Eberhard Karls Universität Tübingen
Wilhelm Schickard Institut Tübingen

Fachbereich Informatik

On Maximizing the Euclidian Distance Between Vertices In Drawings Of Certain Graph Classes (Working Title)

Arbeitsbereich Algorithmik

zur Erlangung des akademischen Grades
Master of Science

Autor:	Benjamin Çoban MatNr. 3526251
Version vom:	15. März, 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

Thoughts:

1. General graph drawing introduction
2. Graph Drawing Symposium introduction
3. why is maximizing distances between vertices of interest?
4. Figures of readability
5. Thesis structure

2 Preliminaries

2.1 Definitions And Terminology

A *graph* $G = (V, E)$ is a tuple consisting of two sets - the set of vertices and the set of edges. 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* [3, 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 [3, P. 1170]. A graph is *biconnected* if the removal of any vertex still leaves the graph connected [5, P. 224]. A graph is *simple* if and only 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* [3, P. 1172].
A graph is *planar* if and only if there exists a crossing-free representation in the plane [3, 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* [4, 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^*

[4, P. 103]

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 [3, 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 [5, 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. [2, 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) [6, P. 4].
A drawing whose minimum enclosing box has width w and height h is called a $w \times h$-drawing and inherits area $w \cdot h$ [2, 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

2.4 Graph Classes

2.4.1 Tree

A graph T is called *tree* if and only if it is connected, acyclic and undirected [3, 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. [3, 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 added so while maintaining a SP-graph. [2, P. 143ff]
A *2-tree* is a recursively defined graph with at least three vertices. If $n = 3$, then the 2-tree is the complete graph K_3. If $n > 3$, start with a K_3 and every vertex added is adjacent to exactly two 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

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 (TW) 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 $v \in W_{t'}, t' \in V_{T'}$

[4, 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 [4, P. 321].

3 Initial Situation

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 a feasible solution in a reduction to an instance of a tree. In fact, this effect will occur in the subsequent section of this thesis.

4.1 The Drawing Algorithm

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

Proof.

$$n = \sum_{i=0}^{h} k^i = \frac{k^{h+1}-1}{k-1} \tag{2}$$

$$\Leftrightarrow h = \log_k((k-1)n+1) - 1 \tag{3}$$

$$= \frac{\log((k-1)n+1)}{\log k} - 1 \tag{4}$$

$$\Rightarrow h \in O(\log n) \tag{5}$$

□

Theorem 1. *Every 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.

graphics/k-ary_tree_algorithm_construction.pdf

Figure 1: 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:

$$\begin{aligned}\underbrace{2 \cdot \sum_{j=i}^{h-1} k^{h-j-1}}_{\text{drawn from both columns}} &= 2 \cdot \sum_{z=0}^{h-i-1} h^z &(6)\\ &= 2 \cdot \frac{k^{h-i}-1}{k-1} < 2 \cdot k^{h-i} &(7)\end{aligned}$$

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 $2 \cdot \sum_{i=0}^{h} k^i = 2 \cdot \frac{k^{i+1}-1}{k-1} \in \mathcal{O}(n)$, 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 algorithm sums up the approach described above.

Algorithm 1: Drawing algorithm for k-ary trees

```
Input: complete k-ary tree $T$,$k$
Output: Planar 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($root(T), r, h$)
```

Algorithm 2: `Draw`(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      $C \leftarrow v$.DrawCircle$(r)$
       /* 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          $X \leftarrow$ Column$(x(v_i))$
           /* Identify the column at position $x(v_i)$                */
10         $s \leftarrow$ Intersection$(I, X)$
           /* Calculate the intersection of the circle $C$ and the
              column $X$                                              */
11         $y(v_i) \leftarrow round(y(s))$
12         DrawStraightLine$(v, v_i)$
13         Draw$(v_i, r, h)$
```

4.2 Example Drawings

5 Series-Parallel Graphs

6 Summary

7 Extensional Work

8 Future Work

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] 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`.

[3] 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`.

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

[5] 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.

[6] 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`.