Michael Kaufmann
Dorothea Wagner (Eds.)

LNCS 4372

# Graph Drawing

14th International Symposium, GD 2006
Karlsruhe, Germany, September 2006
Revised Papers

gd 06

Springer

Michael Kaufmann   Dorothea Wagner (Eds.)

# Graph Drawing

14th International Symposium, GD 2006
Karlsruhe, Germany, September 18-20, 2006
Revised Papers

Springer

Volume Editors

Michael Kaufmann
Universität Tübingen
Wilhelm-Schickard-Insititut für Informatik
72076 Tübingen, Germany
E-mail: mk@informatik.uni-tuebingen.de

Dorothea Wagner
Universität Karlsruhe
Fakultät Informatik
76128 Karlsruhe, Germany
E-mail: wagner@ira.uka.de

Library of Congress Control Number: 2007920270

CR Subject Classification (1998): G.2, F.2, I.3, E.1

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-70903-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-70903-9 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12020614 06/3142 5 4 3 2 1 0

# Preface

The 14th International Symposium on Graph Drawing (GD 2006) was held in Karlsruhe, Germany, during September 17-20, 2006. The conference attracted 108 participants from 18 countries.

In response to the call for papers, the Program Committee received 91 submissions by (co)-authors from 31 countries. At least three Program Committee members reviewed each submission, and after an intensive evaluation phase, the Program Committee selected 33 long papers, 5 short papers, as well as one system demo and 3 posters. The authors received extensive comments and excerpts from the discussion on their papers. The best paper award went to Jan Arne Telle and David R. Wood for their paper with the title "Planar Decompositions and the Crossing Number of Graphs with an Excluded Minor."

The Chairs of the conference invited two distinguished lecturers: Emo Welzl from the ETH in Zürich gave a very nice and inspiring presentation on "The Number of Crossing-Free Configurations in Planar Point-Sets." Oliver Deussen from the Universität Konstanz talked about "The Algorithmic Beauty of Virtual Nature." Abstracts of the two invited lectures can also be found in this volume.

An ambitious graph drawing contest took place this year under the guidance of Christian Duncan, who also included a report in this volume. In the software exhibition the industrial sponsors of the conference, Tom Sawyer Software, yWorks GmbH and ILOG Inc., had the chance to provide a closer look at the products and developments. Furthermore, Ralph Schunk (Universität zu Köln) presented his software tool "CUPE - CUBIC Pathway Editor."

Many people contributed to the success of GD 2006. First of all, the authors of submitted contributions deserve special thanks, as well as the members of the Program Committee for the careful work and the extensive discussions. For the organizational aspects, we are especially grateful to Lilian Beckert and Michael Baur, who kept the meeting running and took care of all the big and small things which have to be considered during such a meeting.

The conference received considerable support from the hosting organization, Universität Karlsruhe (TH), and from the German Research Foundation DFG. Furthermore, we are grateful to our two gold sponsoring partners, Tom Sawyer Software and yWorks GmbH, as well as to the silver sponsor, ILOG Inc.

Next year, the symposium will take place in Sydney, Australia, and will be hosted by Seok-Hee Hong and Takao Nishizeki at the University of Sydney.

October 2006

Michael Kaufmann
Dorothea Wagner

# Organization

## Steering Committee

| | |
|---|---|
| Franz J. Brandenburg | Universität Passau |
| Giuseppe Di Battista | Università degli Studi Roma Tre |
| Peter Eades | NICTA, University of Sydney |
| Hubert de Fraysseix | Centre d'Analyse et de Mathematique Sociale |
| Patrick Healy | University of Limerick |
| Seok-Hee Hong | NICTA, University of Sydney |
| Michael Kaufmann | Universität Tübingen |
| Takao Nishizeki | Tohoku University |
| Pierre Rosenstiehl | Centre National de la Recherche Scientifique |
| Roberto Tamassia | Brown University |
| Ioannis G. Tollis | University of Crete |
| Dorothea Wagner | Universität Karlsruhe |
| Sue Whitesides | McGill University |

## Program Committee

| | |
|---|---|
| Walter Didimo | Università degli Studi di Perugia |
| Vida Dujmović | McGill University |
| David Eppstein | University of California |
| Patrick Healy | University of Limerick |
| Michael Jünger | Universität zu Köln |
| Michael Kaufmann | Universität Tübingen (*Co-chair*) |
| Stephen G. Kobourov | University of Arizona |
| Yehuda Koren | AT&T Labs |
| Xuemin Lin | University of New South Wales |
| Guy Melançon | LIRMM Montpellier |
| Takao Nishizeki | Tohoku University |
| Maurizio Patrignani | Università degli Studi Roma Tre |
| Bettina Speckmann | TU Eindhoven |
| Géza Tóth | Hungarian Academy of Sciences |
| Dorothea Wagner | Universität Karlsruhe (*Co-chair*) |
| Alexander Wolff | Universität Karlsruhe |
| David R. Wood | Universitat Politècnica de Catalunya |

## Organizing Committee

| | |
|---|---|
| Michael Baur | Universität Karlsruhe |
| Lilian Beckert | Universität Karlsruhe |

|  |  |
|---|---|
| Michael Kaufmann | Universität Tübingen (*Co-chair*) |
| Martin Siebenhaller | Universität Tübingen |
| Dorothea Wagner | Universität Karlsruhe (*Co-chair*) |

