exponential. The output will be a nearly-optimal straight-line drawing $\Gamma_{T'}$. The width lies in $\mathcal{O}(n)$. Removing marked vertices will not alter the ratio or area consumption and the result is a nearly-optimal straight-line drawing Γ_T. □

The following drawing algorithm sums up the approach for a general tree T.

Algorithm 3: Drawing algorithm for a tree T

```
Input: A tree $T$
Output: Straight-line drawing $\Gamma_T$ with nearly-optimal ratio
1 root $\leftarrow v \in V(T)$ with minimal height for $T$
2 $k \leftarrow 0$ for inner node $v$ of $T$ do
3     if $v$.Children.size() $> k$ then
4         $k \leftarrow v$.Children.size()
5 $h \leftarrow$ height of $T$
6 $r \leftarrow k^h$
7 $\Gamma$.DrawVertex(root)
  /* Draw $root(T)$ on any grid point */
8 $\Gamma$.DrawChildren(root, $r$, $h$)
9 return $\Gamma$
```

Algorithm 4: `DrawChildren`(v, r, h)

```
Input: Already drawn vertex $v$, radius and height $r, h \in \mathbb{N}$
Output: Coordinates of all the children of $v$
1  if $v$ leaf then
2      return
3  else
4      $h' \leftarrow$ height$(v)$
5      $d \leftarrow 2 \cdot k^{h-h'-1}$
6      $C \leftarrow \Gamma$.DrawCircle$(r, v)$
7      $q \leftarrow v$.Children.size()
       /* Draw circle with radius $r$ around $v$ */
8      for $i \in [1..q]$ do
9          $x(v_i) \leftarrow x(v) - \frac{(k-1)\cdot d}{2} + (i-1) \cdot d$
           /* $x$-coordinate of $i$-th child of $v$ */
10         $X \leftarrow \Gamma$.Column$(x(v_i))$
           /* Identify the column at position $x(v_i)$ */
11         $s \leftarrow \Gamma$.Intersection$(I, X)$
           /* Calculate the intersection of the circle $C$ and the
              column $X$ */
12         $y(v_i) \leftarrow round(y(s))$
13         $\Gamma$.DrawStraightLine$(v, v_i)$
14         $\Gamma$.Draw_Children$(v_i, r, h)$
```

4.4 Example Drawings

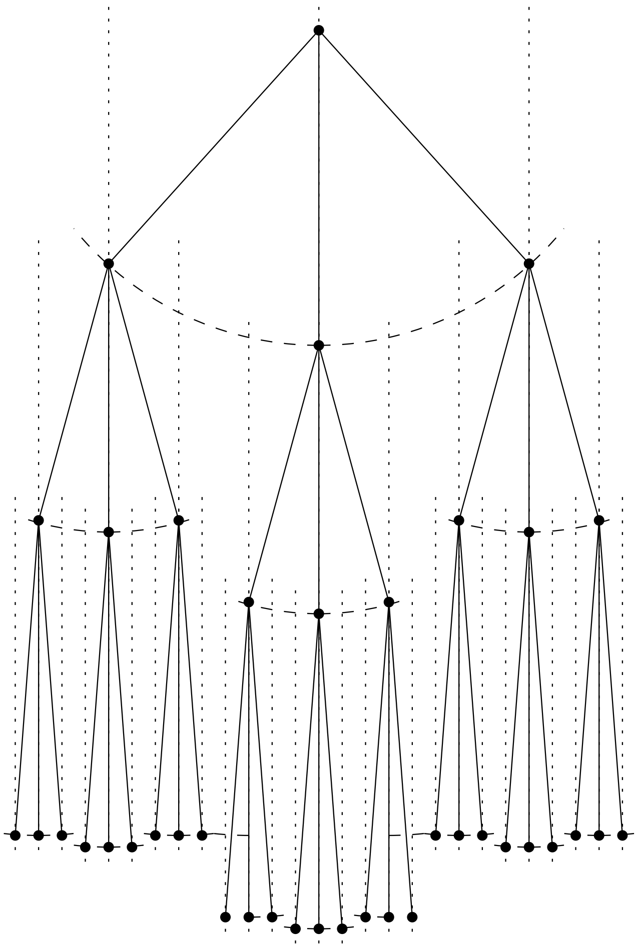


Figure 4: Ternary tree with height 3

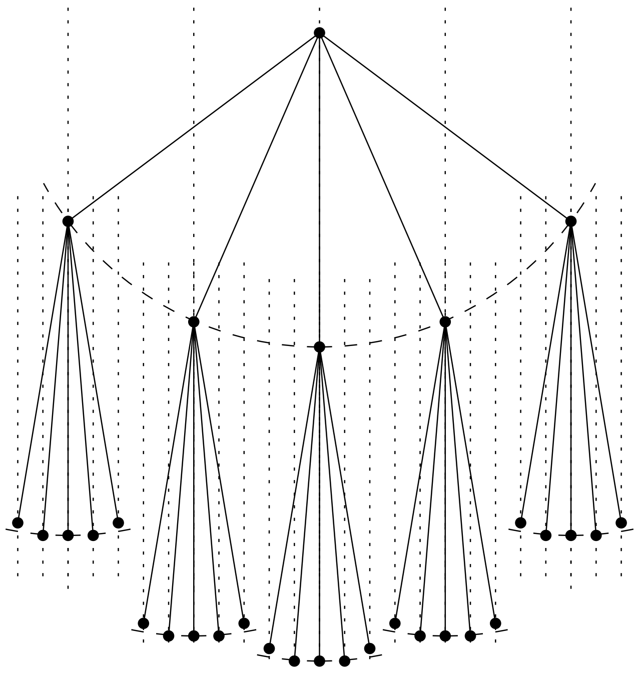


Figure 5: 5-ary tree with height 2

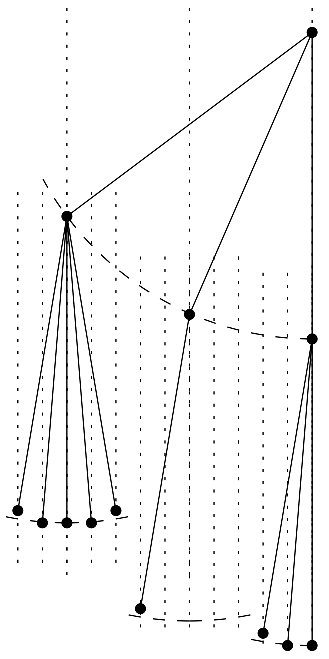


Figure 6: A rooted tree with at most 5 children per inner node

4.5 Implementation

The drawing algorithm for complete k-ary trees as described in this section got implemented as a script in `Python`. The `networkx` library was used to represent the graph structure. For creating a drawing, the `matplotlib` library came in handy since it allows explicite coordinate placements of vertices and a fixed aspect ratio of the resulting drawing.
The script consists of a `main` function and the drawing implementation of algorithm 1. It is possible to set the k and the height when calling the script by command-line arguments. The `main` function evaluates those command-line arguments and calls the recursive coordinate calculations starting from the root vertex with its coordinate $(0, 0)$. All coordinates are rounded to integers in order to produce a straight-line drawing on a grid by default. Afterwards, the map between vertices and their coordinates are forwarded to an instance of the `matplotlib` plot. The output of the script consists of a straight-line drawing, the length of the shortest and longest edge and the ratio of the drawing.

For a full documentation of the implementation, the reader is invited to checkout the `git` repository found on GitHub©[14].

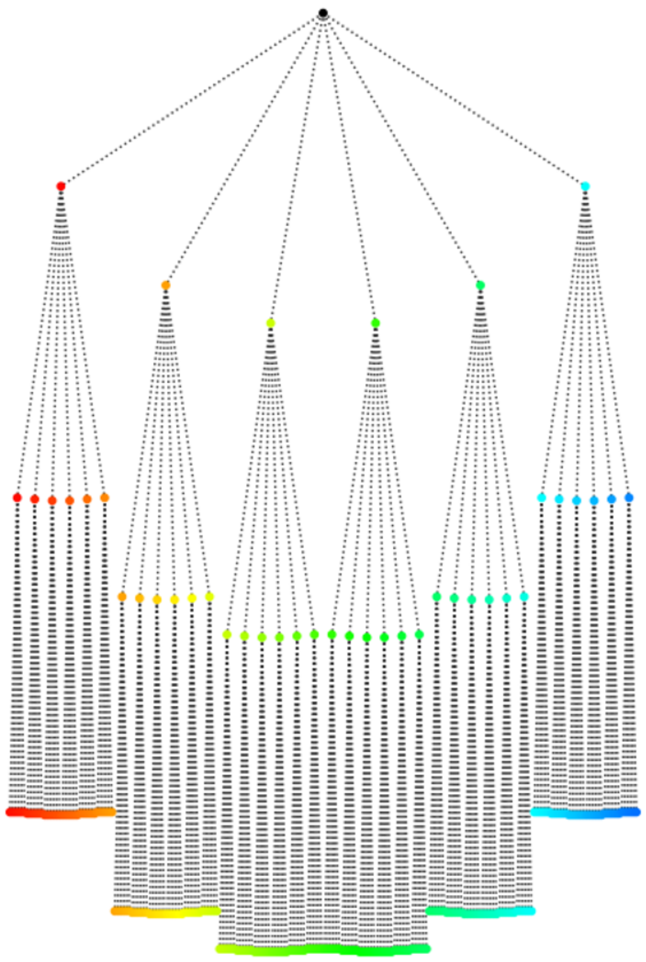


Figure 7: Output drawing of a 6-ary tree of height 3. l_{min} values 215.75, l_{max} values 216.09. The ratio values 1.0015777

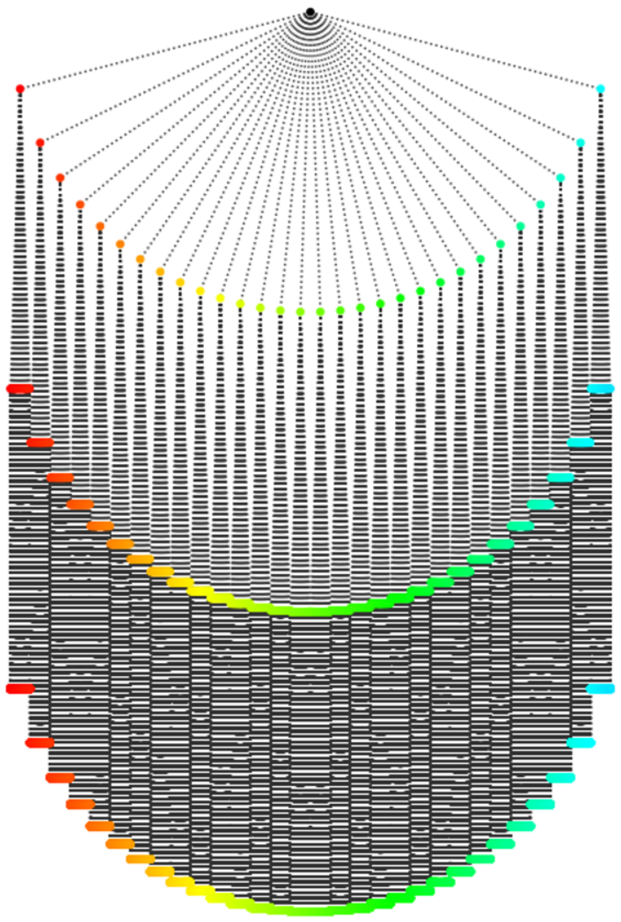


Figure 8: Output drawing of a 30-ary tree of height 3. $l_{\min}$ values 26999.64, $l_{\max}$ values 27000.45. The ratio values 1.00002984

5 Maximal Outerplanar Graphs

This section will address two main approaches of ratio minimization for drawings of maximal outerplanar graphs. Every outerplanar graph can be extended to a maximal outerplanar graph by inserting edges and preserving the outerplanarity property. Any outerplanar graph can then be drawn using a drawing algorithm for maximal outerplanar graphs by inserting edges and removing those after the drawing has been computed. Since any maximal outerplanar graph is a maximal SP-graph, the class of maximal outerplanar graphs is a strict subclass of the maximal SP-graphs and it will be determined whether one of the approaches was suitable to be extended for maximal SP-graphs, which will be the case for the second approach.

5.1 Drawing Algorithm for Complete Outerplanar Graphs with one Bend

After emphasizing general properties of maximal outerplanar graphs, the first approach will address drawing a maximal outerplanar graph based on drawing its weak dual graph. The weak dual graph of an outerplanar graph is a tree and it is already known that any tree admits a nearly-optimal straight-line drawing. Then, the limitations of this approach regarding applying it on maximal SP-graphs are of topic.

