Eberhard Karls Universität Tübingen
Wilhelm Schickard Institut Tübingen

Fachbereich Informatik

On Maximizing The Euclidian Distance Between Vertices In Drawings Of Maximal Series Parallel Graphs

Arbeitsbereich Algorithmik

zur Erlangung des akademischen Grades
Master of Science

Autor: Benjamin Çoban
MatNr. 3526251

Version vom: 6. Juni, 2022

ErstprüferIn: Prof. Dr. Michael Kaufmann
ZweitprüferIn: Prof. Dr. Ulrike von Luxburg

Zusammenfassung

Abstract

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 andern 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 [5].

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 [14]. 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 theme and a set of graphs and will have one hour of processing. 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 [9].

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 [10].

In 2022, the 30th International Symposium of Graph Drawing and Network Visualization held in Tokio, Japan [11] 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 6 contains drawing algorithms for the graph class of *series parallel graphs*. The subclass of *outerplanar* graphs and 2-*trees* are of particular interest. Those drawings will improve the worst case ratio behaviour described in Section 3. In section 7, work related to the content of this thesis will be presented and section 8 describes future work.

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* [6, 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 [6, P. 1170]. A graph is *biconnected* if the removal of any vertex still leaves the graph connected [8, 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* [6, P. 1172]. Unless otherwise mentioned, a graph is presumed to be simple.
A graph is *planar* if and only if there exists a crossing-free representation in the plane [6, Page 100]. 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* [7, 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^*

[7, P. 103] The *weak dual graph* of G is the dual graph of G without considering the outer face.
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 [6, P.603].

2.2 Graph Drawing Models And Representations

An $n \times n$ *grid* is a graph consisting of n rows and n 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 [6, 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 [8, P. 225]. In this context, a graph will be drawn on an underlying grid. 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 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. [4, 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) [13, P. 4].
A drawing whose minimum enclosing box has width w and height h is called a $w \times h$-drawing and inherits area $w \cdot h$ [4, P. 145].
The *euclidian distance* 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. [4, P. 145]

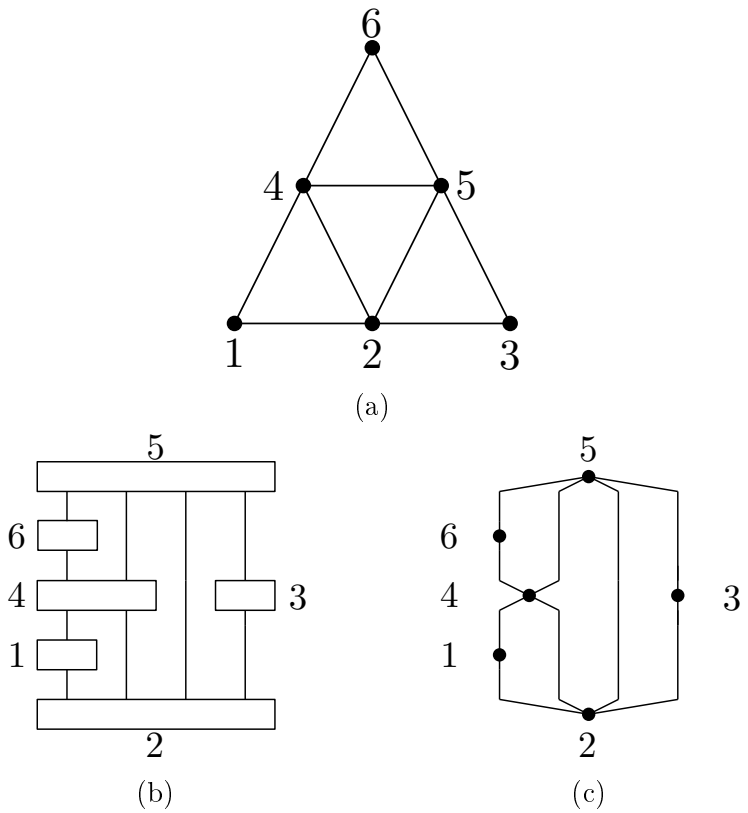


Figure 1: 1a is a straight-line drawing of a maximal outerplanar graph G, 1b is a box drawing of G and 1c is a polyline drawing derived from 1b

2.4 Graph Classes

2.4.1 Tree

A graph T is called *tree* if and only if it is connected, acyclic and undirected [6, P. 1172].
A tree is *rooted* if one of its vertices is distinguished from the other ones, called the *root*. When considering a path from the root to any other vertex w, 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.

A vertex with no children is called a *leaf*. Any vertex which is not a leaf is called an *internal node*. The length of the simple path from the root to any vertex v denotes the *depth* of v in T. A *level* of T consists of all vertices of the same depth. The *height* of T is equal to the largest depth of any vertex of T. [6, P. 1176ff]

2.4.2 k-ary Tree

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 depth and all internal nodes have k children.

2.4.3 Outerplanar Graphs, Series Parallel Graphs and 2-Trees

An *outerplanar graph* is a planar graph that can be drawn such 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. [4, 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. For a 2-tree, it holds that $k = 2$. 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.

[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 $v \in W_{t'}, t' \in V_{T'}$

[7, 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 [7, P. 321].

3 Initial Situation

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

3.2.1 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, $l_{\min}$ is the minimal Euclidian distance between two adjacent vertices in Γ_G. Then, the edge-length ratio r of Γ_G is defined as:

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

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.

3.2.2 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 [15, P. 202ff] and Schnyder drawings [3, 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 bounding square Γ_G is drawn on, $k \in \mathcal{O}(n)$. 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 Ω_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.

3.3 On the edge-length ratio of 2-trees

The class of 2-trees 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. 2-trees are biconnected, but not triconnected. This property implies a high amount of possible embeddings

for a given 2-tree 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 2-trees 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 2-trees is unbounded, meaning, that for a given constant r, there exists a sufficiently large 2-tree 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 $\mathcal{O}(n^{0.695})$ [5, 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. While the Live Challenge during the Graph Drawing

Symposium addresses the practical aspects regarding the task of optimization, this thesis concerns approaches to guarantee an asymptotic behaviour of the ratio for certain graph classes. The grid

3.4.1 A small example

This example will illustrate the issue of optimizing a drawing with respect to its edge-length ratio.