## Contest Committee

|  |  |
|---|---|
| Christian Duncan | Louisiana Tech University (*Chair*) |
| Gunnar Klau | Freie Universität Berlin |
| Stephen G. Kobourov | University of Arizona |
| Georg Sander | ILOG Deutschland GmbH |

## External Referees

Greg Aloupis
Radoslav Andreev
David Auber
Janos Barat
Michael Baur
Marc Benkert
Carla Binucci
Nicolas Bonichon
Prosenjit Bose
Christoph Buchheim
Yves Chiricota
Markus Chimani
Marek Chrobak
Sebastien Collette
Sabine Cornelsen
Pier Francesco Cortese
Elias Dahlhaus
Maylis Delest
Daniel Delling
Emilio Di Giacomo
Tim Dwyer
Alejandro Estrella-Balderrama
J. Joseph Fowler
Fabrizio Frati
Éric Fusy
Marco Gaertler
Emden Gansner
Markus Geyer
Francesco Giordano
Luca Grilli
Carsten Gutwenger
Robert Görke
Stefan Hachul
Herman Haverkort
Martin Harrigan
Mao Lin Huang
Xiaodi Huang
Takehiro Ito
T.J. Jankun-Kelly
Fabien Jourdan
Bastian Katz
Karsten Klein
Shankar Krishnan
Yoshiyuki Kusakari
Wei Lai
Stefan Langerman
Katharina A. Lehmann
Steffen Mecke
Sascha Meinert
Kazuyuki Miura
Pat Morin
Elena Mumford
Shin-Ichi Nakano
Daren Nestor
Nikola S. Nikolov
Martin Nöllenburg
Merijam Percan
Maurizio Pizzonia
Md. Saidur Rahman
Aimal Tariq Rextin
Adrian Rusu
Falk Schreiber
Michael Schulz
Martin Siebenhaller
Matthew Suderman
Alexandre Tarassov
Csaba D. Tóth
Francesco Trotta
Frank van Ham
Imrich Vrt'o
Nelson Wong
Kai Xu
Qing Zhang
Xiao Zhou

## Sponsoring Institutions

Deutsche
Forschungsgemeinschaft
DFG

Universität Karlsruhe (TH)
Research University • founded 1825

Fakultät für Informatik

yWorks
the diagramming company

Tom Sawyer
SOFTWARE®

ILOG®
Changing the rules of business™

# Table of Contents

# The Number of Triangulations on Planar Point Sets

Emo Welzl

Institute of Theoretical Computer Science
ETH Zurich, Switzerland

**Abstract.** We give a brief account of results concerning the number of triangulations on finite point sets in the plane, both for arbitrary sets and for specific sets such as the $n \times n$ integer lattice.

Given a finite point set $P$ in the plane, a *geometric graph* is a straight line embedded graph with vertex set $P$ where no segment realizing an edge contains points from $P$ other than its endpoints. We are interested in *crossing-free geometric graphs* on a given planar point set, i.e. segments are not allowed to share points other than common endpoints. A maximal crossing-free geometric graph on a point set $P$ is called a *triangulation of P*.

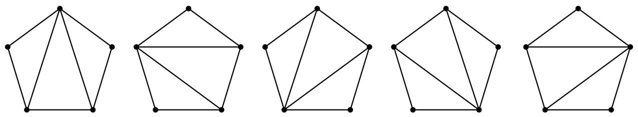


**Fig. 1.** All triangulations of five points in convex position

If, as it is the case in Fig. 1, the points are in convex position, i.e. vertices of a convex polygon, then every triangulation clearly must contain all edges of "its" convex polygon, and we are left with choosing a triangulation for this polygon. Euler was the first to consider how many choices there are for a convex $n$-gon, but it was proven only later that this number is $C_{n-2}$, where $C_m := \frac{1}{m+1}\binom{2m}{m} = \Theta(m^{-3/2}4^m)$, known as the Catalan numbers; (see, e.g., also Pólya's article *On Picture-Writing* [14]).

The example of four points already shows that position matters: Four points in convex position allow two triangulations, while there is only one otherwise. David Avis was perhaps the first to ask what the maximal possible number of triangulations of a general $n$-point set is. An upper bound of $n^{O(n)}$ is easy to obtain, but in 1982 Ajtai, Chvátal, Newborn, and Szemerédi [3] showed that for any set of $n$ points the number of *all* crossing-free geometric graphs is at most $c^n$ for $c = 10^{13}$. The constant in the bound for triangulations has been successively improved [18,6,17,16]. The currently best bound of $43^n$ [15] is derived via considering random triangulations of finite point sets in general position with triangular convex hull. For a random non-extreme vertex in such a random triangulation one can show that it has degree 3 with probability at least $\frac{1}{43}$ ("random" refers here always to "uniformly at random"); interestingly, this yields the claimed bound on the number of triangulations.

Note that every crossing-free geometric graph is contained in some triangulation and a triangulation has at most $3n-6$ edges. Therefore, any bound of the form $\tau^n$ for triangulations yields a bound of $2^{3n-6}\tau^n < (8\tau)^n$ for all crossing-free geometric graphs; so this stands at $344^n$.