5.1.1 Properties of Maximal Outerplanar Graphs

Lemma 3. *A maximal outerplanar graph G inherits triangles as inner faces.*

Lemma 4. *The* weak dual graph G^* *of a maximal outerplanar graph G is a simple tree with maximum degree 3 for any vertex.*

Proof. The weak dual graph G^* is connected since G is maximal outerplanar. Suppose, that G^* contains a cycle $\mathcal{C}$. Then, there exists a vertex in G which is enclosed from the outerface by faces according to $\mathcal{C}$ in G^* and G is not outerplanar. This implies that G^* must be acyclic and considering the connectedness, G^* is a tree. Since any face f is a triangle, the degree of v_f in G^* is at most three. The simplicity is derived from the maximal outerplanarity property. If there were multiple edges between vertex v_f and $v_{f'}$ in G^*, then there would be at least one vertex in G which does not lie on the outerface. □

Lemma 5. *Let G be a maximal outerplanar graph with n vertices and G^* the weak dual graph, a rooted tree with degree up to three for every vertex v_f. Then, the height of G^* ranges between $\Omega(\log n)$ and $\mathcal{O}(n)$.*

Proof. Since G is a planar graph, it contains $\mathcal{O}(n)$ faces. The rooted tree G^* inherits the following property:

1. The root has at most three children

2. The subtrees rooted at the children of the root are binary trees

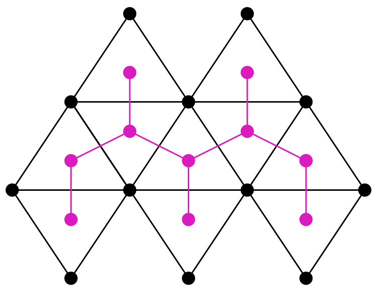
Placing $\mathcal{O}(n)$ vertices in three binary trees connected to a root vertex results in a height of at least $\Omega(\log n)$ due to the k-ary tree height property from Lemma 2. In the worst case, G^* will be a chain of vertices, therefore a rooted tree with height $\mathcal{O}(n)$. □

Lemma 6. *A maximal outerplanar graph G can be extended to a maximal outerplanar supergraph G^+.*

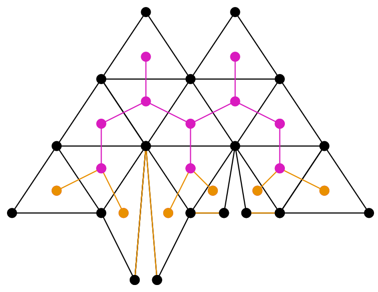
Proof. The vertex insertion works analogously to the recursive definition of a maximal SP-graph. A new vertex can be added by adding a new vertex v_f in the dual graph G^* so that the degree of G^* is still at most 3. The newly created face f then inherits a new vertex for G, adjacent to two preexisting adjacent vertices. □

Observation 1. *Let G be a maximal outerplanar graph with a straight-line drawing Γ_G and a ratio r. Extending G with its drawing Γ_G to a maximal outerplanar supergraph G^+ and Γ_{G^*} with sufficient many vertices, the ratio increases.*

When G^* inherits a height of $\mathcal{O}(\log n)$, a new problem for a drawing emerges. When starting drawing the root of G^*, new vertices are added in all directions, enclosing more and more area along the iterative drawing. This effect is illustrated for an example graph in the figure 9.



(a) A straight-line drawing of a maximal outerplanar graph G with its weak dual graph magenta-colored. The ratio is optimal



(b) Extending G results in a ratio increase due to area restrictions, colored in orange

Figure 9: Illustration of area restriction for dense maximal outerplanar graphs

This results in short Euclidian distances relative to the longest edge, increasing the ratio.

5.1.2 The Algorithm

The first approach of a drawing algorithm addresses a ratio optimization for maximal outerplanar graphs whose weak dual graph is a rooted tree inheriting logarithmic height. These class of maximal outerplanar graphs are described with help of properties regarding the weak dual graph in the following way.

Definition 1. *A maximal outerplanar graph G is called* complete *if its weak dual graph G^* fulfills these properties:*

1. *The root vertex has exactly three children.*
2. *Every other inner node has exactly two children. In other words, the subtrees adjacent to the root vertex are complete binary trees.*

A given maximal outerplanar graph can be drawn by using a drawing algorithm for its weak dual graph. In section 4, we describe a drawing algorithm for k-ary trees that produces a nearly-optimal straight-line drawing. The drawing algorithm 1 can be used with a minor modification to draw the weak dual graph of any complete maximal outerplanar graph G, since G^* is a subtree of a ternary tree with the same height.

Algorithm 5: `DrawOuterWeakDual(`G`)`

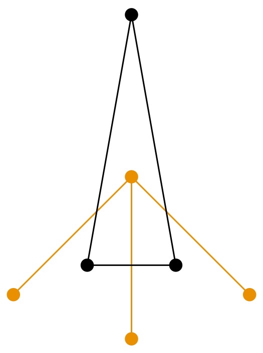
```
Input: A complete maximal outerplanar graph $G$
Output: Straight-line drawing $\Gamma_{G^*}$ with nearly optimal ratio
1 $G^* \leftarrow$ weak dual graph of $G$ with minimal height
2 $h \leftarrow$ height($G^*$)
3 root $\leftarrow G^*$.root
4 Draw(root)
5 Draw_3-ary_Children(root,$3^h$,1)
6 for $v \in$ root.children do
7     Draw_2-ary_Children($v$,$2^{h-1}$,$h-1$)
8 return $\Gamma$
```

Algorithm 5 produces a nearly optimal straight-line drawing for the weak dual graph of a complete outerplanar graph G. The resulting drawing Γ_{G^*} provides assistance to draw the complete outerplanar graph G. Every vertex of Γ_{G^*} serves as an anchor point for the drawing of its corresponding face in G.

Starting at the root of G^*, a triangle is drawn around the root in a way, that each edge of G^* from the root to its three children crosses exactly one edge of the corresponding triangle face in G. The vertices for the triangle are placed as follows. The first vertex lies above the already drawn root with an Euclidian distance to the root sufficiently high in order to preserve planarity for the remaining drawing. The other two vertices are placed inbetween the two triangles defined by the root, its {left, right} and middle children.

In order to guarantee a valid vertex and bend placement at every inner node v^*, the drawing is stretched horizontally and vertically by a factor of three.

Then, the drawing algorithm iterates over the height of G^*. For every vertex v^* of height $i \in \{1, ..., h\}$ in G^*, two bend points are placed 1 `UL` left and right from $\Gamma_{G^*}(v^*)$, and the new vertex v is placed 1 `UL` below from $\Gamma_{G^*}(v^*)$.


(a) Inserting the triangle around the root of G^*

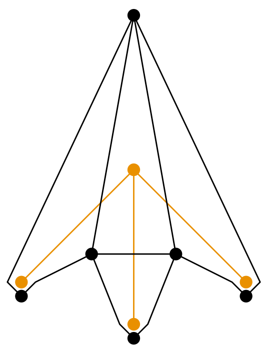

(b) Placing vertices and having one bend per edge preserves outerplanarity

Figure 10: Scheme of creating a polyline drawing of a complete maximal outerplanar graph based on a straight-line drawing of its weak dual graph

v^* is adjacent to an edge e^* defined in G^* that is crossing exactly one edge $e = (v_1, v_2)$ of G. This crossing corresponds to the vertices defining the new face, $\{v, v_1, v_2\}$. Since G^* consists of three complete binary subtrees connected to the root, using the bends to draw the new face will preserve outerplanarity since the coordinate of every inner node inherits its unique x value by construction of algorithm 1 and every vertex lies on the outer face. This drawing approach is summarized in the following algorithm.

```
Algorithm 6: DrawCompleteOuterplanar($G$)
  Input: Complete outerplanar graph $G$
  Output: Polyline drawing $\Gamma_G$ with one bend per edge and ratio
          $\mathcal{O}(\log n)$
1  $\Gamma \leftarrow$ DrawOuterWeakDual($G$)
2  $h \leftarrow$ height($G^*$)
3  for $v^* \in G^*$ do
4      $x(v^*) \leftarrow 3 \cdot x(v^*)$
   /* Draw the triangle face around the root of $G^*$            */
5  $root \leftarrow \Gamma(G^*$.root)
6  $l \leftarrow \Gamma(G^*$.root.leftChild)
7  $m \leftarrow \Gamma(G^*$.root.middleChild)
8  $r \leftarrow \Gamma(G^*$.root.rightChild)
9  $v_1 \leftarrow \Gamma$.PlaceVertex($x(root), y(root) + h \cdot 3^h$)
10 $v_2 \leftarrow \Gamma$.PlaceVertex($\frac{x(root)+x(l)+x(m)}{3}, \frac{y(root)+y(l)+y(m)}{3}$)
11 $v_3 \leftarrow \Gamma$.PlaceVertex($\frac{x(root)+x(m)+x(r)}{3}, \frac{y(root)+y(m)+y(r)}{3}$)
12 $\Gamma$.DrawLineSegment($v_1, v_2$)
13 $\Gamma$.DrawLineSegment($v_1, v_3$)
14 $\Gamma$.DrawLineSegment($v_3, v_2$)
   /* Iterate over the height to draw the remaining vertices
      of $G$                                                     */
15 for $i \in [1..h]$ do
16     for $v^* \in G^*$ with height $i$ do
17         $b_l \leftarrow \Gamma$.PlaceBendPoint($x(v^*) - 1, y(v^*)$)
18         $b_r \leftarrow \Gamma$.PlaceBendPoint($x(v^*) + 1, y(v^*)$)
19         $v \leftarrow \Gamma$.PlaceVertex($x(v^*), y(v^*) - 1$)
20         $e^* \leftarrow (v^*$, parent($v^*$))
21         $e \leftarrow (v_1, v_2)$ edge intersecting $e^*$ // w.l.o.g. $v_1$ ordered left
               of $v_2$
22         $\Gamma$.DrawLineSegment($v, b_l$)
23         $\Gamma$.DrawLineSegment($b_l, v_1$)
24         $\Gamma$.DrawLineSegment($v, b_r$)
25         $\Gamma$.DrawLineSegment($b_r, v_2$)
26 $\Gamma$.delete($G^*$)
27 return $\Gamma$
```

5.1.3 Analysis

Theorem 3. *Every complete outerplanar graph G admits a polyline drawing Γ_G on $\mathcal{O}(n^2 \log n)$ area with one bend per edge, preserving outerplanarity. The drawing is constructed in linear time and the ratio lies in $\mathcal{O}(\log n)$.*

Proof. The triangle around the root of G^* consists of a vertex v_1 placed atop of the root vertex with distance $h{\cdot}r^h$ and two vertices v_2, v_3 placed at the centroids of the triangles defined by {root,root.leftChild,root.middleChild} and {root,root.middleChild,root.rightChild}. The placement of the vertices v_1, v_2 and v_3 guarantee exactly one intersection between an edge of G and an

edge of G^* in Γ.
After stretching the drawing horizontally by a factor of three, there exist free grid points next to every vertex of G^* which guarantees a grid point placement left and right of any vertex v^*. During the iteration over the height starting at height 1, every intersection between an edge of G^* and G refer to vertices on the outerface and a new face of G is attached on the outerface, preserving the outerplanarity of the drawing.
Since v_1 is placed with a distance of $h \cdot r^h$ atop of the root of G^* and v_2 and v_3 partition the x coordinate of the binary subtrees rooted at the children of the root of G^*, planarity is preserved.
The resulting area bounds are asymptotically the same compared to a drawing of a ternary tree since the width and the height are multiplied by a constant. The area consumption lies still in $\mathcal{O}(n^2 \log n)$.
The construction of G^* with minimal height lies in $\mathcal{O}(n)$ since there are $\mathcal{O}(n)$ faces for any maximal outerplanar graph. Drawing G^* and G lies in $\mathcal{O}(n)$. The runtime of this algorithm is therefore in linear time.
The minimal Euclidian distance values at least 2^h by construction of algorithm 1 and lies in $\mathcal{O}(n)$. The length of the longest polyline spans the whole height of the drawing and lies in $\mathcal{O}(n \log n)$. The ratio lies therefore in $\mathcal{O}(\log n)$ since the height of G^* lies in $\mathcal{O}(\log n)$. □

5.1.4 Example Drawing

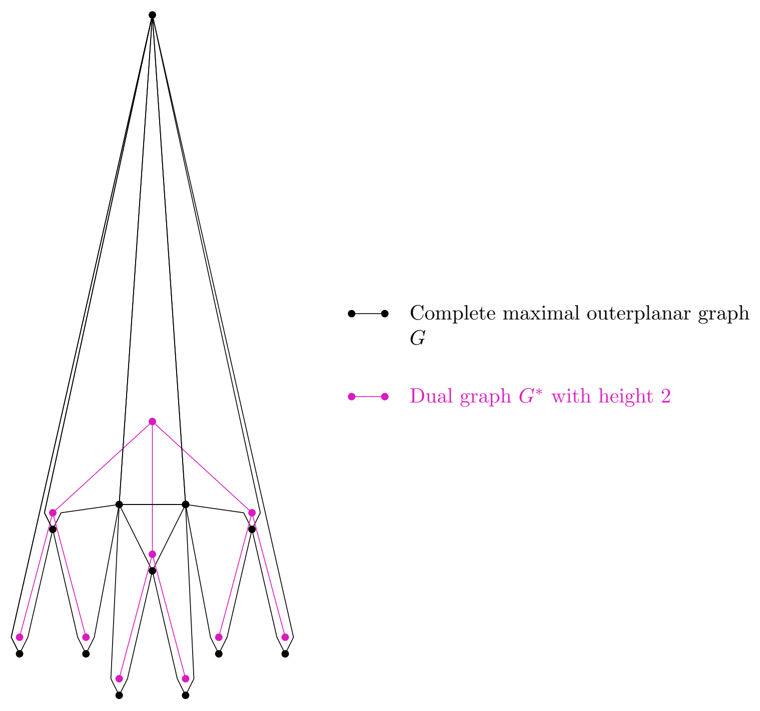

Complete maximal outerplanar graph G
Dual graph G^* with height 2

Figure 11: Polyline drawing of a complete outerplanar graph G. The dual graph G^* inherits a height of 2.