4 k-ary Trees

When analyzing a general problem of graph drawings, considering trees may come in handy 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. 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.
Since any k-ary tree is acyclic per definition, T is connected, but not biconnected and the treewidth of T equals 1. Having n vertices, T inherits exactly $n-1$ edges. The small bags in a tree decomposition and the low amount of edges makes k-ary 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, this effect will occur in the subsequent section of this thesis.

4.1 Properties Of Complete k-ary Trees

Lemma 1. *The* height h *of a complete k-ary tree T is in $\mathcal{O}(\log n)$.*

Proof.

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

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

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

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

□

4.2 The Drawing Algorithm

Theorem 1. *Every complete k-ary tree admits a planar straight-line drawing 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.

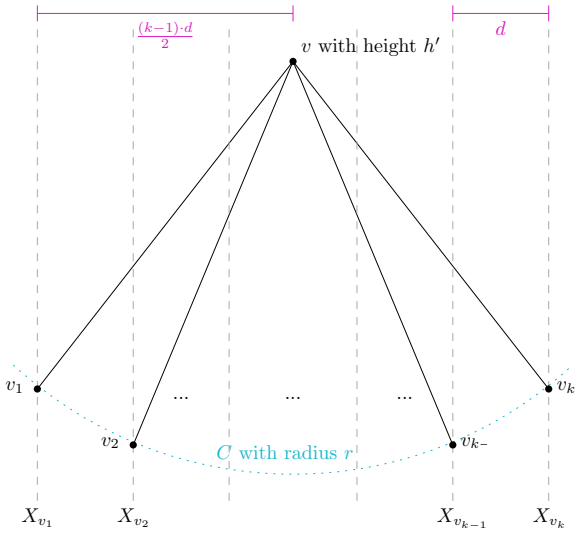


Figure 2: Illustration of drawing algorithm at a vertex v with height h

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

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

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

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 bound 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, no bends were used and the ratio is bound by $1 + \varepsilon, 0 \leq \varepsilon < 1$. □

The following drawing algorithm sums up the approach described above.

Algorithm 1: Draw_k-ary_tree(h)