The $43^n$-bound is probably far from optimal. On the other end [2] show that there are sets of $n$ points with as many as $\Omega(8.48^n)$ triangulations.

*Lattice triangulations.* The extremal properties for general point sets are still wide open, but even for "simple" concrete point sets the number of triangulations seems hard to analyze. One such example is the $n \times n$ integer lattice $L_{n\times n} := \{0,1,\ldots,n\}^2$ (with $(n+1)^2$ points); see [9] for a brief discussion of problems where lattice triangulations occur.

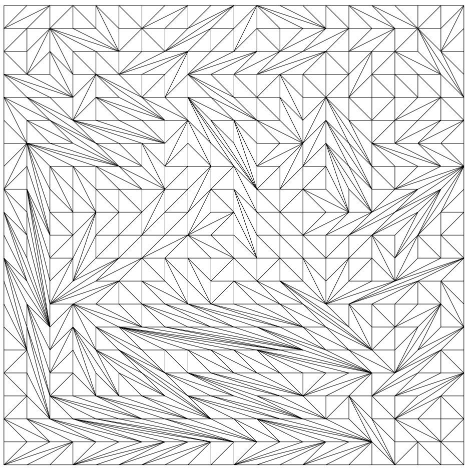


**Fig. 2.** Triangulated $20 \times 20$ lattice

The number of triangulations of $L_{n\times n}$ was first shown in [13] to be at most $64^{n^2}$ (this is, in fact, more than the general upper bound known today). Anclin [4] improved that to $8^{n^2}$ with the following argument: First, it is easy to show that in every triangulation each edge contains exactly one of the half-integral points $(\frac{1}{2}\{0,1,\ldots,2n\})^2 \setminus L_{n\times n}$ as its midpoint, and, conversely, every such half-integral point lies on one of the edges. Second, he proves that if we choose the edges through these half-integral points in a row by row and left-to-right fashion, then at each point there are at most two choices compatible with the edges chosen so far. Half-integral points on the boundary leave no choice, so there are at most

$3n^2 - 2n$ binary choices to be made which readily yields the $8^{n^2}$ bound. As one soon realizes, even interior half-integral points often allow only one choice as we get to them, which indicates that the bound over counts. Indeed, [10] argue that the bound can be set to $O(6.86^{n^2})$, the best estimate currently known. The existence of at least $\Omega(4.15^{n^2})$ triangulations on $L_{n \times n}$ was certified in [9].

*Other special point sets.* There are a few special types of configurations other than convex position for which the number of triangulations is known; see Fig. 3 for such sets (we skip formal definitions of the configurations and we ignore polynomial factors in the counting).

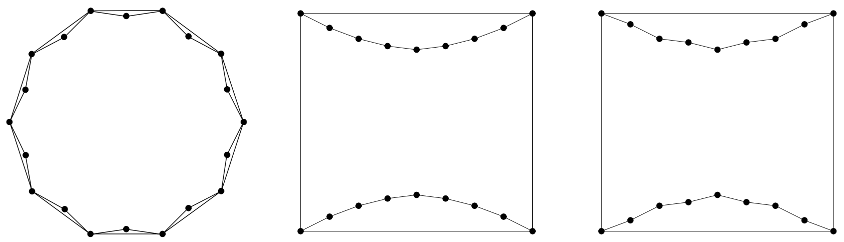


**Fig. 3.** The double circle (20 points), the double chain (18 points), and the double zig-zag chain (18 points). Edges shown are those which have to appear in every triangulation of the set.

The double circle has $\sqrt{12}^n$, $\sqrt{12} \approx 3.4641$, triangulations [8], which is obviously significantly less than for convex position and the smallest number known for $n$ points in general position (no three points on a line). For some time the double chain considered in [7] with $8^n$ triangulations was the set exhibiting the most triangulations known, but then [2] analyzed the double zig-zag chain with $\sqrt{72}^n$, $\sqrt{72} \approx 8.4853$, triangulations.

*Algorithmic aspects.* In [5] it is shown that the set of all triangulations of a point set can be enumerated in time $O(t \cdot \text{poly}(n))$, where $t$ is the number of triangulations. But when it comes to counting, i.e. computing this number $t$, nothing is known at all other than some heuristics [1].

An interesting related question is that of the mixing rate of the random walk on the flip graph of all triangulations of a given point set. In this graph the triangulations represent vertices, and two triangulations are adjacent if we can obtain one from the other by removing one edge and replacing it by another one (an operation called edge-flip). This graph is connected and has diameter $O(n^2)$. A random walk (with some waiting time for technical reasons) will eventually produce a random triangulation, but nothing is known about the time it takes for that to happen—other than the case of points in convex position, where polynomial time mixing has been demonstrated [11,12]. Polynomial time mixing of this random walk would imply polynomial time generation of an approximate random triangulations, and (with help of the result in [15]) polynomial time approximate counting of triangulations.