5.1.5 Limitations

As addressed by Observation 1, this algorithm works fine for a complete maximal outerplanar graph G since then, the height of G^* is bounded by $\mathcal{O}(\log n)$.

On the other hand, when G is a not a complete maximal outerplanar graph, the height of G^* might be of linear size. This would worsen the area bounds and the ratio of the resulting drawing drastically since the resulting height is determined by the height of G^* multiplied by a factor of n. The ratio will become linear in the worst case on $\mathcal{O}(n^3)$ area, which is worse compared to any well-established straight-line drawing algorithm producing a drawing on $\mathcal{O}(n^2)$ area with a ratio of $\mathcal{O}(n)$. In addition, the drawing algorithm will only work for simple weak dual graphs. The weak dual graph of a maximal SP graph is a multigraph.

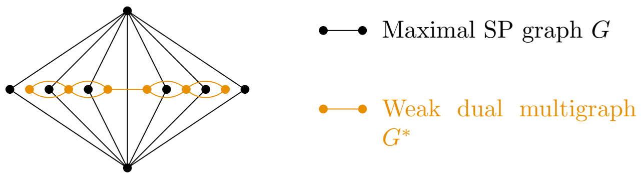


Figure 12: A maximal SP graph with its weak dual multigraph, colored in orange

The approach of drawing the weak dual graph at first followed by the outerplanar graph serves as an idea for a drawing with the *layering property*. The resulting drawings are valid layerings since the vertices of G added on a fixed height of G^* are not adjacent. As already observed, a layered drawing algorithm might guarantee a reasonable ratio by defining a minimal distance between two layers and therefore between two adjacent vertices. Since drawing the weak dual graph is not applicable for SP-graphs, the *tree decomposition* of an SP-graph will serve as a guidance tool for the sequence of vertices drawn by a layering algorithm. The implementation of the drawing algorithm will use the *SPQR-Tree* derived from a tree decomposition.

5.2 Drawing Algorithm for Maximal Outerplanar Graphs with two Bends

The second approach will create a box drawing of a maximal outerplanar graph at first instance. Based on the layering idea acquired by the first approach, the box drawing will inherit the layering property, meaning that no two boxes of adjacent vertices will have the same y coordinate. This layering property will come in handy when maximizing the Euclidian distance between adjacent vertices. The resulting box drawing will then be transfered to polyline drawings as described in section 2. Finally, it will be exposed that this second approach will produce polyline drawings with reasonable area consumption and therefore a reasonable ratio.
Before the drawing algorithm is presented, the *partitioning of binary trees* will be defined, followed by more maximal outerplanar graph properties of interest. The partitioning will prove to be useful to examine upper bounds of the resulting drawings, while the properties will justify the steps of the drawing algorithm.

5.2.1 Partitioning a Binary Tree in Complete Binary Subtrees

When analyzing upper bounds for the ratio of a drawing for a maximal outerplanar graph G, it is mandatory to take a look at the structure of its tree decomposition. Like the weak dual graph of any maximal outerplanar graph, the tree decomposition consists of three binary trees connected to the root. The following definition of a *partitioning* describes the difference of a tree T to being a complete binary tree. This definition of tree partitionings is very similar to a tree decomposition and will help during the analysis of the resulting drawing algorithm for any maximal outerplanar graph G.

Definition 2. *A* partitioning $P = (\mathcal{P}, \mathcal{W})$ *of a binary tree T is defined with the following properties:*

- *For every vertex p in $\mathcal{P}$ there exists a bag w in $\mathcal{W}$*
- *Every bag w in $\mathcal{W}$ contains a complete binary subtree of T and is non-empty*
- *If there exists an edge in T between two complete subtrees described by w_1 and w_2 out of $\mathcal{W}$, then p_1 and p_2 share an edge in $\mathcal{P}$*
- *$\mathcal{P}$ is a rooted tree*

A partition *is a pair consisting of a vertex of $\mathcal{P}$ and its corresponding bag $\mathcal{W}$. A partitioning P is* minimal *if every other partitioning P' of T contains a higher amount of partitions. The* size *of a partition is defined by the amount of vertices in its bag.*

Let T be a binary tree with n vertices. For a partitioning P covering all the vertices of T, the number of bags of size $\mathcal{O}(K)$ is bounded by $\mathcal{O}(\frac{n}{K})$, whereas K is bounded by n. Also, any combination of partitions are possible. Presuming that a partitioning covers $\mathcal{O}(n)$ vertices of a graph, any combination of multiple partitionings of constant amount is possible. For example, let $|V(G)| = 2n$. One chain of $\mathcal{O}(n)$ vertices followed by a complete binary tree with n vertices results in a legitimate rooted tree with height $\mathcal{O}(n)$. The minimal partitioning of this example consists of $\mathcal{O}(n)$ partitions in total.

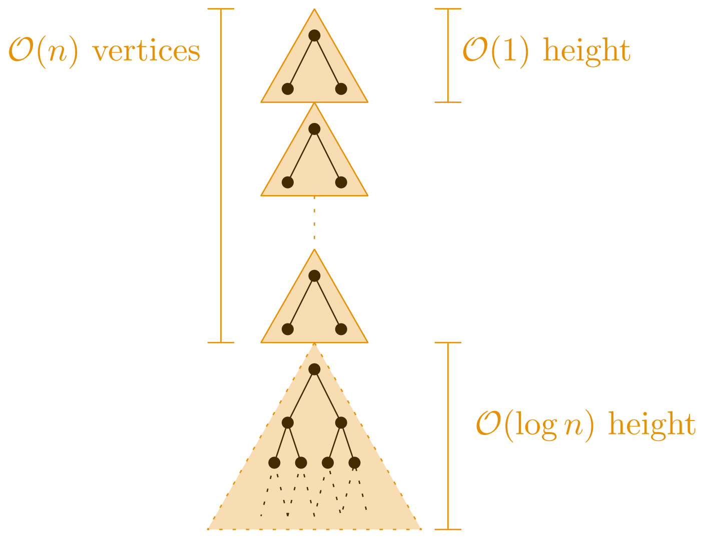


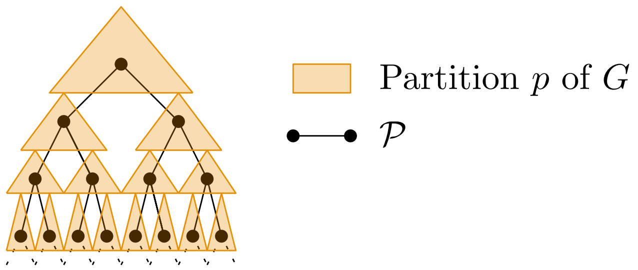
Figure 13: A rooted tree of $\mathcal{O}(n)$ height partitioned in complete binary subtrees (in orange)

The analysis of the amount of layers created in a box drawing for an outerplanar graph G will be based on the structure of its tree decomposition. By Lemma 2, for this analysis it suffices to consider partitions based on binary trees since every complete rooted tree has asymptotically the same height bound.

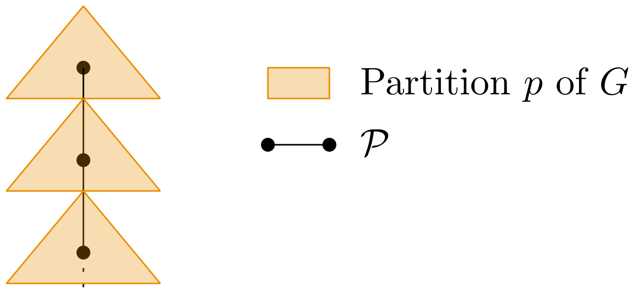
Lemma 7. *The height of G interacts with the minimal partitioning P of G in the following way:*

1. *If P consists of $\mathcal{O}(n)$ many partitions of constant size, the height of G lies between $\Omega(\log n)$ and $\mathcal{O}(n)$*
2. *If P consists of $\mathcal{O}\left(\frac{n}{\log n}\right)$ partitions of size $\mathcal{O}(\log n)$, then the height of G lies between $\Omega\left(\log\left(\frac{n}{\log n}\right) \cdot \log\log n\right)$ and $\mathcal{O}(n)$*
3. *If P consists of $\mathcal{O}(\sqrt{n})$ partitions of size $\mathcal{O}(\sqrt{n})$, then the height of G lies between $\Omega(\log^2 n)$ and $\mathcal{O}(\sqrt{n}\log n)$*
4. *if P consists of a constant amount of partitions of size $\mathcal{O}(n)$, then the height is bounded by $\mathcal{O}(\log n)$.*

Proof. Let T be a binary tree and $P = (\mathcal{P}, \mathcal{W})$ its minimal partitioning. Without loss of generality, it is presumed that all partitions of P are of the same size. Since $\mathcal{P}$ is a tree, figure 14 illustrates the extremal cases for $\mathcal{P}$ are considered for the height of T:



(a) $\mathcal{P}$ represents a complete binary tree



(b) $\mathcal{P}$ represents a chain of vertices

Figure 14: Extremal cases regarding the structure of P of a partitioning $\mathcal{P}$

One extremal case covers $\mathcal{P}$ being a complete binary tree. The other extremal case covers a chain of vertices of $\mathcal{P}$. For case 14a, $\mathcal{P}$ is a tree with height $\mathcal{O}(\log|\mathcal{P}|)$ and up to $\mathcal{O}(|\mathcal{P}|)$ height in case 14b.

1. A complete binary tree of constant size inherits a constant height. With $|\mathcal{P}| \in \mathcal{O}(n)$, substituting every vertex of $\mathcal{P}$ with complete binary trees of constant height does not alter the height asymptotically and the height bounds are valid.
2. A complete binary tree with $\mathcal{O}(\log n)$ vertices inherits a height of $\mathcal{O}(\log \log n)$. The height of $\mathcal{P}$ lies in $\mathcal{O}\left(\log\left(\frac{n}{\log n}\right)\right)$ for case 14a and $\mathcal{O}\left(\frac{n}{\log n}\right)$ for case 14b. Therefore, the height of G lies in $\mathcal{O}\left(\log\left(\frac{n}{\log n}\right) \cdot \log\log n\right)$ for case 14a and $\mathcal{O}\left(\frac{n}{\log n} \cdot \log\log n\right)$ for case 14b.
3. If every vertex of $\mathcal{P}$ represents a complete binary tree of height $\mathcal{O}(\log \sqrt{n}) = \mathcal{O}(\log n)$, the resulting total height of G is in $\mathcal{O}(\log^2 n)$ for case 14a and in $\mathcal{O}(\sqrt{n}\log n)$ for case 14b.
4. A constant multiple of $\log n$ height for a vertex in $\mathcal{P}$ results in a total height of $\mathcal{O}(\log n)$ for G.

□

After defining a partitioning for binary trees, the next section will address properties of an outerplanar graph G regarding its tree decomposition. It will be shown that the tree decomposition of G consists of up to three binary trees connected to the root and why DFS for traversal of the tree decomposition will be a good approach for a drawing algorithm.

5.2.2 More Properties of Maximal Outerplanar Graphs

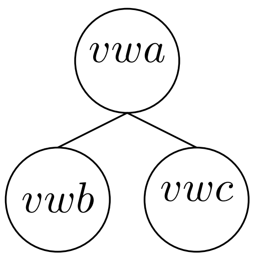
Lemma 8. *Any outerplanar graph G has a tree decomposition (T, W) such that T is of degree at most 3. T is isomorphic to the weak dual graph G^*.*

Proof. Consider the weak dual graph G^* considered in Definition 1. For vertices v_1^*, v_2^* in G^*, insert vertices t_1, t_2 in T which are adjacent if v_1^*, v_2^* are adjacent in G^*. The corresponding bags w_1, w_2 contain the vertices of G which define the face referred by v_1^*, v_2^* in G^*. T is isomorphic to G^* and therefore is of degree at most 3. □

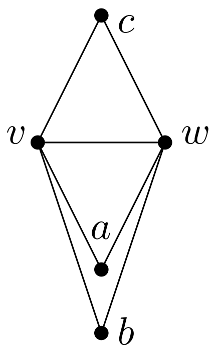
The tree decomposition of an outerplanar graph G will be traversed during the drawing algorithm. When a vertex of the tree decomposition is explored, its bag contains the current vertices of G which were either already drawn, but still relevant for the remaining drawing or newly placed. The following properties describe the occurrence of a vertex of G in its tree decomposition.

Lemma 9. *Let v, w be any two adjacent vertices in a maximal outerplanar graph G. Then, the connected subtree T of a tree decomposition of G containing both v and w contains at most two vertices.*

Proof. If T would contain at least 3 vertices, then there would exist three distinct vertices in G forming a 3-clique with v and w, destroying the outerplanarity property of G. See figure 15 for an illustration.