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** Γ

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' ← height(v)
5      d ← 2 · k^(h−h'−1)
6      C ← Γ.DrawCircle(r, v)
       /* Draw circle with radius r around v                    */
7      for i ∈ [1..k] do
8          x(v_i) ← x(v) − ((k−1)·d)/2 + (i − 1) · d
           /* x-coordinate of i-th child of v                  */
9          X ← Γ.Column(x(v_i))
           /* Identify the column at position x(v_i)           */
10         s ← Γ.Intersection(I, X)
           /* Calculate the intersection of the circle C and the
              column X                                         */
11         y(v_i) ← round(y(s))
12         Γ.DrawStraightLine(v, v_i)
13         Γ.Draw_k-ary_Children(v_i, r, h)
```

4.3 Example Drawings

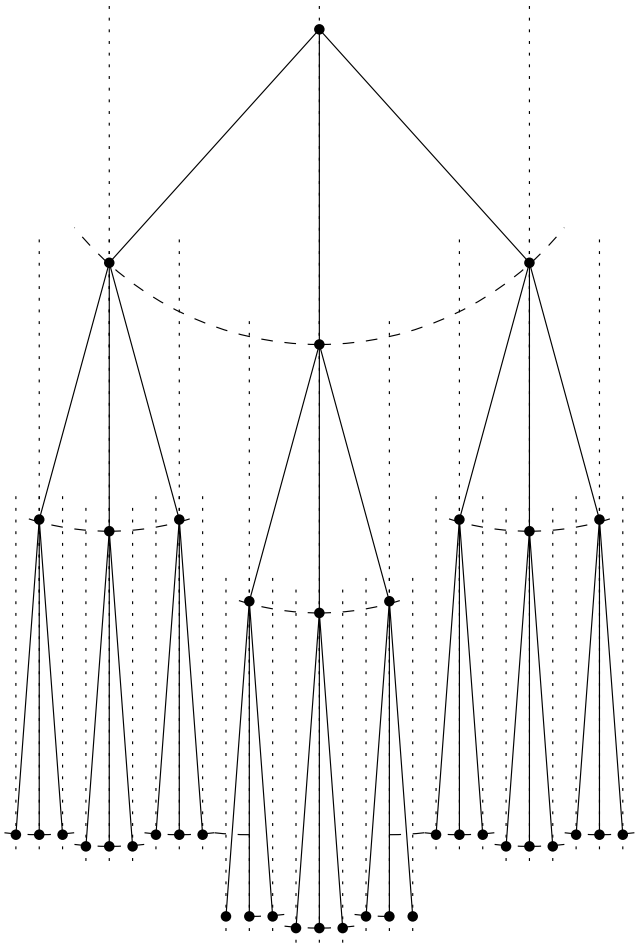


Figure 3: Tertiary tree with height 3

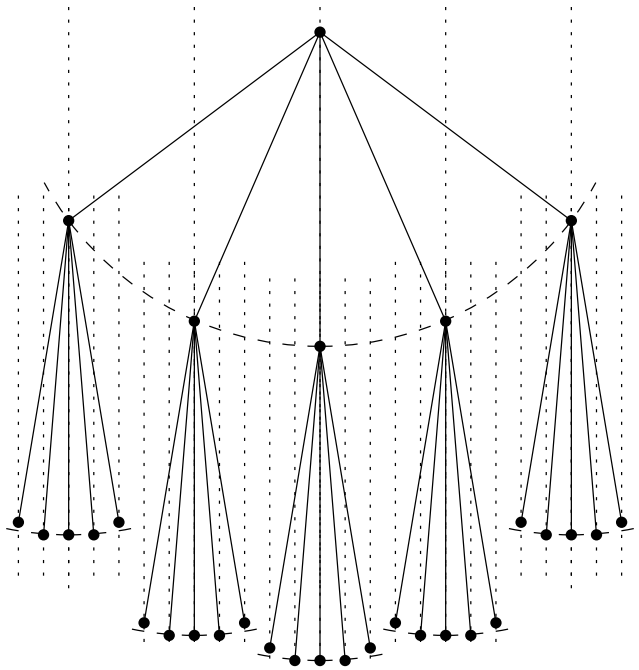


Figure 4: 5-ary tree with height 2

4.4 Implementation

The drawing algorithm for complete k-ary trees as described in thie 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©[12].

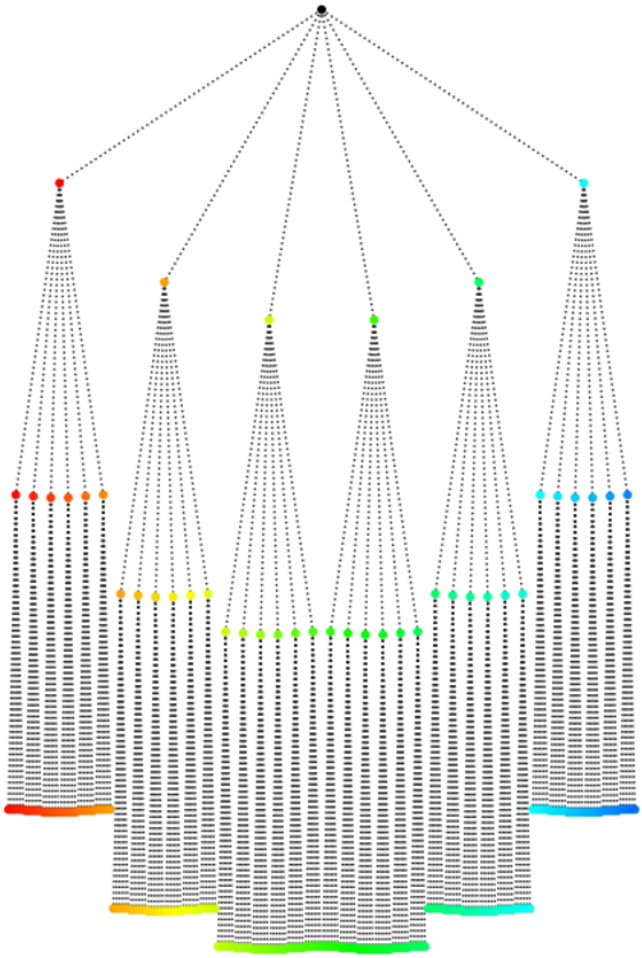


Figure 5: Output drawing of a 6-ary tree of height 3. $l_{\min}$ values 215.75, $l_{\max}$ values 216.09. The ratio values 1.0015777

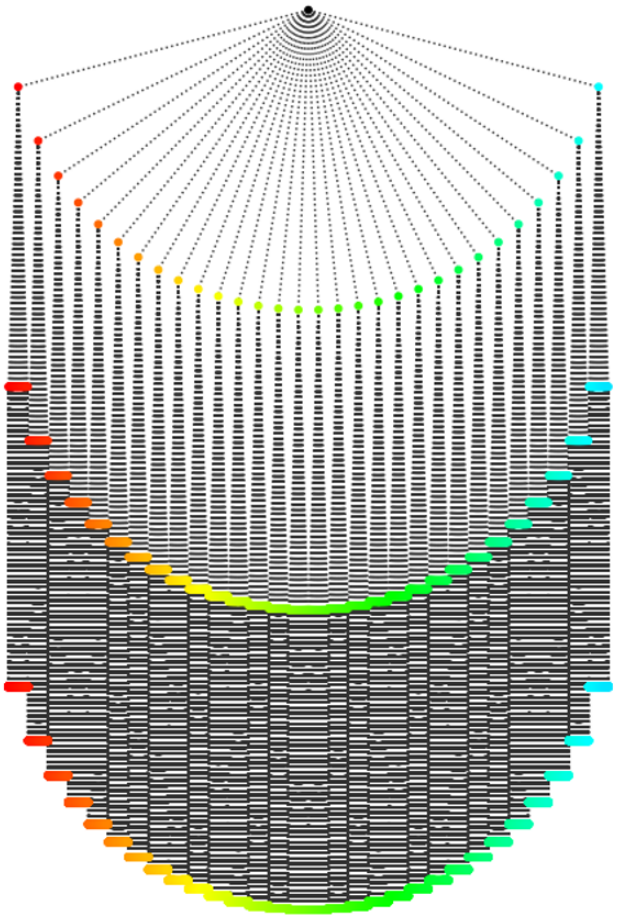


Figure 6: Output drawing of a 30-ary tree of height 3. $l_{\min}$ values 26999.64, $l_{\max}$ values 27000.45. The ratio values 1.00002984

5 Maximal Outerplanar Graphs

This thesis addresses two main approaches of ratio minimization for drawings of 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. Furthermore, any maximal outerplanar graph is a 2-tree, therefore the class of maximal outerplanar graphs is a strict subclass of the 2-trees.
At first, the approaches will address the maximal outerplanar graphs since maximal outerplanar graphs are a subclass of maximal SP-graphs. After the analysis regarding the properties of the resulting drawings it will be discussed whether the approach will be suitable to be extended for 2-trees. This section will present two polyline drawing algorithms. The first drawing algorithm takes advantage of the already existing drawing algorithm **??** for k-ary trees, since the weak dual graph of a maximal outerplanar graph inheits a tree structure. The second drawing algorithm for outerplanar graphs will be suited to be extended for 2-trees, since every maximal SP-graph contains a maximal outerplanar graph as a subgraph.

5.1 Properties Of Maximal Outerplanar Graphs

Lemma 2. *A maximal outerplanar graph G inherits triangles as inner faces, except for the outerface.*

Lemma 3. *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^* values 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 4. *Let G be a maximal outerplanar graph with n vertices and G^* the dual graph excluding the outerface, 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

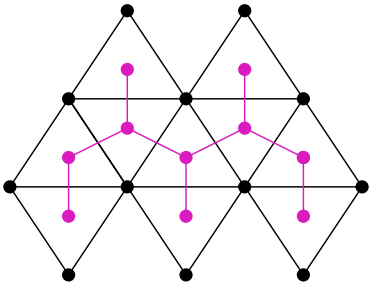
Placing $\mathcal{O}(n)$ vertices in three binary trees connected to a root vertex results in a height of at least $\Omega(\log n)$ due to the k-ary tree height property from Lemma 1. In the worst case, G^* will be a chain of vertices, therefore a rooted tree with height $\mathcal{O}(n)$. □

Lemma 5. *A maximal outerplanar graph G can be extended to a maximal outerplanar supergraph G^+.*

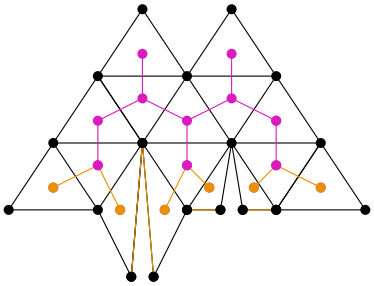
Proof. The vertex insertion works analougously to the recursive definition of a 2-tree. A new vertex can be added to G by adding a new vertex v_f in the dual graph G^* so that the degree of G^* is still at most 3. The newly created face f must lie on the outerface and must be a triangle. Otherwise, the outerplanarity property is destroyed. □

Observation 1. *Let G be a maximal outerplanar graph G with a straight-line drawing Γ_G and a ratio r_G. When G is extended to a maximal outerplanar supergraph G^+, the ratio r_{G^+} increases regarding Γ_{G^+} based on Γ_G.*

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, as illustrated in the following figures:



(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 7: Illustration of area restriction for dense maximal outerplanar graphs

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

5.2 Drawing Algorithm For Complete Outerplanar Graphs With One Bend

The first approach of a drawing algorithm addresses a ratio optimization for dense outerplanar graphs. These class of maximal outerplanar graphs are de-

scribed with help of properties regarding the weak dual graph in the following way.

Definition 1. *A maximal outerplanar graph 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 of height $h-1$*

A given maximal outerplanar graph can be drawn by using a drawing algorithm for its weak dual graph. In section 4, the k-ary tree drawing algorithm produces a straight-line drawing in $\mathcal{O}(n^2 \log n)$ area with a ratio of $1+\varepsilon, \varepsilon > 0$. The drawing algorithm **??** 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 3-ary tree with the same height.

Algorithm 3: `DrawOuterWeakDual(`G`)`

Input: A complete maximal outerplanar graph G
Output: Straight-line drawing Γ_{G^*} with nearly optimal ratio