## References

1. O. Aichholzer, The path of a triangulation, *Proc. 15th Ann. ACM Symp. on Computational Geometry* (1999), 14–23.
2. O. Aichholzer, T. Hackl, H. Krasser, C. Huemer, F. Hurtado, and B. Vogtenhuber, On the number of plane graphs, *Proc. 17th Ann. ACM-SIAM Symp. on Discrete Algorithms* (2006), 504–513.
3. M. Ajtai, V. Chvátal, M. M. Newborn, and E. Szemerédi, Crossing-free subgraphs, *Annals Discrete Math.* **12** (1982), 9–12.
4. E. E. Anclin, An upper bound for the number of planar lattice triangulations, *J. Combinat. Theory, Ser. A* **103** (2003), 383–386.
5. D. Avis and K. Fukuda, Reverse search for enumeration, *Discrete Appl. Math.* **65** (1996), 21–46.
6. M. O. Denny and C. A. Sohler, Encoding a triangulation as a permutation of its point set, *Proc. 9th Canadian Conf. on Computational Geometry* (1997), 39–43.
7. A. García, M. Noy, and J. Tejel, Lower bounds on the number of crossing-free subgraphs of $K_N$, *Comput. Geom. Theory Appl.* **16** (2000), 211–221.
8. F. Hurtado and M. Noy, Counting triangulations of almost-convex polygons, *Ars Combinatorica* **45** (1997), 169–179.
9. V. Kaibel and G. Ziegler, Counting lattice triangulations, *British Combinatorial Surveys* (C.D. Wensley, ed.), Cambridge University Press, 2003.
10. J. Matoušek, P. Valtr, and E. Welzl, On two encodings of lattice triangulations, manuscript (2006).
11. L. McShine and P. Tetali, On the mixing time of the triangulation walk and other Catalan structures, *in:* DIMACS-AMS volume on Randomization Methods in Algorithm Design (Eds. P.M. Pardalos, S. Rajasekaran, and J. Rolim) *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **43** (1998), 147–160.
12. M. Molloy, B. Reed, and W. Steiger, On the mixing rate of the triangulation walk, *in:* DIMACS-AMS volume on Randomization Methods in Algorithm Design (Eds. P.M. Pardalos, S. Rajasekaran, and J. Rolim), *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **43** (1998),179–190.
13. S. Yu. Orevkov, Asymptotic number of triangulations in $\mathbf{Z}^2$, *J. Combinat. Theory, Ser. A* **86** (1999), 200–203.
14. G. Pólya, On picture-writing, *The American Mathematical Monthly* **63** (1956), 689–697. *Also in*: G. L. Alexanderson, *The Random Walks of George Pólya*, MAA, 2000.
15. M. Sharir, E. Welzl, Random triangulations of planar point sets, *Proc. 22nd Ann. ACM Symp. on Computational Geometry* (2006), 273-281.
16. F. Santos and R. Seidel, A better upper bound on the number of triangulations of a planar point set, *J. Combinat. Theory, Ser. A* **102** (2003), 186–193.
17. R. Seidel, On the number of triangulations of planar point sets, *Combinatorica* **18** (1998), 297–299.
18. W. S. Smith, *Studies in Computational Geometry Motivated by Mesh Generation*, Ph. D. Thesis, Princeton University, 1989.

# The Algorithmic Beauty of Digital Nature

Oliver Deussen

University of Constance, Germany
Oliver.Deussen@uni-konstanz.de
http://graphics.uni-konstanz.de

In recent years the development of graphics hardware and efficient rendering algorithms enabled game developers to create large landscapes and render them at interactive rates. However, the shown scenes are still rough approximations that do not reach the complexity of real nature. To obtain sufficient simulations with a degree of realism that comes close to nature, a couple of problems have to be solved. In this extended abstract these challenges are roughly sketched, references are given for further readings.

## 1 Modelling

Creating a good scene requires powerful modeling algorithms at different levels. First a sufficient set of plant models has to be created. Nature is very diverse: modeling the most important plants that are found in Europe requires thousands of different models, and this is why efficient modeling algorithms for plants have to be found. In our software xfrog we combine rule-based modeling methods with procedural elements. This allows us creating a large variety of models efficiently. For an overview of the method I refer to [1]. Figure 1 shows an example of modeling a sunflower. The user combines components to create the model. The components describe parts of the plant or multiplication algorithms.

The plant models then have to be combined to a virtual landscape. At this stage other modeling programs are needed that enables the user to edit a huge number of plant objects. The plants interact with each other; complex patterns arise due to seeding mechanisms and the fight for resources. Sometimes, the development of a landscape has to be simulated. We developed a series of modeling tools ranging from painting interfaces for plant distributions up to complex ecological simulation algorithms. Chapter Three of [1] gives an overview of possible methods.