(a) Tree decomposition T consisting of three vertices. Its bags share two vertices of G



(b) Non-outerplanar drawing representing T

Figure 15: Vertices c, d, e forming a clique with a, b destroys outerplanarity

□

Lemma 10. *Let (T, W) be the tree decomposition of a maximal outerplanar graph G, $t_1, t_2, t_p \in V(T)$ and t_1, t_2 are the children of t_p. Then, the following holds:*

1. *For $i = 1, 2$, the bags w_i and w_p share exactly two vertices*
2. *w_1 and w_2 share exactly one vertex*

Proof.
1. Since G is a maximal outerplanar graph and therefore a 2-tree, all bags contain exactly 3 vertices. Since t_p is the parent of t_1, their bags have two vertices in common when adding a vertex to two adjacent vertices of t_p.
2. This follows directly by Lemma 9.

□

Lemma 11. *Let (T, W) be a tree decomposition of a given maximal outerplanar graph G. The subtree T' of T containing a vertex v of G in its bags is a chain of vertices and of length $\mathcal{O}(\texttt{height}(T))$.*

Proof. Assume, there exists a vertex t in T' with degree at least 3. Since every bag is of size 3, there would be more than two bags sharing two common vertices, contradicting lemma 9. Therefore, the degree of any vertex of T' is at most 2 and since T is a tree and therefore acyclic, T' is a chain of vertices. □

The drawing algorithm will produce a box drawing Γ_G at first instance. Vertices are placed as boxes on a layer and edges are placed vertically between layers. Every new vertex or edge inserted into the box drawing will have a greater x coordinate than the previous ones. The box drawing will emerge from left to right during the traversal of the tree decomposition.
The following definition will help describing for how long layer is occupied for the subsequent vertex insertions inside Γ_G.

Definition 3. *When traversing a tree decomposition (T, W) of a graph G with DFS, a vertex $v \in V(G)$ is called* active *as soon as a vertex of the connected subtree T' of T containing v in its bags is explored during DFS. A vertex $v \in V(G)$ is called* finished *if T' has been fully explored by DFS.*

Lemma 12. *Let G be a maximal outerplanar graph and (T, W) its tree decomposition. When using DFS as graph traversal for a drawing algorithm of G, each vertex v of G is active for up to $\mathcal{O}(h_T)$ steps, whereas h_T describes the height of T.*

Proof. This follows directly by Lemma 11. □

Lemma 13. *Let G be a complete maximal outerplanar graph and (T, W) its tree decomposition. When traversing (T, W) by using DFS, there are at most $\mathcal{O}(\log n)$ vertices active simultaneously.*

Proof. Let $v \in V(G)$ and T'_v be the subtree of T, containing v in its bags. By Lemma 12, T'_v is a chain of vertices. Since G is maximal outerplanar, by Lemma 9 it holds that the amount of occurences of any other vertex $W \in V(G) \setminus \{v'\}$ is bounded by 2. Since G is complete, the height of T is bounded by $\mathcal{O}(\log n)$.s Figure 16 illustrates a path from the root of T to a leaf crossing up to $\mathcal{O}(\log n)$ subtrees T'_v.

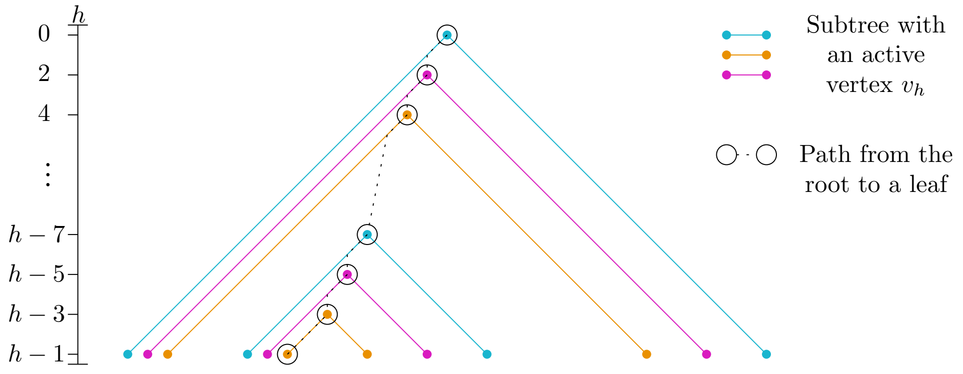

h
0
2
4
⋮
h − 7
h − 5
h − 3
h − 1
Subtree with an active vertex v_h
Path from the root to a leaf

Figure 16: A complete binary tree with up to $\mathcal{O}(\log n)$ active vertices for a DFS path

Let T_r be the root and T_l a leaf of T. Then, the path p from T_r to T_l is unique. For every vertex p_i of p, at most one vertex is set to being active during the DFS graph traversal, when starting at T_r up to T_l. When T_l is explored, there are at most $\mathcal{O}(\log n)$ vertices still active. The following holds for p_i in p:

- If p_i is a leaf and explored, then the induced new active vertex $v'_i \in V(G)$ is already finished since $|V(T_{v'_i})| = 1$.
- If p_i is not a leaf, then all the active vertices induced by exploring p_j with $j > i$ are finished first when using DFS.

This means that up to $\mathcal{O}(\log n)$ vertices are active simultaneously when exploring T. □

Lemma 14. *When prioritizing subtrees of minimal height while traversing any maximal outerplanar graph G with DFS, then there are up to $\mathcal{O}(\log^2 n)$ vertices active simultaneously.*

Proof. Let G be a maximal outerplanar graph with and (T, W) be its tree decomposition. The height of T is bounded by $\mathcal{O}(n)$. Since the degree of any vertex in T is at most 3, there are up to three binary trees connected to the

root of T. The minimal partitioning of the three binary subtrees $P_i = (\mathcal{T}_i, \mathcal{W}_i)$ of T, $1 \leq i \leq 3$, consist of binary tree partitions and $\mathcal{T}_i$ are binary trees.
Consider a minimal partitioning $\mathcal{P}$ and a path $\tilde{t} = (t_1, ..., t_k)$ starting at the root and ending at a leaf of $\mathcal{T}$. When subtrees of minimal heights are prioritized during every step of the DFS of T, then it holds that by the time the DFS reached a partition $t_j \in \tilde{t}$, then all the other partitions $t_i, i < j$ have been almost fully explored, leading to a constant amount of locally active vertices.
In the case of P being a complete binary tree, the priority of the subtree with the smallest height does not take any effect. Considering Lemma 7, the largest upper bound for the height of T lies in $\mathcal{O}(\log^2 n)$ when a minimal partitioning P consists of $\sqrt{n}$ partitions with $\sqrt{n}$ vertices per partition bag and $\mathcal{T}$ being a complete binary tree. analogously to the proof of Lemma 13, traversing a path from the root of T to any leaf will set up to $\mathcal{O}(\log^2 n)$ vertices active. □

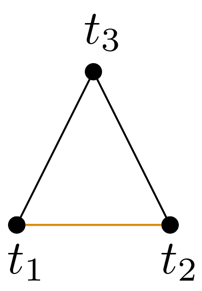
5.2.3 The Algorithm

Based on the fact that the layering induced by the drawing algorithm 6 did improve the ratio for maximal outerplanar graphs whose weak dual graph height is bounded by $\mathcal{O}(\log n)$, this new approach will create a box drawing with the layering property. In order to implement the drawing algorithm for a maximal outerplanar graph G, its tree decomposition will be transferred to an $SPQR$-Tree.

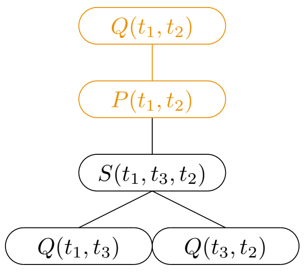
Lemma 15. *There exists a function $f : (T, W) \to \mathcal{T}$ which derives an SPQR-Tree out of a tree decomposition of a maximal SP-graph G with the following properties:*

1. *There is no R node in $\mathcal{T}$*
2. *The skeleton of any serial node S consists of exactly three vertices s_1, s_2, s_3*
3. *The asymptotic height of $\mathcal{T}$ equals the asymptotic height of T*

Proof. Let G be an maximal SP-graph and (T, W) its tree decomposition. Let t_r be the root of T, representing a triangle of the vertices v_1, v_2, v_3. Its $SPQR$-Tree $\mathcal{T}$ consists of three Q vertices, one for every edge of the triangle, one S node and one P node. $\mathcal{T}$ is illustrated in the figure below:



(a) A triangle



(b) An $SPQR$-Tree of 17a

Figure 17: A triangle illustrated in figure 17a and its $SPQR$-Tree in figure 17b. The reference edge is marked in orange.

When traversing T, every newly explored vertex of T represents an addition of a new vertex of G to $\mathcal{T}$. Let v be the vertex of interest, adjacent to v_1 and v_2. v_1 and v_2 are already part of $\mathcal{T}$ and since the edge (v_1, v_2) exists, there is a Q node in $\mathcal{T}$ representing this edge. This Q node is denoted as $Q(v_1, v_2)$. There are two cases to consider:

Case 1: $Q(v_1, v_2)$ is adjacent to an S node

If $Q(v_1, v_2)$ is part of a serial composition in $\mathcal{T}$, adding v will introduce a parallel composition around the split pair (v_1, v_2) between v and the residual graph $G \setminus \{v_1, v_2\}$. The following figure illustrates the insertion of v into $\mathcal{T}$:

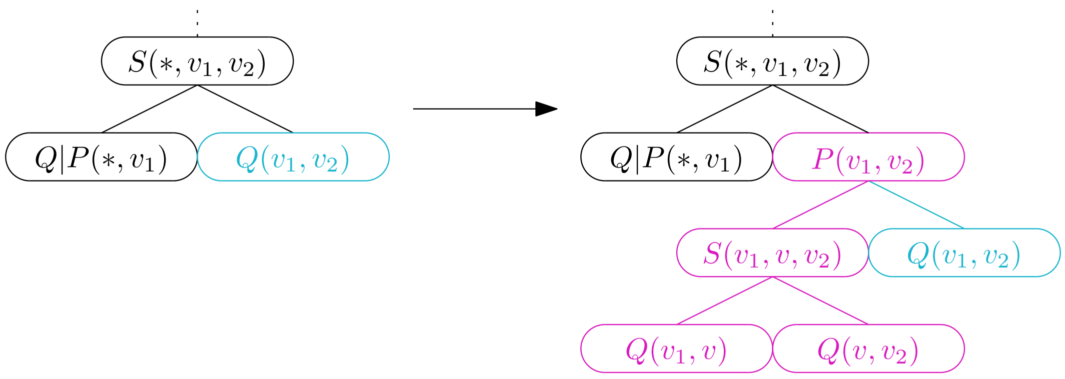


Figure 18: Insertion of a new vertex v when $Q(v_1, v_2$ is not attached to a P node)

Case 2: $Q(v_1, v_2)$ is adjacent to $P(v_1, v_2)$

Since there is already a parallel composition around the split pair (v_1, v_2), inserting v will be a serial composition as illustrated in the following figure:

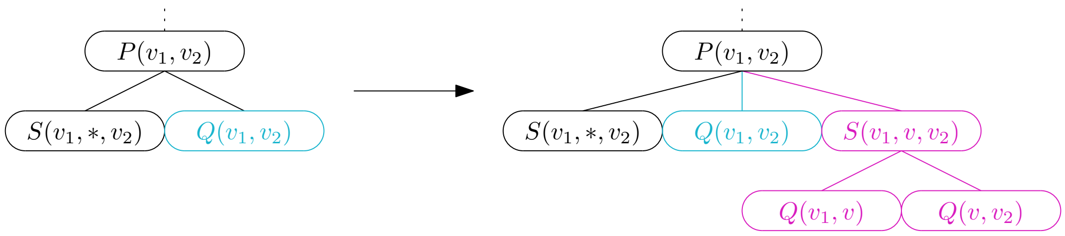


Figure 19: Insertion of a new vertex v when $Q(v_1, v_2$ is not attached to a P node)

Since G is a maximal SP-graph, there is not any R node in $\mathcal{T}$ by definition. The second property holds as an invariant during the process of creating $\mathcal{T}$ since it is true for any triangle and any new vertex insertion. Every vertex of T is substituted with a constant amount of vertices in $\mathcal{T}$, so the height of $\mathcal{T}$ equals the height of T asympotically.
So there exists a function f which derives an $SPQR$-Tree from a given tree decomposition with the stated properties. $\square$

The drawing algorithm will transfer a tree decomposition (T, W) of a maximal outerplanar graph G to an $SPQR$-Tree $\mathcal{T}'$. At first, the triangle represented by the root r of T is drawn, initializing a box drawing with three layers.

Afterwards, the algorithm explores $\mathcal{T}'$ by DFS starting at the root node. This is achieved by a stack implementation in the pseudocode. Every edge insertion will correspond to a trivial case. The parallel case will elongate already existing boxes. Vertex insertions into a box drawing occur when the DFS hits a serial node.
During the serial phase, a new vertex v is placed inside the box drawing and connected to its adjacent vertices a and b. Without loss of generality, let the layer of a be atop the layer of b. It will be checked, if there is a layer available for the box of v to be placed on. If there exists such a layer so that edges (v, a) and (v, b) are drawable without destroying planarity, the layer is called *reachable*. There are three cases to consider regarding the reachability of a layer.