```
1 G* ← weak dual graph of G with minimal height
2 h ← height(G*)
3 root ← G*.root
4 Draw(root)
5 Draw_3-ary_Children(root,3^h,1)
6 for v ∈ root.children do
7 |  Draw_2-ary_Children(v,2^(h-1),h-1)
8 return Γ
```

Algorithm 3 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 $\Gamma_{G^*}(v^*)$.
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 planarity since the coordinate of every inner node inherits its unique x value by construction of algorithm 1. This drawing approach is summarized in the following algorithm.

Algorithm 4: `DrawCompleteOuterplanar(`G`)`

Input: Complete outerplanar graph G
Output: Polyline drawing Γ_G with one bend per edge and ratio $\mathcal{O}(\log n)$

$\Gamma \leftarrow$ `DrawOuterWeakDual(`G`)`
$h \leftarrow$ `height(`G^*`)`
for $v^* \in G^*$ **do**
  $x(v^*) \leftarrow 3 \cdot x(v^*)$
`/* Draw the triangle face around the root of` G^* `*/`
$root \leftarrow \Gamma($`$G^*$.root`$)$
$l \leftarrow \Gamma($`$G^*$.root.leftChild`$)$
$m \leftarrow \Gamma($`$G^*$.root.middleChild`$)$
$r \leftarrow \Gamma($`$G^*$.root.rightChild`$)$
$v_1 \leftarrow \Gamma$`.PlaceVertex(`$x(root), y(root) + h \cdot 3^h$`)`
$v_2 \leftarrow \Gamma$`.PlaceVertex(`$\frac{x(root)+x(l)+x(m)}{3}, \frac{y(root)+y(l)+y(m)}{3}$`)`
$v_3 \leftarrow \Gamma$`.PlaceVertex(`$\frac{x(root)+x(m)+x(r)}{3}, \frac{y(root)+y(m)+y(r)}{3}$`)`
Γ`.DrawLineSegment(`v_1, v_2`)`
Γ`.DrawLineSegment(`v_1, v_3`)`
Γ`.DrawLineSegment(`v_3, v_2`)`
`/* Iterate over the height to draw the remaining vertices of` G `*/`
for $i \in [1..h]$ **do**
  for $v^* \in G^*$ *with height* i **do**
    $b_l \leftarrow \Gamma$`.PlaceBendPoint(`$x(v^*) - 1, y(v^*)$`)`
    $b_r \leftarrow \Gamma$`.PlaceBendPoint(`$x(v^*) + 1, y(v^*)$`)`
    $v \leftarrow \Gamma$`.PlaceVertex(`$x(v^*), y(v^*) - 1$`)`
    $e^* \leftarrow (v^*,$`parent(`v^*`))`
    $e \leftarrow (v_1, v_2)$ edge intersecting e^* `// w.l.o.g.` v_1 `ordered left of` v_2
    Γ`.DrawLineSegment(`v, b_l`)`
    Γ`.DrawLineSegment(`b_l, v_1`)`
    Γ`.DrawLineSegment(`v, b_r`)`
    Γ`.DrawLineSegment(`b_r, v_2`)`
Γ`.delete(`G^*`)`
return Γ

5.2.1 Analysis

Theorem 2. *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`}. By construction it holds, that v_2 and v_3 partition three binary subtrees rooted at the children of the root of G^* by their x coordinate. 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 3-ary 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 **??** 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 $\mathcal{O}(\log n)$. □

5.2.2 Example Drawing

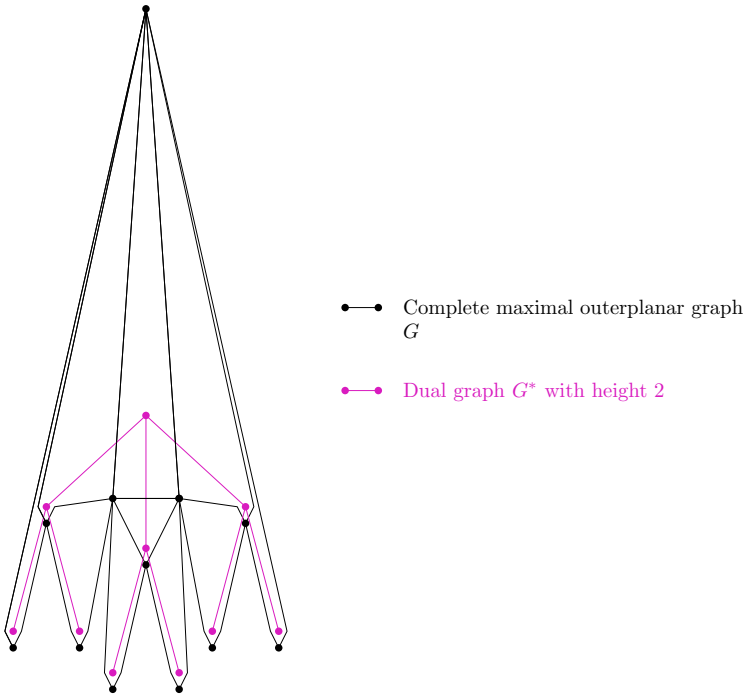


Figure 8: Polyline drawing of a complete outerplanar graph G. The dual graph G^* inherits a height of 2.

5.2.3 Limitations

As addressed by Observation 1, this algorithm works fine for a complete maximal outerplanar graph G since then, the height of G^* is bound by $\mathcal{O}(\log n)$. On the other hand, when G is a loose maximal outerplanar graph, a height of G^* might be of linear size and therefore, the drawing algorithm is not improving the ratio contrary to a straight-line drawing. 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.

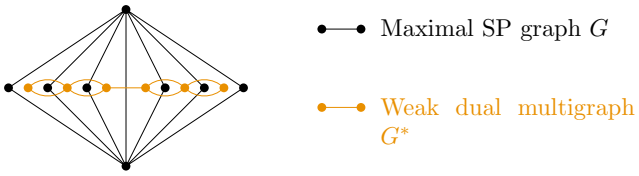


Figure 9: A maximal SP graph with its weak dual multigraph, colored in orange

The approach of drawing the weak dual graph at first followed by the outerplanar graph serves as an idea for a *layered drawing*. 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 the weak dual graph is not suitable for 2-trees, the *tree decomposition* of an SP-graph will serve as a guidance tool for the sequence of vertices drawn by a layering algorithm. The drawing algorithm will use the *SPQR tree* derived from a tree decomposition.
When analyzing upper bounds for the ratio of a drawing for a maximal outerplanar graph G', it is crucial to take a look at the tree structure of its tree decomposition. The following definition describes the difference of a k-ary tree T to being a complete k-ary tree. This description of k-ary tree partitionings is very similar to a tree decomposition and will help during the analysis of a maximal outerplanar graph G with its tree decomposition, since any tree decomposition is a k-ary tree.