## 2 Level-of-Detail Modeling

Having modeled plant models and positions, we end up with very complex geometry even for a small landscape. A single tree model may consist of millions of surfaces, a complete forest of billions or trillions. Efficient level-of-detail algorithms are necessary to obtain interactive rendering of these scenes. Unfortunately, standard algorithms for computer graphics fail here since a plant consists

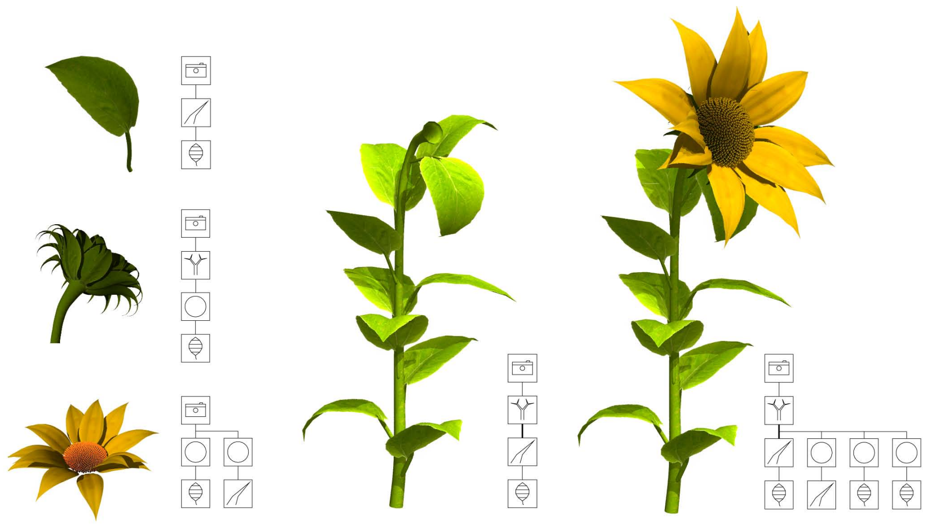


**Fig. 1.** Construction of a sunflower model

of many isolated triangles while conventional surfaces in graphics typically are smooth. Therefore, we developed special Level-of-Detail algorithms for plants that work with pointsets or so-called billboard clouds. In Figure 2 a representation of a complex plant by sets of billboards is shown. If standard clustering techniques are used to determine proper billboards artifacts occur. An improved method uses clustering together with semantic information about the model (see [2]).

## 3 Rendering

Rendering the models is also an interesting task. The interaction of light with the plant surfaces and especially leaves is not trivial. Subsurface scattering and different optical properties of plant tissues require to adapt standard rendering techniques to these models. This is done by modeling the leaves as layered surfaces with different optical properties. Doing so, we are able to simulate the optical properties (Figure 3(a)).

## 4 Non-photorealistic Rendering

For many applications photorealistic rendering is not optimal. This is why we also focussed on abstract representations of plants. This encompasses pen-and-ink illustrations as well as watercolor simulations. We combine photorealistic and abstract representations in order to visualize existing and planned elements in landscapes. This is used in gardening, landscaping, and architecture.

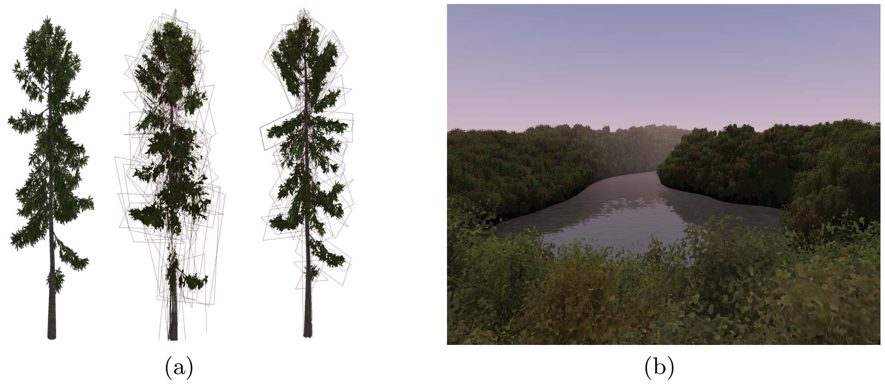


**Fig. 2.** a) Approximation of a tree model by sets of billboards. Left the original model, in the middle a standard k-means clustering, on the right an improved clustering; b) Landscape with billboard plants.

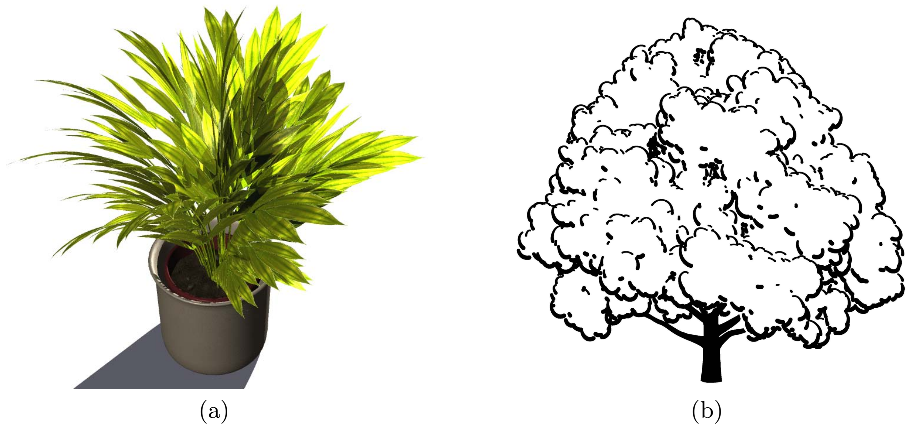


**Fig. 3.** a) Rendering a plant model with translucency; b) Non-photorealistic rendering

## References

1. Deussen, O., Lintermann, B.: Digital desing of Nature - computer generated plants and organics. Springer-Verlag (2005)
2. Behrendt, S., Colditz, C., Franzke, O., Kopf, J., Deussen, O.: Realistic real-time rendering of landscapes using billboard clouds. Computer Graphics Forum **24** (2005)