Case 1 The reachable layer is inbetween the layer of a and b.
In this case, place the box representing v on a layer inbetween the layers of a and b and draw the edges accordingly.

Case 2 The reachable layer is either atop a or below b.
Without loss of generality, assume that there is a reachable layer atop of a. The other subcase is symmetrical to this case. Place v on the reachable layer above a and insert edges accordingly. See figure 20 for an illustration.

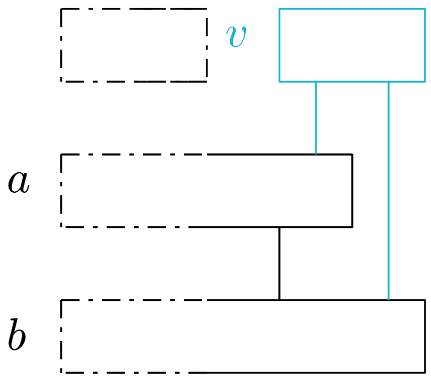
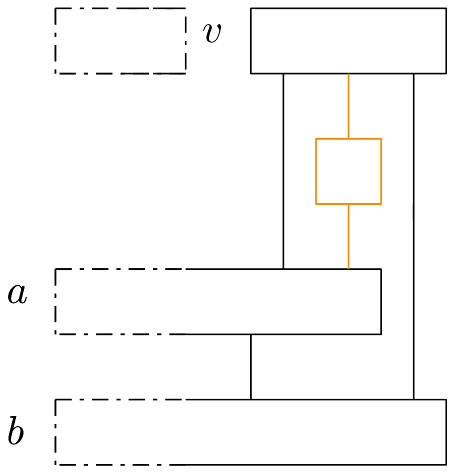
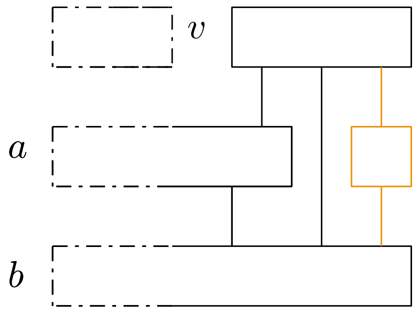


Figure 20: the box v is placed atop the layer of a and edges inserted to a and b

Subsequent vertex insertions inbetween the pairs (v, a) or (v, b) are still possible as illustrated in figure 21. Notice that insertions inbetween (v, a) destroys the outerplanarity property.



(a) Inserting subsequent vertices between v and a, destroying outerplanarity



(b) Inserting subsequent vertices between v and b

Figure 21: Placements for the remaining drawing

Case 3 A layer becomes reachable according to case 1 or 2 by layer reassignments of already drawn boxes.
Moving relevant components of the box drawing to a free area might make a layer reachable for v. See figure 22 for an illustration.

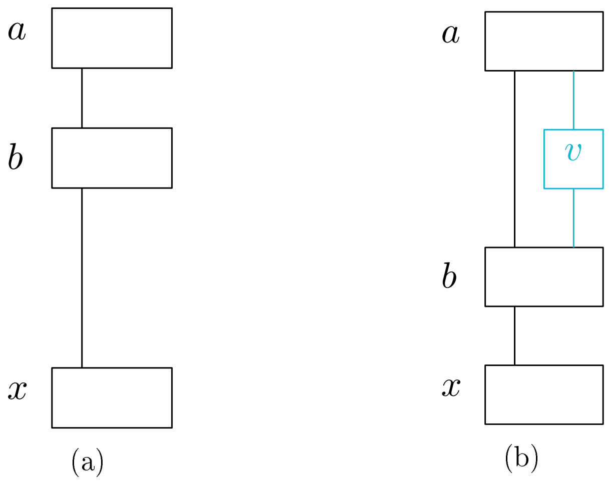


Figure 22: Reassigning b creates a reachable layer for v

If there is no reachable layer available, a new layer is created inbetween a and b.
For a vertex $T \in V(\mathcal{T})$, the child of T which roots the subtree of minimal height

or size will always be prioritized during the DFS. If the heights and size of the subtrees are the same, prioritize the subtree with the longest active vertex. This combined strategy of DFS and subtree priority will exclude unneccesarily inserted new layers in the resulting box drawing for (T, W).

The following pseudocode creates a box drawing for a given $SPQR$-Tree:

Algorithm 7: `DrawSPQR`($\mathcal{T}$)

```
Input: $SPQR$-tree $\mathcal{T}$ of a graph $G$
Output: Box drawing $\mathcal{B}_G$
1  Stack stack
2  SPQRVerticesDone $\leftarrow \emptyset$
3  $v \leftarrow \mathcal{T}$.root stack.push($\mathcal{T}$.root)
4  column $\leftarrow 0$
5  $\delta \leftarrow$ pairwise distance between layers
6  Lower Layer $l_{low} \leftarrow 0$
7  Upper Layer $l_{up} \leftarrow l_{low} + \delta$
8  $B$.DrawBox($q_1, l_{up}$, column)
9  $B$.DrawBox($q_2, l_{low}$, column)
10 ActiveVertices $\leftarrow \{q_1, q_2\}$
11 while VerticesDone $\neq V(\mathcal{T})$ do
12     $v \leftarrow$ stack.pop
13     List children $\leftarrow v$.children
14     children.SortByHeightOfSubtreesDescending
15     while children $\neq \emptyset$ do
16         stack.push(head(children))
            children.remove(head(children))
17     if $v$ is $Q$ node then ;   // $Q$ node refers an edge $(q_1, q_2)$ of $G$
18
19         $B$.DrawEdge($(q_1, q_2)$, column)
20         if $q_1$ finished then
21             ActiveVertices.remove($q_1$)
22         if $q_2$ finished then
23             ActiveVertices.remove($q_2$)
24     else if $v$ is $P$ node then
25         column $\leftarrow$ column $+ 1$
26         for $v \in$ ActiveVertices do
27             $B$.extendBox($v$)
28         $l_{up} \leftarrow layer(p_1)$
29         $l_{low} \leftarrow layer(p_2)$
30     else if $v$ is $S$ node then ;           // Serialization $S(s_1, s_2, s_3)$
31
32         column $\leftarrow$ column $+ 1$
33         for $v \in$ ActiveVertices do
34             $B$.extendBox($v$)
35         if $\nexists$ reachable layer then
36             $l \leftarrow B$.addLayer($l_{up}, l_{low}$)
37         else
38             $l \leftarrow$ reachable layer either atop, below or inbetween $l(s_1)$ and
                $l(s_3)$
39         $B$.DrawBox($s_2, l$)
40         ActiveVertices.add($s_2$)
41     VerticesDone.add($v$)
```

The whole strategy is summed up in the following pseudocode:

```
Algorithm 8: DrawMaximalOuterplanar($G$)
  Input: Maximal Outerplanar Graph $G$
  Output: Polyline drawing $\Gamma_G$ with two bends
1 $(T, W) \leftarrow$ tree decomposition of $G$ with lowest height
2 $\mathcal{T} \leftarrow f(T, W)$
3 $\delta \leftarrow n$
4 $\mathcal{B}_G \leftarrow$ DrawSPQR($\mathcal{T}$)
5 $\Gamma_G \leftarrow \mathcal{B}_G$.TransferToPolyline
6 return $\Gamma_G$
```

5.2.4 Analysis

Algorithm 8 takes a maximal outerplanar graph G as input argument. Calculating the tree decomposition of G with lowest height runs in $\mathcal{O}(n)$ time since there are $\mathcal{O}(n)$ faces in G. Transferring the tree decomposition to an $SPQR$-Tree described by the function of Lemma 15 takes $\mathcal{O}(|V(T)|) = \mathcal{O}(n)$ steps.
The drawing step described in Algorithm 7 takes the $SPQR$-Tree and creates a box drawing by iterating over its nodes. For the $SPQR$-Tree $\mathcal{T}$, there are $\mathcal{O}(n)$ P, S and Q nodes, respectively. In case of a Q node, drawing an edge takes constant time. In case of either a P or a S node, there are $\mathcal{O}(|\texttt{ActiveVertices}|)$ box extensions.
Transferring a box drawing to a polyline drawing includes the substitution of $\mathcal{O}(n)$ boxes with a grid point and $\mathcal{O}(n)$ edge reroutes over two bends per edge. The runtime therefore equals $\mathcal{O}(n \cdot |\texttt{ActiveVertices}|)$.

Theorem 4. *Let G be a complete maximal outerplanar graph. The drawing algorithm 8 produces a polyline drawing Γ_G on area $\mathcal{O}(n^2 \log n)$ with two bends per edge and ratio $r \in \mathcal{O}(\log n)$, when the distance between two layers is set to n.*

Proof. Consider the drawing algorithm 7, producing a box drawing $\mathcal{B}_G$. The algorithm first creates a tree decomposition of G, (T, W), and then uses DFS on T in its iteration to explore new active vertices which will be drawn in a potentially new layer. An active vertex v blocks a layer of $\mathcal{B}_G$ for further vertex insertions. When the DFS finished exploring the subtree T_v of T, the layer of v is availiable for new vertex insertions.
Let p be a path from the root of T to a leaf which creates the highest amount of layers. By Lemma 13, there are at most $\mathcal{O}(\log n)$ vertices active during the DFS graph traversal of the tree decomposition of G. Since T_v is a chain of vertices for any vertex $v \in V(G)$, there exists a subtree of T of asymptotically the same height as T_v which does not contain v in its bags. After drawing $\mathcal{O}(\log n)$ vertices defined by p on separate layers, the amount of layers created in $\mathcal{B}_G$ suffice for the remaining drawing and is asympotically bounded by $\mathcal{O}(\log n)$.
The resulting box drawing therefore inherits $\mathcal{O}(\log n)$ layers with a pairwise distance of n. The height of the drawing is bounded by $\mathcal{O}(n \log n)$. Since for every edge of G there exists a column in $\mathcal{B}_G$, the width is bounded by $O(n)$. The total area consumption therefore lies in $\mathcal{O}(n^2 \log n)$.
When transferring the box drawing $\mathcal{B}_G$ to a polyline drawing Γ_G, every box b_v is substituted with a grid point v inside of b_v and every edge incident to b_v is

rerouted by a bend point to the grid point for v. The polyline drawing produced by algorithm 8 inherits two bends per edge and has asymptotically the same area bound as the box drawing of algorithm 7. The longest edge $l_{\max}$ spans the width and the height and its length is bounded by $\mathcal{O}(n \log n) + \mathcal{O}(n) = \mathcal{O}(n \log n)$. The minimal Euclidian distance between to adjacent vertices values n implied by the layering with its distance and the ratio therefore lies in $\mathcal{O}(\log n)$. □

Theorem 5. *Let G be a maximal outerplanar graph. The drawing algorithm 8 produces a polyline drawing Γ_G on area $\mathcal{O}(n^2 \log^2 n)$ with two bends per edge and ratio $r \in \mathcal{O}(\log^2 n)$, when the minimal distance between two layers is set to n.*

Proof. Considering Lemma 14, up to $\mathcal{O}(\log^2 n)$ vertices are active simultaneously when traversing the tree decomposition of G with DFS and prioritizing the subtrees of minimal height. Analogously to the proof of Theorem 4, there are up to $\mathcal{O}(\log^2 n)$ layers inserted into the box drawing which will suffice for the whole drawing of G. After transferring the box drawing to a polyline drawing, the resulting height lies in $\mathcal{O}(n \log^2 n)$. The width still lies in $\mathcal{O}(n)$. The ratio is therefore bounded by $\mathcal{O}(\log^2 n)$. □

Theorem 6. *Let G be a maximal outerplanar graph and (T, W) its tree decomposition. Furthermore, $P = (\mathcal{P}, \mathcal{W})$ is the minimal partitioning of T. If the height of $\mathcal{P}$ is in $\mathcal{O}(n)$ and the size of every partition is bounded by a constant, then G admits a polyline drawing on $\mathcal{O}(n^2)$ area with a constant ratio.*

Proof. Let G be a maximal outerplanar graph with its partitioning $P = (\mathcal{P}, \mathcal{W})$. Let c be a constant so that for every partition $p_i \in \mathcal{P}$ it holds that $|w_i| \leq c, w_i \in \mathcal{W}$ and $\mathcal{P}$ be a chain of vertices of height $\mathcal{O}(n)$. Since the size of every partition is bounded by c, the height of the regarding complete binary tree is bounded by $\mathcal{O}(1)$.
When $\mathcal{P}$ is a chain of vertices, the drawing algorithm 8 finishes a partition before proceeding to the next element of $\mathcal{P}$ due to the combined strategy of DFS and subtree priority. Since the amount of layers inserted into $\mathcal{B}_G$ are determined by the amount of simultaneously active vertices, $\mathcal{B}$ inherits a constant amount of layers, resulting in a box drawing of $\mathcal{O}(n^2)$ area and a constant ratio. □

Theorem 7. *Any outerplanar graph G with n vertices admits a polyline drawing with two bends per edge and a ratio $r \in \mathcal{O}(\log^2 n)$ on $\mathcal{O}(n^2 \log^2 n)$ area.*