5.3 Partitioning A k-ary Tree In Complete k-ary Subtrees

Definition 2. *A* partitioning $P = (\mathcal{P}, \mathcal{W})$ *of a k-ary 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 k-ary 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.*

Let T be a k-ary tree with n vertices. Then, for a partitioning P covering all the vertices of T, the following holds:

1. The amount of partitions of size $\mathcal{O}(1)$ is bound by $\mathcal{O}(n)$
2. The amount of partitions of size $\mathcal{O}(\log n)$ is bound by $\mathcal{O}\left(\frac{n}{\log n}\right)$
3. The amount of partitions of size $\mathcal{O}(\sqrt{n})$ is bound by $\mathcal{O}(\sqrt{n})$
4. The amount of partitions of size $\mathcal{O}(n)$ is bound by $\mathcal{O}(1)$

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 k-ary tree with n vertices results in a legitimate k-ary tree with height $\mathcal{O}(n)$. The minimal partitioning of this example consists of $\mathcal{O}(n)$ partitions in total.

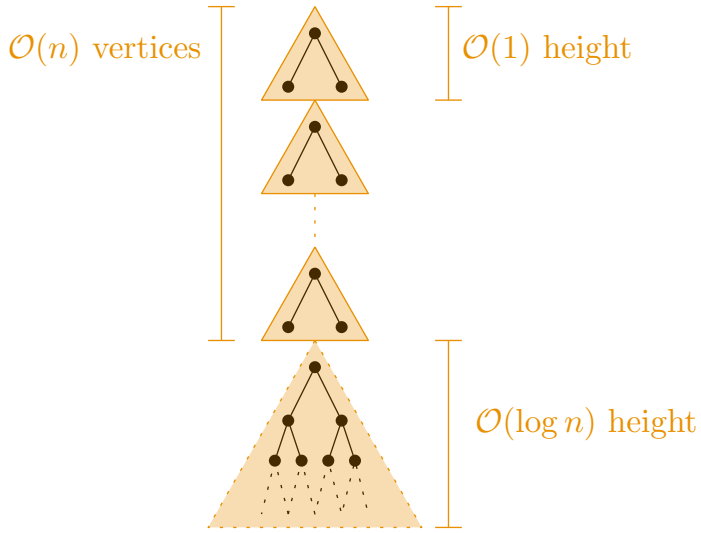


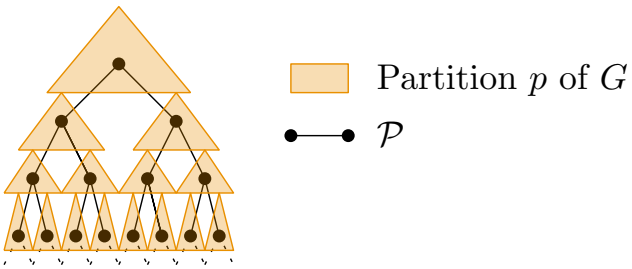
Figure 10: A k-ary tree of $\mathcal{O}(n)$ height partitioned in complete k-ary subtrees (in orange)

Lemma 6. *The height of* G *interacts with the minimal partitioning* P *of* G *in the following way:*

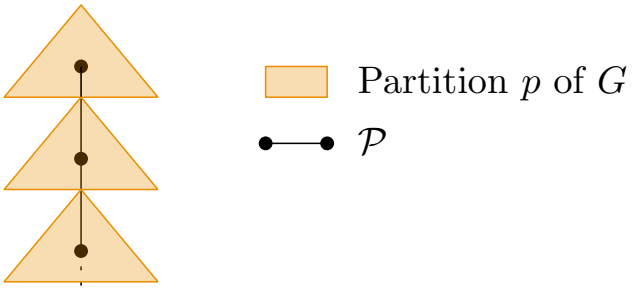
1. *If* P *consists of* $\mathcal{O}(n)$ *many partitions of constant size, the height of* G *lies between* $\Omega(\log n)$ *and* $\mathcal{O}(n)$
2. *If* P *consists of* $\mathcal{O}\left(\frac{n}{\log n}\right)$ *partitions of size* $\mathcal{O}(\log n)$*, then the height of* G *lies between* $\Omega\left(\log\left(\frac{n}{\log n}\right) \cdot \log\log n\right)$ *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 bound by $\mathcal{O}(\log n)$.*

Proof. Let T be a k-ary 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, the following 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 11: Extremal cases regarding the structure of P of a partitioning $\mathcal{P}$

One extremal case covers $\mathcal{P}$ being a complete binary tree. By Lemma 1, for the height investigation of G it suffices $\mathcal{P}$ to be binary since every complete k-ary tree has asymptotically the same height bound. The other extremal case covers a chain of vertices of $\mathcal{P}$. For case 11a, $\mathcal{P}$ is a tree with height $\mathcal{O}(\log |\mathcal{P}|)$ and up to $\mathcal{O}(|\mathcal{P}|)$ height in case 11b.

1. A complete k-ary tree of constant size inherits a constant height. With $|\mathcal{P}| \in \mathcal{O}(n)$, substituting every vertex of $\mathcal{P}$ with complete k-ary trees of constant height does not alter the height asymptotically and the height bounds are valid.

2. A complete k-ary 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 11a and $\mathcal{O}\left(\frac{n}{\log n}\right)$ for case 11b. Therefore, the height of G lies in $\mathcal{O}\left(\log\left(\frac{n}{\log n}\right) \cdot \log \log n\right)$ for case 11a and $\mathcal{O}\left(\frac{n}{\log n} \cdot \log \log n\right)$ for case 11b.

3. If every vertex of $\mathcal{P}$ represents a complete k-ary 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 11a and in $\mathcal{O}(\sqrt{n} \log n)$ for case 11b.

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.

□

5.4 More Properties Of Maximal Outerplanar Graphs

Lemma 7. *Any maximal outerplanar graph G' has a tree decomposition (T', W') such that T' is of degree at most 3.*

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. □

Lemma 8. *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. □

Lemma 9. *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 8.

□

Lemma 10. *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 is a path and of length $\mathcal{O}(\texttt{height}(T))$.*