# Integrating Edge Routing into Force-Directed Layout

Tim Dwyer, Kim Marriott, and Michael Wybrow

Clayton School of Information Technology,
Monash University, Clayton, Victoria 3800, Australia
{tdwyer,marriott,mwybrow}@csse.monash.edu.au

**Abstract.** The typical use of force-directed layout is to create organic-looking, straight-edge drawings of large graphs while combinatorial techniques are generally preferred for high-quality layout of small to medium sized graphs. In this paper we integrate edge-routing techniques into a force-directed layout method based on constrained stress majorisation. Our basic procedure takes an initial layout for the graph, including polyline paths for the edges, and improves this layout by moving the nodes to reduce stress and moving edge bend points to straighten the edges and reduce their overall length. Separation constraints between nodes and edge bend points are used to ensure that nodes do not overlap edges or other nodes and that no additional edge crossings are introduced.

**Keywords:** graph layout, constrained optimisation, force-directed layout, edge routing.

## 1 Introduction

Researchers and practitioners in various fields have been arranging diagrams automatically using physical "mass-and-spring" models since at least 1965 [1]. Typically, the objective of such *force-directed* techniques is to minimise the difference between actual and ideal separation of nodes [2], for example:

$$stress(X) = \sum_{i<j} w_{ij}(||X_i - X_j|| - d_{ij})^2 \tag{1}$$

where $w_{ij}$ is typically $\frac{1}{{d_{ij}}^2}$, $X_i$ gives the placement in two or more dimensions of the $i^{th}$ node and $d_{ij}$ is the ideal distance between nodes $i$ and $j$ based on the graph path length between them.

One of the attractive qualities of such physical models is that the physical analogy can be easily extended to include additional aesthetic requirements by adding additional forces between objects in the drawing. For example, to preserve the edge crossings in an initial layout Bertault [3] added a repulsive force between nodes and their projection points on edges, thus preventing nodes from passing through edges during layout. This method was only applicable to layouts with straight-line edges and point-size nodes. Brandes et al. [4] and later Finkel and

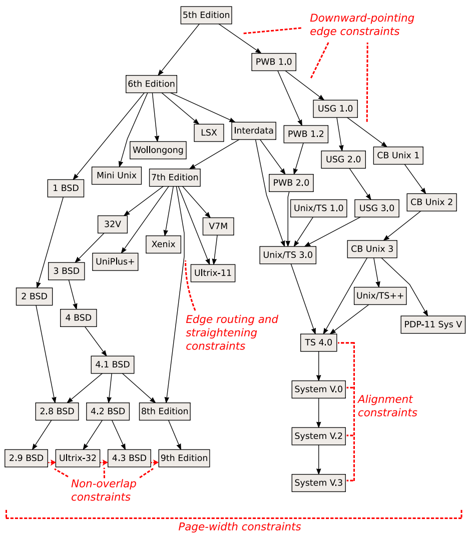


**Fig. 1.** A directed graph drawn using constrained force-directed layout. Separation constraints are used to ensure: (1) directed edges point downwards; (2) selected nodes are horizontally or vertically aligned; (3) the drawing fits within the page boundaries; and (4) nodes do not overlap edges or other nodes. Allowing constraints between nodes and edges means that edge routing criteria can also be considered, e.g., keeping edges as straight as possible while avoiding unnecessary crossings. The "history of unix" graph data is from `http://www.graphviz.org`.

Tamassia [5] allowed the edges to bend by treating dummy nodes as control points for splines. The problem with these methods is that the splines could not easily be prevented from overlapping one another and creating new crossings.

Recently, the force-directed model has been extended to allow *separation constraints* of the form $u + g \leq v$, enforcing a minimum gap $g$ between the positions $u$ and $v$ of pairs of objects in either the $x$ or $y$ dimensions in the drawing [6]. The basic idea is to modify the iterative step in *functional majorisation* [7] to solve a one-dimensional quadratic objective subject to the separation constraints for that dimension. Previously, it has been shown that separation constraints allow aesthetic requirements—such as placement of nodes below other nodes in directed graphs or containment of nodes in clusters—to be integrated into force-directed layout [6]. In this paper we show how they can be used to take into account aesthetic criteria involving edge routing.

Our basic procedure takes as input an initial layout for the graph including poly-line paths for the edges and rectangular bounding boxes for the node labels (or other text or graphics associated with nodes). This layout is then improved by moving nodes and edge bend points so that edges are straightened and made more uniform in length. Our approach is similar in spirit to that of Bertault, but instead of introducing repulsive forces between nodes and edges we introduce separation constraints between edge bend points and nodes and other edges. The result is drawings with no overlap between nodes and edges, and which preserve the edge crossing properties of the starting layout. Further, these drawings should have low stress with respect to the original goal function (1) while minimizing edge bends and overall edge length.

Our method—explored in detail in Section 2—has three main advantages over previous approaches. First, separation constraints allow us to guarantee that edge-edge crossings will not be introduced and that there will be no overlap between nodes or between nodes and edges. Second, since our approach is based on functional majorization it has better convergence properties than the earlier approaches which were solved with iterative local-search methods similar to those introduced by Kamada and Kawai [2] or Fruchterman and Reingold [8]. Third, as illustrated in Figure 1, we can use additional separation constraints to enforce other aesthetic criteria in the layout.