Proof. Insert marked edges into an outerplanar graph G without destroying the outerplanarity until G is maximal. Draw G according to algorithm 8 into a polyline drawing Γ_G with ratio $r \in \mathcal{O}(\log^2 n)$ on $\mathcal{O}(n^2 \log^2 n)$ area, according to theorem 5. Removing the marked edges from G again does not alter the properties of Γ_G. □

5.2.5 An Example

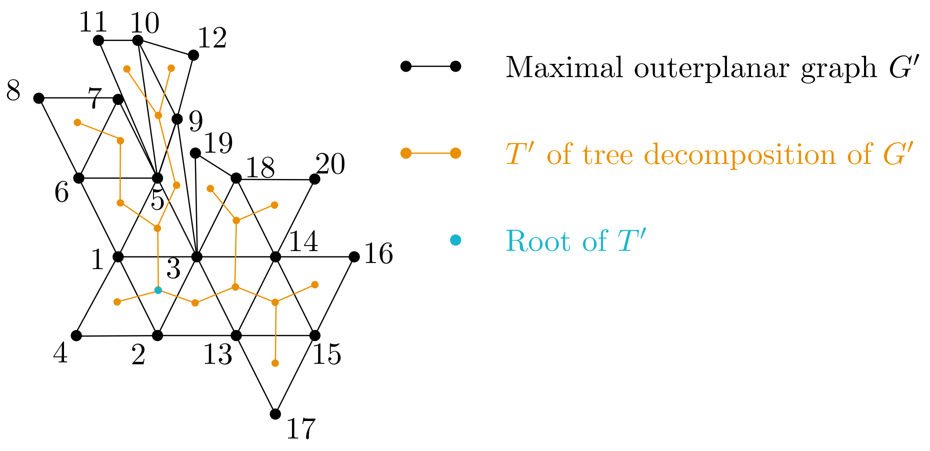
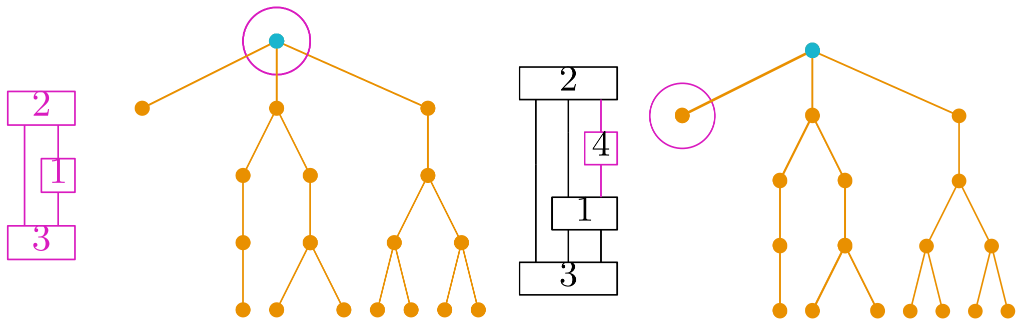

11
10
12
8
7
9
19
18
20
6
5
14
16
1
3
4
2
13
15
17
Maximal outerplanar graph G'
T' of tree decomposition of G'
Root of T'

Figure 23: A straight-line drawing of a maximal outerplanar graph G with T out of its tree decomposition

The drawing algorithm draws the triangle defined vertices $1, 2$ and 3 on three different layers and then explores T by DFS und prioritizes the subtree of minimal height. In the illustration of T, the magenta enclosure illustrates the progression of the DFS.
In order to illustrate an example drawing properly, the distance between layers δ is set to 4.



(a) Box drawing $\mathcal{B}$ of the triangle $T(1, 2, 3)$

(b) Inserting vertex 4 into $\mathcal{B}$

Figure 24: Initializing a box drawing out of the root of T and inserting the first vertex

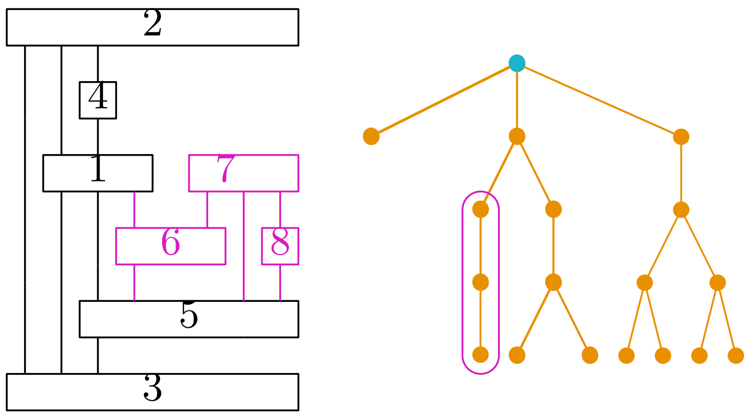


Figure 25: After drawing vertex 6, vertex 1 is finished and the vertices 6 to 8 can alternate between two layers

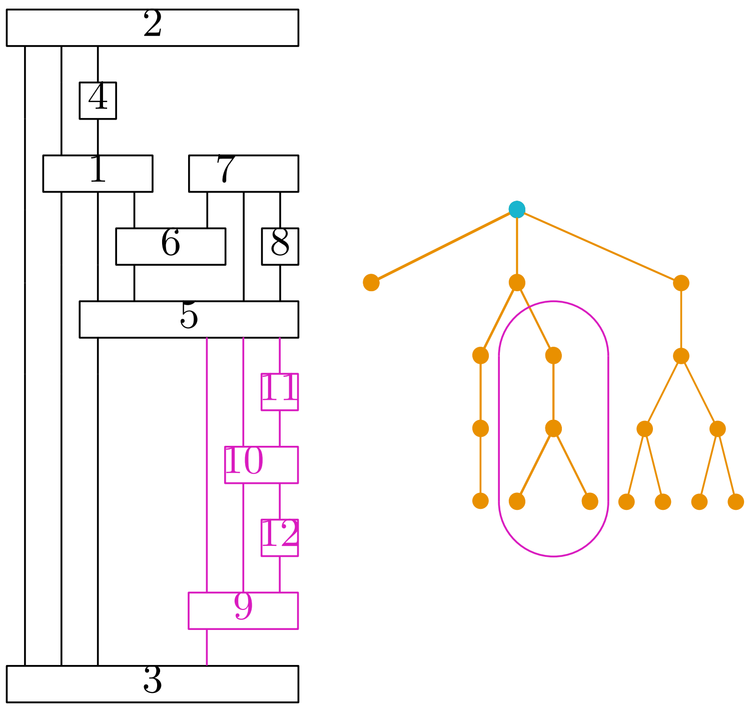


Figure 26: The subtree adjacent to vertices 5 and 3 is drawn

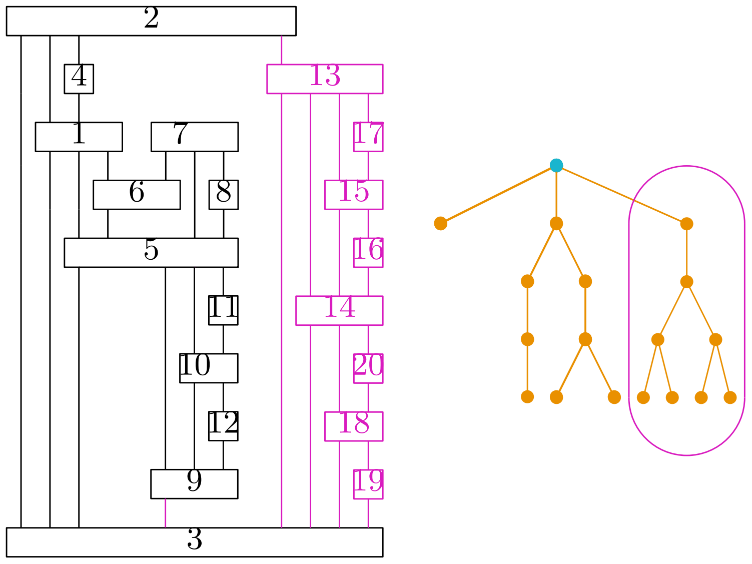


Figure 27: The subtree adjacent to vertices 2 and 3 is drawn

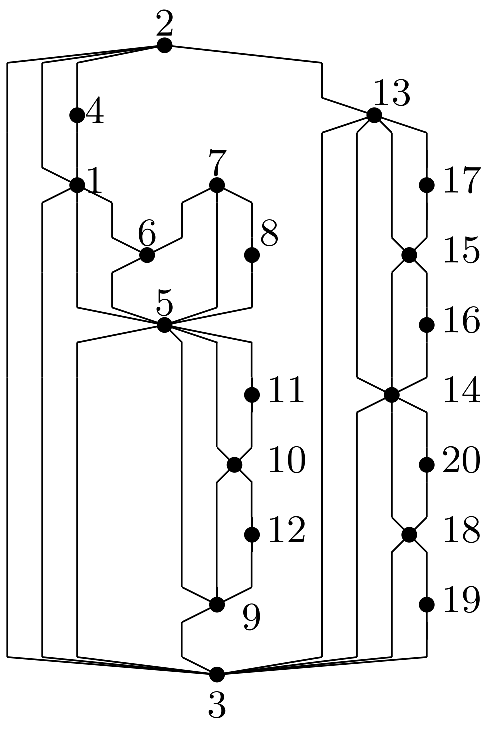


Figure 28: The resulting polyline drawing Γ_G derived from $\mathcal{B}_G$, with two bends per edge

5.2.6 Preserving Outerplanarity

Theorem 8. *There exists an alternation of Algorithm 7 that preserves the outerplanarity property of $\mathcal{G}$ in a resulting box drawing $\mathcal{B}_G$.*

Proof. Start at the root of T, drawing a triangle on three layers. Without loss of generality, let v_1 be on the top layer, v_2 on the bottom layer and v_3 inbetween, placed left of the edge (v_1, v_2). Since G is maximal outerplanar, by Lemma 10 the three binary subtrees adjacent to the root of T address either (v_1, v_2), (v_1, v_3) or (v_2, v_3). If a binary subtree refers to (v_1, v_3), draw to the left inbetween the regarding layers. Analouguously for the binary tree referring to (v_2, v_3). If a binary subtree refers the edge (v_1, v_2), draw to the right. Figure 29 illustrates this scheme of outerplanarity preservation. Since, by lemma 8, T is isomorphic to the weak dual graph G^*, every vertex of T corresponds to a face of G in this figure.

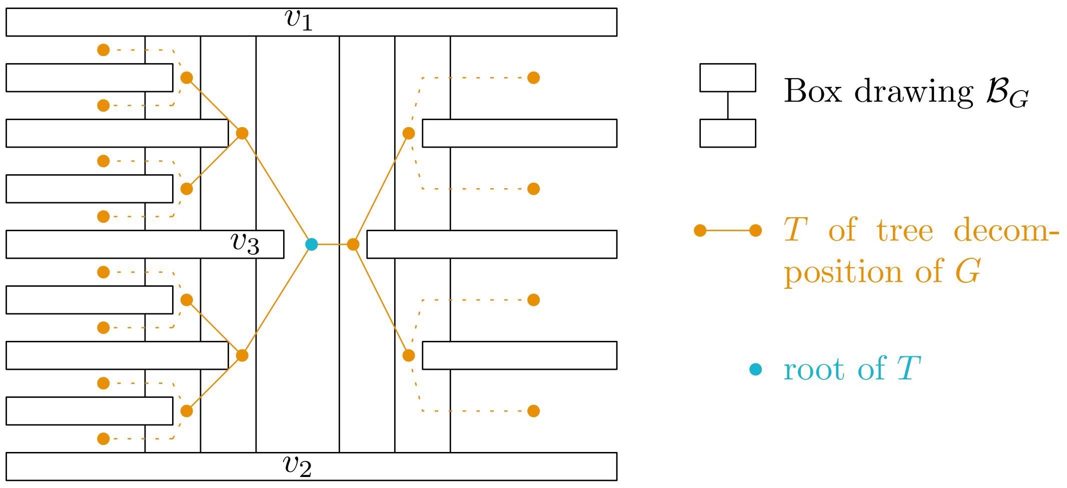


Figure 29: Preserving outerplanarity scheme for a maximal outerplanar graph G

Consider the second case during a serial phase of algorithm 7. When a reachable layer is atop or below the already drawn vertices, there might be the possibility of destroying the outerplanarity property. So in order to preserve the outerplanarity property, the reachable layer atop or below is used if there are no subsequent vertex insertions destroying the outerplanartiy property. Otherwise, a new layer is inserted.
Since every box of $\mathcal{B}_G$ lies on the outerface, every vertex lies on the outerface in the resulting polyline drawing Γ_G. The runtime does not alter asymptotically. $\square$

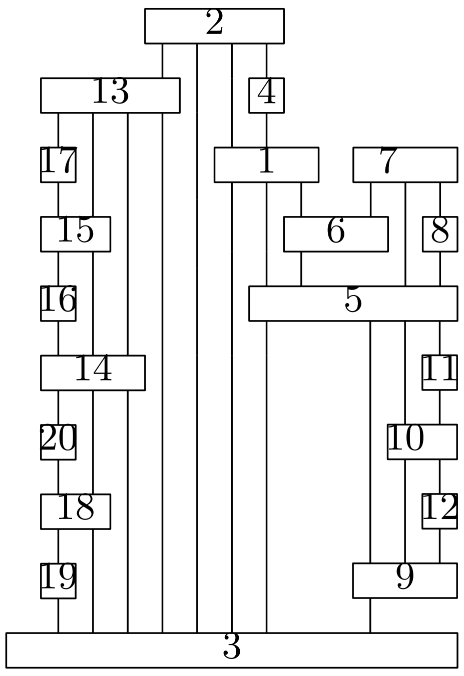


Figure 30: Alternate box drawing of example illustrated at 27, preserving outerplanarity

6 Maximal Series-Parallel Graphs

Maximal series-parallel graphs correspond to the class of 2-trees [1, P. 2] and are a suitable class of interest for fundamental research since any 2-tree is biconnected, but not triconnected and inherits a constant treewidth. Since maximal SP-graph contains a largest maximal outerplanar subgraph, creating a compact layered polyline drawing for maximal outerplanar graphs described in section 5 will prove to be useful when drawing maximal SP-graphs.
At the very end of this section, it will be investigated whether the same approach will improve the ratio for drawings of a even more restricted subclass of planar graphs, the so-called *planar 3-trees*.