Let t_p the parent of t_1, t_2 and t_3 and $v \in w_p$. Assume, that $v \in w_1, w_2, w_3$. Since the bags are of size three, there would be a vertex $v' \in w_p$ such that the subtree of T containing v, v' contains more than two vertices, contradicting Lemma 8. Then, the subtree T containing v is of degree 2 and therefore a list.

Definition 3. *When traversing a tree decomposition $(T, W)_G$ 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 11. *Let G' be a maximal outerplanar graph and (T', W') its tree decomposition with $h_{T'}$ as height of T'. When using DFS as graph traversal for a drawing algorithm of G', a vertex v' of G' is active for at least $\Omega(h_{t'})$ steps.*

Proof. This follows directly by Lemma 10. □

Lemma 12. *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 11, $T'_{v'}$ is a chain of vertices. Since G' is maximal outerplanar, by Lemma 8 it holds that the amount of occurences of any other vertex $w' \in V(G') \setminus \{v'\}$ is bound by 2. Since G' is complete, the height of T' is bound by $\mathcal{O}(\log n')$.

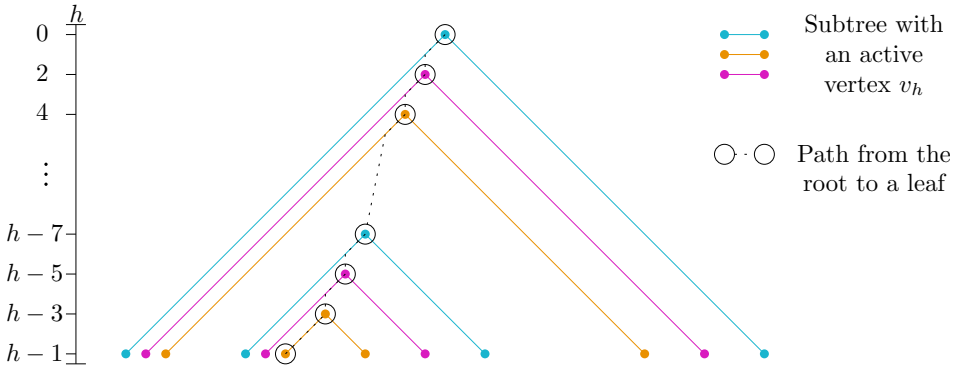


Figure 12: A complete binary tree with up to $\mathcal{O}(\log n)$ active vertices for a DFS path

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 13. *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 bound 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 6, 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. Analoguously to the proof of Lemma 12, traversing a path from the root of T to any leaf will set up to $\mathcal{O}(\log^2 n')$ vertices active. □

5.5 Drawing Algorithm For Maximal Outerplanar Graphs With Two Bends

Based on the fact that a layering induced by the drawing algorithm 4 did improve the ratio for dense maximal outerplanar graphs, a new approach will create a box drawing with the layering property.
In order to draw a maximal outerplanar graph G, its tree decomposition will be transferred to an $SPQR$ tree.

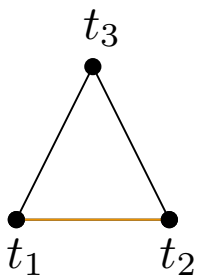
Lemma 14. *A maximal SP-graph inherits a treewidth of 2.*

Proof. This follows by the recursive definition of maximal SP-graphs. The K_3 consists of three vertices and in its tree decomposition (T, W), T consists of one vertex, its bag of the three vertices defining the K_3. Adding a vertex v adjacent to two already adjacent vertices v_1, v_2, a new vertex t is added to T with $\{v, v_1, v_2\}$ in its bag in W. Therefore, every bag contains exactly three vertices of G and the treewidth of a maximal SP-graph values 2. □

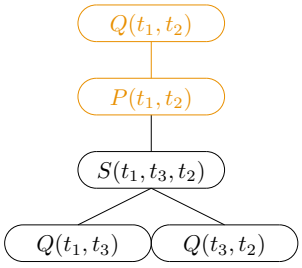
Lemma 15. *There exists a function $f : (T, W) \to \mathcal{T}$ which derives an $SPQR$ tree out of a tree decomposition of a maximal SP-graph G with the following properties:*

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 13a

Figure 13: A triangle illustrated in figure 13a and its $SPQR$ tree in figure 13b. 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 $\mathcal{T}$:

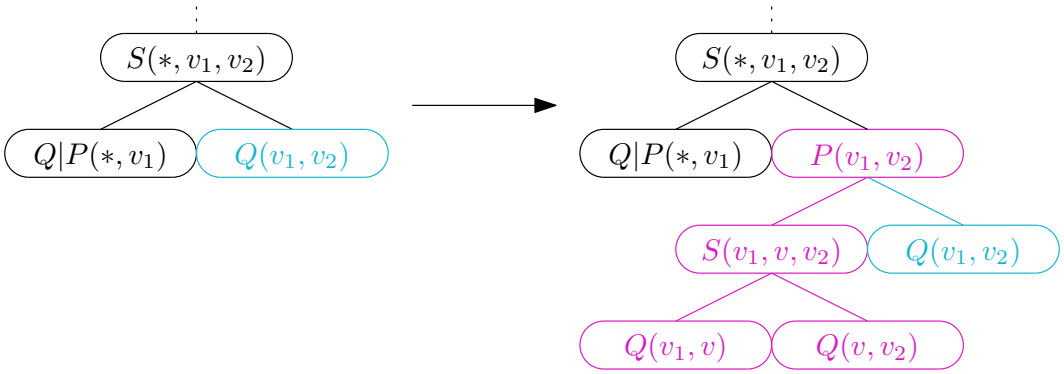


Figure 14: Insertion of a new vertex v when $Q(v_1, v_2$ is not attached to a P node)

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:

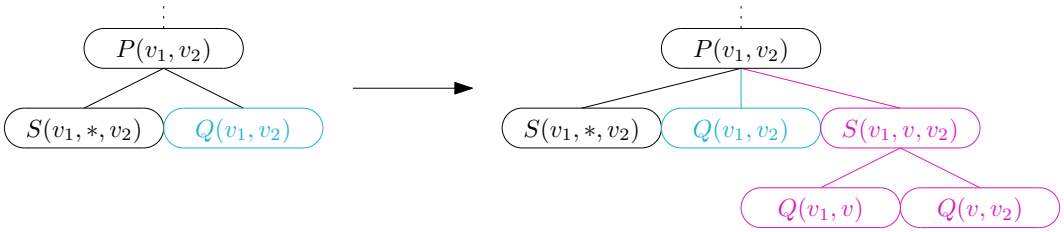


Figure 15: Insertion of a new vertex v when $Q(v_1, v_2$ is not attached to a P node)

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. □

The drawing algorithm will transfer a tree decomposition (T', W') of a maximal outerplanar graph G' to an $SPQR$ tree $\mathcal{T}'$. Afterwards, the algorithm

explores $\mathcal{T}'$ by DFS starting at the root node. This is achieved by a stack implementation in the pseudocode. For a vertex $t' \in V(\mathcal{T}')$, the child of t' which roots the subtree of minimal height will always be prioritized during the DFS. 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 5: `DrawSPQR`($\mathcal{T}$)

Input: $SPQR$-tree $\mathcal{T}$ of a graph G
Output: Box drawing $\mathcal{B}_G$