A key question is how to obtain the initial layout. Bertault [3] considered input generated with a planarisation based technique [9]. Such methods seek to find a maximal planar subgraph, draw this subgraph with no edge crossings, and then use heuristics to reinsert edges whilst creating as few crossings as possible.[1] Typically, the resulting drawings have few edge crossings but are aesthetically displeasing, which is why techniques such as that of Bertault [3] or the earlier simulated annealing based beautification algorithm [11] have been suggested as post-processing steps to improve the layout. The procedure described here can also be used for this purpose.

In Section 3 we introduce an alternative approach for obtaining the initial layout. Our technique first positions the nodes by performing a force-directed layout on the graph, ignoring edge routing. Next, we use the incremental connector routing library described in [12] to find the connector routes that minimize edge length and amount of bend. We then iteratively improve this using a simple greedy heuristic, re-routing the edges with the largest number of crossings to reduce crossings as long as this does not lead to very long edges or a large number of bends, as seen in Figure 7. The advantage of our technique is that the heuristic considers number of edge crossings, edge length, number of bends and degree of "bendiness." Clearly, all of these measures are important [13, 14] and since there is a trade-off between them it is important to consider them together. As far as we know this is the first approach to do so since previous approaches have either been planarization based and only considered the number of crossings [9], or have not considered crossings at all [15, 16, 12].

## 2 Modelling Edge Routing in Force-Directed Layout

We assume as input a graph $G = (V, E)$ and a layout for the graph. Each node $v \in V$ has a bounding box of dimensions given by $width(v)$ and $height(v)$, and each edge $e$ has a minimum width $width(e)$. The routing for each edge consists of a curved or piece-wise linear path between and around node bounding boxes. We use the functions $xpos(e, h)$ which returns a list of intersection points between the edge $e$ and the line $y = h$; $top(e)$ and $bottom(e)$ which return the topmost and bottom-most coordinate, respectively, through which $e$ passes. The functions $ypos(e, h)$, $leftlimit(e)$ and $rightlimit(e)$ are defined symmetrically.

[1] Of course there is no guarantee that such reinsertion strategies produce a drawing that is optimal with respect to crossings since the general crossing minimisation problem is NP-complete [10].

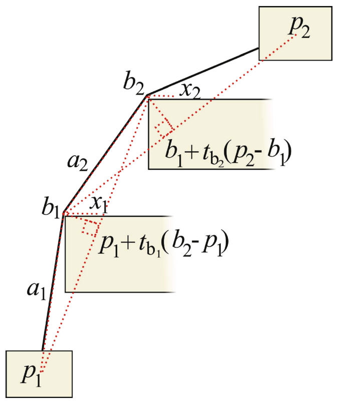


**Fig. 2.** The bend $b_1$ will be straightened to its projection point on the line $\boldsymbol{p_1b_2}$, e.g., in the $x$-dimension, to $x_1$. Bend $b_2$ will be similarly straightened towards the line $\boldsymbol{b_1p_2}$. The potential bend points $a_1$ and $a_2$ will be straightened to the line segment between actual bend points or the ends of the line, i.e., $a_1$ will be straightened to the line segment $\boldsymbol{p_1b_1}$.

Recall from [7] that in functional majorisation the value of the stress function (1) is reduced by alternately minimising quadratic forms in the horizontal and vertical axes that bound the stress functions:

$$x^T Lx - l_x^T x \quad , \quad y^T Ly - l_y^T y \tag{2}$$

where: $x$ and $y$ are $|V|$ dimensional vectors of node positions in each axis; the $|V| \times |V|$ Hessian matrix L is the graph Laplacian; and the linear arguments $l_{x,y}$ are computed before processing each axis based on the difference between ideal separation of nodes and their actual separation at the current placement (for details see [7]).

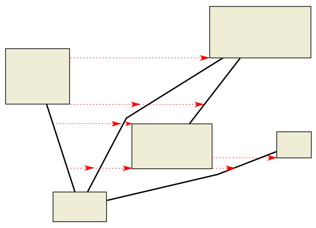
The input to our new layout problem includes bend points for some edges. Ideally we would like such edges to be straightened while still satisfying the original node separation objectives of the goal function and without creating any node/edge, node/node intersections.

Consider a bend point $b = (x_b, y_b)$ on a line from $p_1 = (x_1, y_1)$ to $p_2 = (x_2, y_2)$. The minimal change to the position of $b$ which straightens the line is to move $b$ to its projection $p_b$ on to the line $\boldsymbol{p_1p_2}$. We let $t_b$ be distance of $p_b$ along $\boldsymbol{p_1p_2}$, that is, $p_b = p_1 + t_b(p_2 - p_1)$ where $t_b = \frac{\boldsymbol{p_1b} \cdot \boldsymbol{p_1p_2}}{||\boldsymbol{p_1p_2}||^2}$ (from the dot product rule for scalar projection) Since all paths have minimal length and bends we have that the projection point must lie between $p_1$ and $p_2$, i.e. that $0 \leq t_b \leq 1$ .