6.1 Drawing Algorithm for Maximal Series-Parallel Graphs with two Bends

This approach will extend the drawing algorithm for maximal outerplanar graphs to be suitable for any maximal SP-graph. At first, the largest maximal outerplanar subgraph of a maximal SP-graph is found and a box drawing is drawn according to the second drawing algorithm as described in section 5. The difference will then be inserted at suitable columns. It will be shown that the insertion of the remaining maximal SP-graph in the box drawing will not raise the area bounds significantally and the similar results apply.

6.1.1 Properties of Maximal Series-Parallel Graphs

Lemma 16. *There exists a linear time algorithm which computes the largest maximal outerplanar subgraph for any maximal SP-graph.*

Proof. Consider the tree decomposition (T, W) of G. Add the root of T to T' and its bag to W'. From the root of T, pick the three children which root the subtrees with the maximum height and size and their respective bags. Then, recursively pick the two children which root the subtrees with the maximum height and size and add them to T' with their respective bags to W'. This procedure terminates some leaves of T.
Every vertex of the resulting T' is of degree maximum 3 and by Lemma 8, the graph G' induced by T is maximal outerplanar. Since during this procedure, the nodes with the maximum subtrees are chosen and the properties described in Lemma 10 hold, there exists no other maximal outerplanar subgraph of G which is larger. □

Lemma 17. *Let G be a maximal SP-graph with tree decomposition (T, W) and G' any maximal SP subgraph of G with tree decomposition (T', W'). Then, it holds:*

1. $(T', W') \subseteq (T, W)$
2. *There exists a vertex in $T \setminus T'$ whose bag shares exactly two vertices of G with the bag of the root of T'*

Proof.
- Since the tree decomposition of a maximal SP-graph is unique, any maximal SP subgraph G' of G is represented in T by T'.

- Let t be the root of T'. Consider the parent of t in T. Then, by Lemma 9, t and its parent share two vertices in their respective bags.

□

6.1.2 The Algorithm

A maximal SP-graph G with its tree decomposition (T, W) will be evaluated for its largest maximal outerplanar subgraph G' with tree decomposition (T', W'). $\tilde{T}$ describes the difference of T and T' and is a set of subtrees of T.
In order to create a polyline drawing similarily as described in section 5, it is crucial to find a suiting $SPQR$-Tree derivation of a maximal SP-graphs for the box drawing algorithm 7 to function. Fortunately, the function deriving a $SPQR$-Tree of a tree decomposition of a maximal outerplanar graph G' is also applicable for maximal SP-graphs.

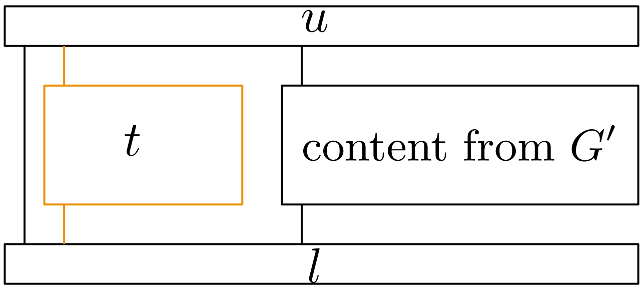
Lemma 18. *The function deriving a $SPQR$-Tree from a tree decomposition defined in Lemma 15 is applicable to any maximal SP-graph G.*

Proof. Let G be a maximal SP-graph with its tree decomposition (T, W). Contrary to a tree decomposition of a maximal outerplanar graph, any node T can have arbitrary many children. Inserting a new vertex described with a vertex in T works analogously to the prior scheme. For a maximal SP-graph, the resulting $SPQR$-Tree can inherit arbitrary many serial nodes attached to a parallel node. □

After (T', W') is drawn into a box drawing B', the remaining subtrees in $\tilde{T}$ will be inserted accordingly, resulting in a box drawing B for G. Finally, a polyline drawing will be derived from B.

Lemma 19. *Let G be a maximal SP-graph and G' its largest maximal outerplanar graph with its box drawing $\mathcal{B}_{G'}$ created by Algorithm 7. For every tree t in $\tilde{t}$ there exists a column in $\mathcal{B}_{G'}$ where t can be inserted into without destroying planarity.*

Proof. Consider the two shared vertices $u, l \in G$ of the root of t with its parent. Since the parent of t is part of G', u and l are placed on their respective layers in $\mathcal{B}_{G'}$ and connected by an edge at column x with its content drawn from G'. Insert t by slicing the box drawing so that there are sufficient free columns beginning at $x + 1$ and draw t. This way, planarity is preserved.



(a) t inserted inbetween upper and lower layer u and l (colored in orange)

□

Algorithm 8: `DrawMaximalSPGraph`(G)

```
Input: Maximal Series Parallel Graph $G$
Output: Polyline Drawing $\Gamma_G$ with two bends per edge
1  $(T, W) \leftarrow$ tree decomposition of $G$
2  $G' \leftarrow$ largest maximal outerplanar subgraph of $G$
3  $(T', W') \leftarrow$ tree decomposition of $G'$
4  $\tilde{T} \leftarrow T \setminus T'$
5  $\Gamma_{G'} \leftarrow$ DrawMaximalOuterplanar$(G')$
6  for $t \in \tilde{T}$ do
7      $p \leftarrow$ parent$(t$.root$)$
8      $u \leftarrow \Gamma_{G'}$.upperLayer$(p)$
9      $l \leftarrow \Gamma_{G'}$.lowerLayer$(p)$
10     $\Gamma_{G'}$.Insert$(t, u, l)$
11 return $\Gamma_{G'}$
```

6.1.3 Analysis

Lemma 20. *When the remaining subtrees of $\tilde{t}$ are inserted into a box drawing $\mathcal{B}_{G'}$, the area bounds do not increase asymptotically.*

Proof. Let G be a maximal SP-graph with $\mathcal{B}_{G'}$ a box drawing from the largest maximal outerplanar graph of G created by Algorithm 7. For the insertion of $t \in \tilde{t}$ into $\mathcal{B}_{G'}$, consider the subtrees adjacent to the parent p of t in T. Since t is not in T', there exists subtree t' of T' adjacent to p which was already drawn.
If the insertion of a tree t would create more layers than drawing t' in the first place resulting in a new upper bound for the height of the drawing, then there would be a maximal outerplanar subgraph of G containing t' which is contradicting the assumption. In the width of the drawing, there are still as many columns as there are edges in G. The area bounds do not increase asymptotically. □

Theorem 9. *Any maximal SP-graph admits a polyline drawing in $\mathcal{O}(n^2 \log^2 n)$ area with two bends per edge and a ratio bound by $\mathcal{O}(\log^2 n)$, when the minimal distance between two layers is set to n.*

Proof. Let $\mathcal{B}_{G'}$ be the box drawing of the largest maximal outerplanar subgraph G' of a maximal SP-graph G. The width of $\mathcal{B}_{G'}$ lies in $\mathcal{O}(n')$. Inserting the remaining subtrees of $\tilde{t}$ increases the width asymptotically to $\mathcal{O}(n)$ since there is at most one column per edge of G. Lemma 20 states that the height bound is asymptotically not altered and analogously to the proof of theorem 5, the ratio of the resulting polyline drawing Γ_G is bound by the amount of total layers, when the minimal distance is set to n. □

Theorem 10. *Any SP-graph G admits a polyline drawing in $\mathcal{O}(n^2 \log^2 n)$ area with two bends per edge and a ratio bound by $\mathcal{O}(\log^2 n)$, when the minimal distance between two layers is set to n.*

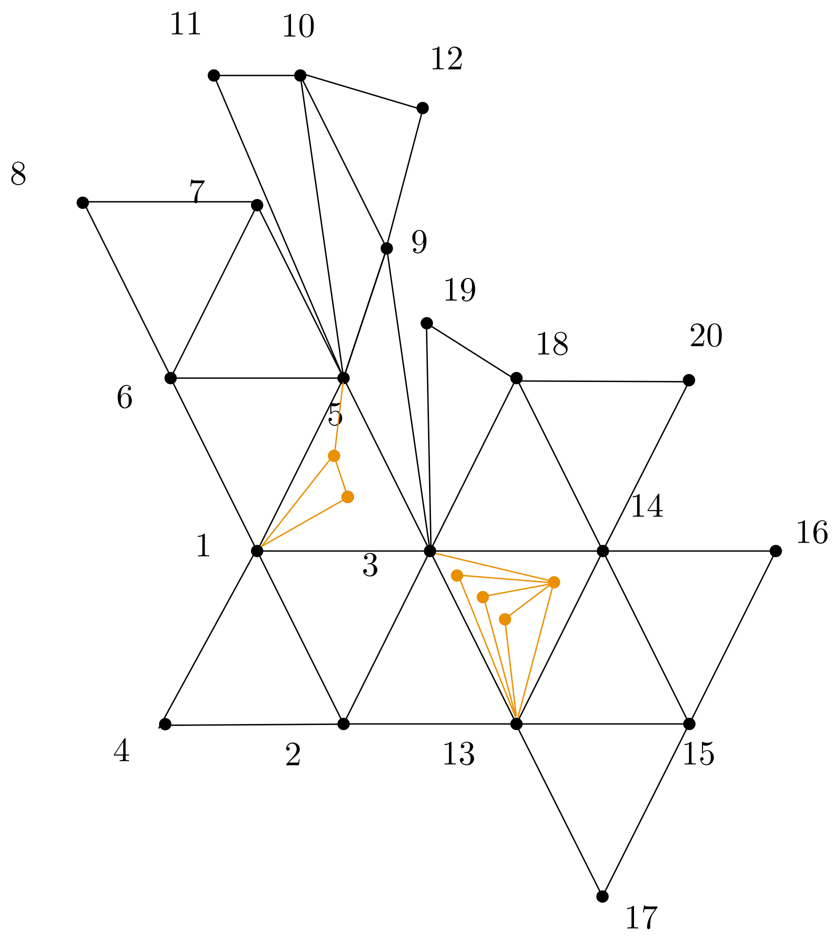
Proof. Insert marked edges until G is a maximal SP-graph. Draw G with algorithm 8 and by theorem 9, the area and ratio upper bounds hold. Removing the marked edges again does not alter any of the upper bounds. □

The last theorem emphasizes the result of a planar small-area drawing for series-parallel graphs, regardless of ratio.