```
1  Stack stack
2  SPQRVerticesDone ← ∅
3  v ← T.root stack.push(T.root)
4  column ← 0
5  δ ← pairwise distance between layers
6  Lower Layer l_low ← 0
7  Upper Layer l_up ← l_low + δ
8  B.DrawBox(q_1, l_up, column)
9  B.DrawBox(q_2, l_low, column)
10 ActiveVertices ← {q_1, q_2}
11 while VerticesDone ≠ V(T) do
12     v ← stack.pop
13     List children ← v.children
14     children.SortByHeightOfSubtreesDescending
15     while children ≠ ∅ 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 ← column + 1
26         for v ∈ ActiveVertices do
27             B.extendBox(v)
28         l_up ← layer(p_1)
29         l_low ← layer(p_2)
30     else if v is S node then ;          // Serialization S(s_1, s_2, s_3)
31
32         column ← column + 1
33         for v ∈ ActiveVertices do
34             B.extendBox(v)
35         if ∄ free layer between layer(s_1) and layer(s_3) then
36             l ← B.addLayer(l_up, l_low)
37         else
38             l ← free layer between layer(s_1) and layer(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 6: `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** $\Gamma_{G'}$

5.5.1 Analysis

Algorithm 6 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 5 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 3. *Let G' be a complete maximal outerplanar graph. The drawing algorithm 6 produces a polyline drawing $\Gamma_{G'}$ on area $\mathcal{O}(n'^2 \log n')$ with two bends per edge and ratio $r \in \mathcal{O}(\log n')$, when the minimal distance between two layers is set to n'.*

Proof. Consider the drawing algorithm 5, 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 12, there are up to $\mathcal{O}(\log n)$ 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 asympotitcally bound 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 bound by $\mathcal{O}(n \log n)$. Since for every edge of G' there exists a column in $\mathcal{B}_{G'}$, the width is bound 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 $\Gamma_{G'}$, 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 6 inherits two bends per edge and has asymptotically the same area bound as the box drawing of algorithm 5. The longest edge $l_{\max}$ spans the width and the height and its length is bound 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 4. *Let G' be a maximal outerplanar graph. The drawing algorithm 6 produces a polyline drawing $\Gamma_{G'}$ 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 13, up to $\mathcal{O}(\log^2 n')$ are active simultaneously when traversing the tree decomposition of G' with DFS and prioritizing the subtrees of minimal height. Analouguously to the proof of Theorem 3, 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 bound by $\mathcal{O}(\log^2 n')$. □

Theorem 5. *There exists a class of maximal outerplanar graphs for which every graph 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 bound by c, the height of the regarding complete k-ary tree is bound by $\mathcal{O}(1)$.
When $\mathcal{P}$ is a chain of vertices, the drawing algorithm 6 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.

□

5.5.2 Preserving Outerplanarity

Theorem 6. *There exists an alternation of Algorithm 5 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 9 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.

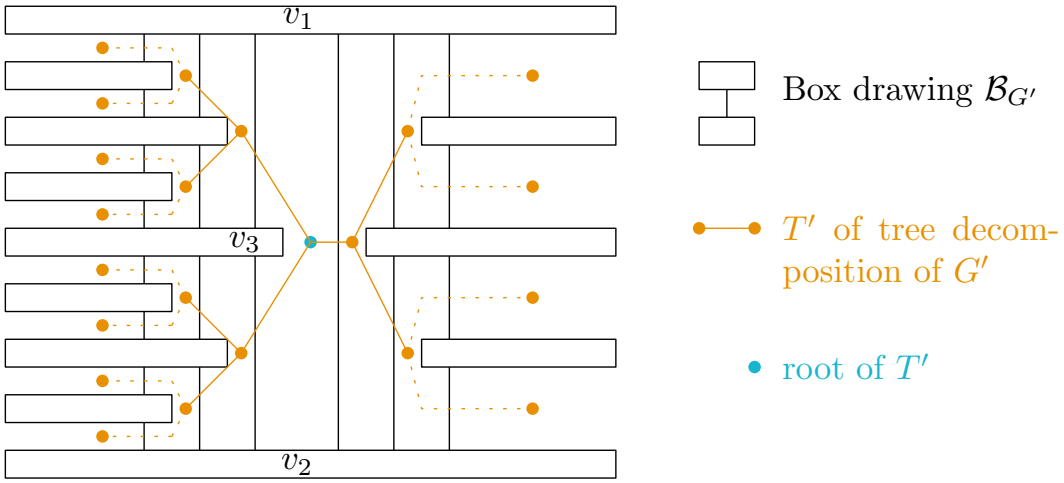


Figure 16: Scheme to draw a complete maximal outerplanar graph with preserving the outerplanarity property

This way, every box lies on the outerface of $\mathcal{B}_{G'}$ and every vertex lies on the outerface in the resulting polyline drawing $\Gamma_{G'}$. The runtime, area consumption and ratio stay the same. □

6 Series-Parallel Graphs

6.1 Properties Of Maximal Series Parallel Graphs

Lemma 16. *For every 2-tree G, there exists a maximal outerplanar subgraph G' that is larger than every other maximal outerplanar subgraph.*

Proof. Consider the tree decomposition (T_G, W_G) of G, T inheriting height h_T. 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 up to h_T and their respective bags which share exactly one vertex. Then, recursively pick the two children which root the subtrees with the maximum height and add them to T' with their respective bags to W' which share exactly one vertex. This procedure terminates at the leaves of T.
Every vertex of the resulting T' is of degree maximum 3 and by Lemma 7, the graph G' induced by T is outerplanar. Since, during this procedure, the nodes with the maximum subtrees are chosen and the properties described in Lemma 9 hold, there exists no other maximal outerplanar subgraph of G which is larger. □

The tree decomposition of a 2-tree is used to derive an according $SPQR$ tree for the implementation of the drawing algorithm.

Lemma 17. *There exists a function f which derives an $SPQR$-tree $\mathcal{T}$ from a given tree decomposition (T, W).*

Proof. Every vertex of T refers to a triangle in G. Start at the root of T. Every Q node refers to an edge of G and is connected to either a P or a S node. A triangle consists of one P, one S node and three Q nodes. The following figure demonstrates the possible modifications in a $SPQR$-tree when a vertex is added to an existing 2-tree according to its recursive definition. In (T, W), any child t_c of t_p is added to the $SQPR$-tree $\mathcal{T}$ accordingly since w_c and w_t share exactly two vertices v_1, v_2. □

6.2 Drawing Algorithm For Maximal Series Parallel Graphs With Two Bends

The tree decomposition (T, W) of a 2-tree G will be evaluated for its largest maximal outerplanar graph with tree decomposition (T', W'). $\tilde{T}$ is defined as the difference of T and T' and is a set of subtrees of T.
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.

6.2.1 Example drawing

7 Related Work

- 3-tree statement

8 Future Work

- compress the area for any k-ary tree with optimal ratio
- Is log square a sharp upper bound
- implementation of maximal SP graph drawing
- find a new approach for 3-trees

9 Acknowledgements

I would like to thank Prof. Michael Kaufmann for instructing this final thesis. Further I appreciate helpful discussions with Dr. Henry Förster. I want to thank especially my family for their patience and encouragement.

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] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] *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).

[11] *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).

[12] *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).

[13] 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.

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

[15] 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.