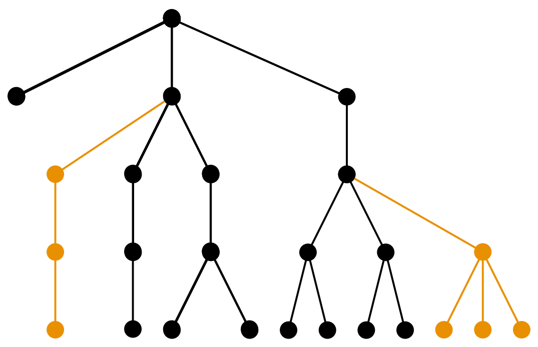
Theorem 11. *Any SP-graph admits a polyline drawing in $\mathcal{O}(n \log^2 n)$ area with two bends per edge.*

Proof. Insert marked edges until G is a maximal SP-graph. Draw G with algorithm 8 and a constant distance between the layers. Afterwards, remove the marked edges from the drawing. □

6.1.4 Example drawing



(a) Maximal SP-graph G with its largest maximal outerplanar subgraph from example 23. The differential subtrees are colored in orange



(b) T of the tree decomposition of G

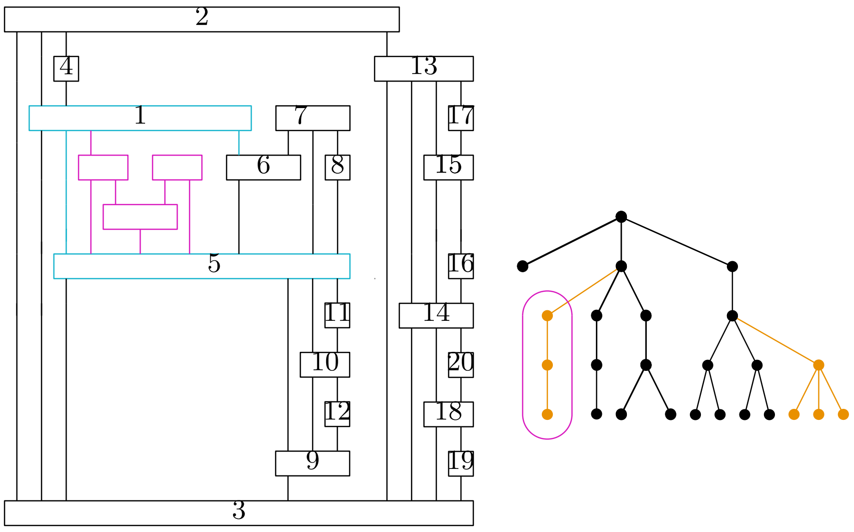


Figure 33: The first differential subtree (encircled in magenta) is inserted between vertices 1 and 5 right next to the edge of $(1, 5)$ (colored in cyan). Since 1 is not finished during the insertion, creating a new layer is mandatory

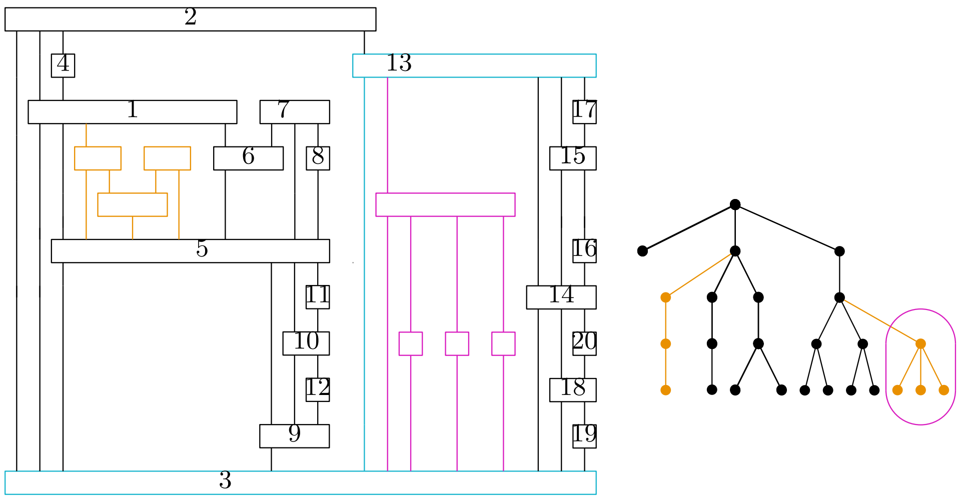


Figure 34: The second differential subtree (encircled in magenta) is inserted between vertices 13 and 3 right next to the edge of $(13, 3)$ (colored in cyan)

6.2 Generalization To Planar 3-trees?

Following the recursive definition of k-trees, 3-trees start with a K_4 and every new added vertex is connected to three adjacent vertices. 3-trees inherit a treewidth of 3 and R nodes in the respective $SPQR$ tree since they are triconnected. The box drawing algorithm 7 does not suffice for this class of planar graphs since the algorithm does not handle the R node case. Still, it is of interest how the idea of a layered box drawing approach will work out based on the tree decomposition of a 3-tree.

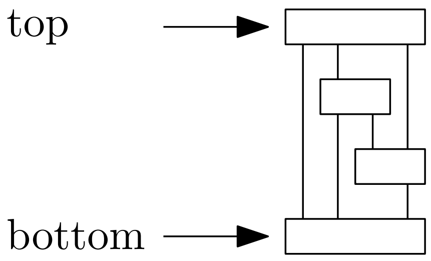
Definition 4. *Let G be a 3-tree with its tree decomposition (T, W). G is called* complete, *if the following property holds:*

- *Every vertex of T is either a leaf or of degree 4*
- *The leaves of T have uniform heights*

In other words, G is called complete if the root of T is the parent of 4 complete tertiary trees of the same height.

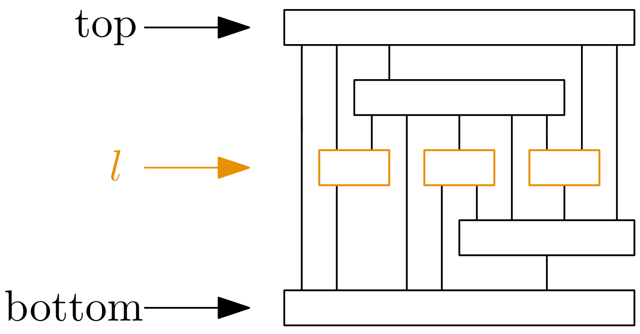
Observation 2. *Let G be a complete 3-tree. Applying a layering scheme similar to algorithm 6, the resulting box drawing $\mathcal{B}_G$ is at least in area bound $\mathcal{O}(n^2)$, when the minimal distance between layers is set to a constant.*

In order validate this observation, a box drawing of K_4 is created as illustrated. K_4 represents the root of the tree decomposition of a complete 3-tree G with height 0 in its tree decomposition T. Let $\mathcal{L}_0$ be the list of layers consisting of the topmost and bottommost layer of the K_4 for $h = 0$.



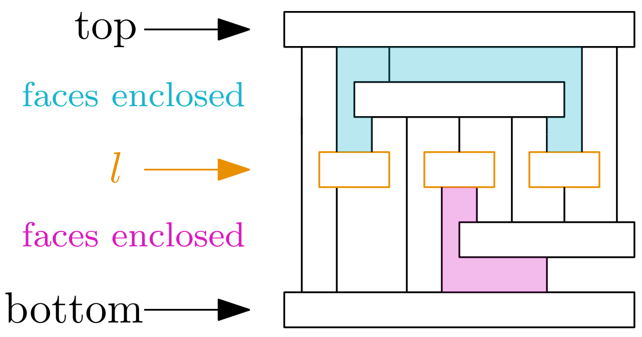
K_4 drawn on four different layers. $\mathcal{L}_0 = (top, bottom)$

The amount of layers created while inserting vertices will determine the height of the resulting drawing. A complete 3-tree G with height h in T is extended to a complete 3-tree G^+ with height $h+1$ when for every inner face f a vertex is inserted and connected to the vertices defining f. These newly created layers are inserted into $\mathcal{L}_h$ according to their position from top to bottom, resulting in $\mathcal{L}_{h+1}$



One layer insertion in $\mathcal{B}$ when extending the height of T from 0 to 1.

Let l be the layer where the new vertices are inserted in according to this scheme. Every vertex insertion creates three new faces out of a pre-existing one. Unfortunately, for any layer l, there will be a face encapsulated between l and both its predecessor and successor in $\mathcal{L}$.



$\mathcal{L} = (top, l, bottom)$

So, in order to insert vertices in all the inner faces so that G is a complete 3-tree and the height of T will be incremented by 1, there are $|\mathcal{L}_h| - 1$ at least new layers necessary.
For a complete 3-tree with height h in its tree decomposition, the total amount of layer insertions is calculated the following way:

$$\sum_{i=0}^{h} 2^i = 2^{h+1} - 1 \underbrace{\in}_{h \in \mathcal{O}(\log n)} \mathcal{O}(n)$$

The resulting box drawing will therefore consume at least $\mathcal{O}(n^2)$ area, when the minimal distance between two layers values a constant. In conclusion, no ratio optimization is possible with this layering approach.

7 Future Work

Tightness Of Ratio Upper Bound For Maximal SP Graphs

As shown in section 5, the amount of simultaneously active vertices during a DFS graph traversal through a maximal outerplanar graphs tree decomposition lies in $\mathcal{O}(\log^2 n)$. One question is whether this upper bound is sharp. It might be possible to deduce the amount of active vertices of G for a path from the root of the tree decomposition to any leaf. The consequence would be a significantly smaller area bound and ratio of any polyline drawing.

Polyline Drawing Implementation For Maximal SP Graphs

An implementation of the drawing algorithm 8 would help evaluating the readability, aesthetics and the ratio of polyline drawings for large-scale maximal SP graphs. In order to implement the algorithm, it is necessary to guarantee consistent data structure representations regarding *undirected graphs* and *SPQR trees* in a programming language of choice. Then, implementing the pseudocodes described in section 5 and 6 would be straight-forward.

Drawing Approach For 3-trees

Since in section **??** it was illustrated that the layering approach based on a tree decomposition is not suitable for a ratio optimization for the class of 3-trees, a new approach for a polyline drawing algorithm is desirable.

Optimize Area Consumption Of Tree Drawings

The drawing algorithm 1 considers a k-ary tree to be complete in order to produce a straight-line drawing. While any tree admits a straight-line drawing with nearly-optimal ratio, the area consumption will be exponentially bounded if the tree is of height $\mathcal{O}(n)$. It would be desirable to construct a drawing algorithm for any input tree so that the area consumption of the resulting drawing stays reasonable and compact while not increasing the nearly-optimal ratio.

8 Acknowledgements

I would like to thank Prof. Dr. Michael Kaufmann for instructing this final thesis. Further I appreciate helpful discussions with Dr. Henry Förster, Julia Katheder and Axel Kuckuk, you guys will shape the teaching in university well for the greater good. I want to thank my parents for encouraging me to absolve the Masters' degree course of computer science.

References

[1] Carlos Alegria et al. “Planar Straight-line Realizations of 2-Trees with Prescribed Edge Lengths”. In: *CoRR* abs/2108.12628 (2021). arXiv: 2108.12628. URL: https://arxiv.org/abs/2108.12628.

[2] Patrizio Angelini et al. “2-Level Quasi-Planarity or How Caterpillars Climb (SPQR-)Trees”. In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021.* Ed. by Daniel Marx. SIAM, 2021, pp. 2779–2798. DOI: 10.1137/1.9781611976465.165. URL: https://doi.org/10.1137/1.9781611976465.165.

[3] Giuseppe Di Battista et al. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999. ISBN: 0-13-301615-3.

[4] Olivier Bernardi and Éric Fusy. “Schnyder decompositions for regular plane graphs and application to drawing”. In: *CoRR* abs/1007.2484 (2010). arXiv: 1007.2484. URL: http://arxiv.org/abs/1007.2484.

[5] Therese C. Biedl. “Small Drawings of Outerplanar Graphs, Series-Parallel Graphs, and Other Planar Graphs”. In: *Discret. Comput. Geom.* 45.1 (2011), pp. 141–160. DOI: 10.1007/s00454-010-9310-z. URL: https://doi.org/10.1007/s00454-010-9310-z.

[6] Vaclav Blazj, Jiri Fiala, and Giuseppe Liotta. “On Edge-Length Ratios of Partial 2-Trees”. In: *Int. J. Comput. Geom. Appl.* 31.2-3 (2021), pp. 141–162. DOI: 10.1142/S0218195921500072. URL: https://doi.org/10.1142/S0218195921500072.

[7] Thomas H. Cormen et al. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009. ISBN: 978-0-262-03384-8. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[8] Reinhard Diestel. *Graph Theory, 4th Edition.* Vol. 173. Graduate texts in mathematics. Springer, 2012. ISBN: 978-3-642-14278-9.

[9] Christian A. Duncan and Michael T. Goodrich. “Planar Orthogonal and Polyline Drawing Algorithms”. In: *Handbook on Graph Drawing and Visualization.* Ed. by Roberto Tamassia. Chapman and Hall/CRC, 2013, pp. 223–246.

[10] *Graph Drawing Symposium 2021.* Graph Drawing Committee. URL: https://algo.inf.uni-tuebingen.de/gd2021/index.php?id=contest (visited on 03/21/2022).

[11] *Graph Drawing Symposium 2021 Contest - Live Challenge.* Graph Drawing Committee. URL: http://mozart.diei.unipg.it/gdcontest/contest2021/index.php?id=live-challenge (visited on 03/21/2022).

[12] *Graph Drawing Symposium 2022 Contest - Live Challenge.* Graph Drawing Committee. URL: http://mozart.diei.unipg.it/gdcontest/contest2022/challenge.html (visited on 03/22/2022).

[13] Jonathan L. Gross. “Embeddings of graphs of fixed treewidth and bounded degree”. In: *Ars Math. Contemp.* 7.2 (2014), pp. 379–403. DOI: 10.26493/1855-3974.366.dd1. URL: https://doi.org/10.26493/1855-3974.366.dd1.

[14] *k-ary tree drawer - A Python Implementation.* Benjamin Coban. URL: `https://github.com/CobbieCobbie/k-ary_tree_drawer/tree/MScThesis` (visited on 06/03/2022).

[15] Ulf Rüegg et al. "A Generalization of the Directed Graph Layering Problem". In: *Graph Drawing and Network Visualization - 24th International Symposium, GD 2016, Athens, Greece, September 19-21, 2016, Revised Selected Papers.* Ed. by Yifan Hu and Martin Nöllenburg. Vol. 9801. Lecture Notes in Computer Science. Springer, 2016, pp. 196–208. DOI: `10.1007/978-3-319-50106-2\_16`. URL: `https://doi.org/10.1007/978-3-319-50106-2%5C_16`.

[16] *Symposia.* Graph Drawing Committee. URL: `http://www.graphdrawing.org/symposia.html` (visited on 03/17/2022).

[17] Luca Vismara. "Planar Straight-Line Drawing Algorithms". In: *Handbook on Graph Drawing and Visualization.* Ed. by Roberto Tamassia. Chapman and Hall/CRC, 2013, pp. 193–222.