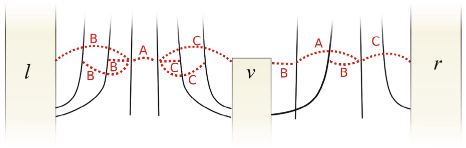
So for an edge $(v_i, v_j) \in E$ with one bend at position $b$ we can straighten the edge when moving vertices and bend points horizontally by minimising $f(x_i, x_j, x_b) = (x_b - (1 - t_b)x_i - t_b x_j)^2$. If an edge is routed through multiple bend points $b[1], b[2], ..., b[n]$ we can straighten all bends by minimising:

$$f(x_i, x_b[1], x_b[2]) + \sum_{k=2}^{n-1} f(x_{b[k-1]}, x_{b[k]}, x_{b[k+1]}) + f(x_{b[n-1]}, x_{b[n]}, x_j)$$

and similarly, when placing points vertically, we will straighten edges by minimising an equivalent set of expressions over $y$. This is illustrated in Figure 2.



(a) Identifying bends and potential bends for horizontal placement



(b) A complex set of constraints generated by the opening of node $v$

**Fig. 3.** Examples showing the separation constraints (dashed arrows) required to prevent the creation of new node/node and node/edge crossings when moving nodes horizontally

In summary, for the three variables $u, v, b$ involved in each bend $b$, we have $f(u, v, b) = (u, v, b)^T H(u, v, b)$ where

$$H(f(u,v,b)) = \nabla^2 f(u,v,b) = 2 \cdot \begin{pmatrix} (1-t_b)^2 & t_b(1-t_b) & t_b - 1 \\ t_b(1-t_b) & t_b^2 & -t_b \\ t_b - 1 & -t_b & 1 \end{pmatrix} \tag{3}$$

Thus, if $B$ is the set of bends (or potential bends, as will be discussed later) we have to define $|B|$ new variables and to redefine the goal functions (2) in terms of $m = |V| + |B|$ variables. We define a new matrix $A$ that contains the quadratic terms for each bend and the ideal node separation terms of the graph Laplacian:

$$A = L' + \sum_{b \in B} A_b \tag{4}$$

where $L'$ is an $m \times m$ matrix with the top-left $|V| \times |V|$ cells set to $L$ and the remaining cells 0; for each bend $b \in B$, $A_b$ is a symmetric matrix with the 9 cells corresponding to the variables $u, v, b$ set to the entries in $H(f(u, v, b))$ as above and all other cells 0. This gives us a new goal function to minimise in each dimension. For example in the $x$ dimension we have:

$$x^T A x + l'^T x \tag{5}$$

Where the linear terms in (2) are unaffected by the new quadratic terms so $l'$ is simply an $m$-vector s.t. $l' = [l|0, \ldots]$. It is simple to prove that $A$ is symmetric and positive semi-definite meaning that efficient convex optimisation methods such as the gradient-projection method described in [6] are applicable. Further, since each bend point requires only a small number of entries in the $A$ matrix the complexity of the optimisation process will increase only linearly in the number of additional bend variables when a sparse matrix data-structure is used.

We solve this quadratic objective subject to a set of separation constraints. These constraints prevent: bend points and hence edges from overlapping any node's bounding box; nodes overlapping one another; bend points passing through another edge and so increasing the number of edge crossings; and guarantee a minimum separation between parallel edges.

Figure 3 shows the constraints generated for when moving nodes horizontally. Notice that additional bend points are introduced in some straight edge segments. A new bend-point is created wherever the top or bottom of a node is visible from that segment where nodes restrict visibility but edges do not. We distinguish between the original *active* bends and these additional *potential* bends. We only want an edge to bend at a potential bend point if the node associated with that bend point (which is currently not touching the edge) is moved as a result of straightening other edges and collides with the edge. We therefore straighten potential bend points to the line segment between active bend points (see Figure 2) and weight them significantly more than active bend points, to avoid introducing new bends where possible.

Figure 4 gives an algorithm for generating these constraints for the $x$-direction: the code for the $y$-direction is symmetric. We generate the constraints using a line-sweep algorithm related to standard rectangle overlap detection methods [17] and the non-overlap constraint generation algorithm [18]. To generate horizontal constraints, we perform a vertical sweep through the nodes and edges, keeping a horizontal "scan line" list of open nodes sorted by horizontal position and an unsorted list of open edges. When the scan line reaches the top of a new node, $v$, this is added to the list and its left and right node neighbours, $l$ and $r$, are computed. The function *neighbourhood_x_constraints* searches along the scan line at $y$ between $l$ and $r$ for intersections with open edges. Bend variables are created at these intersection points and constraints are generated between them. Constraints between edges and the nodes to which they are connected should not be generated or else edges may become "wrapped" around their end-points. We therefore consider several cases for constraint generation. Case A generates constraints between adjacent bend points for edges not connected to $l$, $r$ or $v$. Case B considers edges to the right of $l$ or $v$, skipping those edges connected to $l$ or $v$, and Case C similarly handles edges to the left of $v$ or $r$. These three cases are illuminated in Figure 3(b).

The worst-case time complexity of procedure *generate_x_constraints*$(V, E)$ is $O(|V|(|V| + |E| \log |E|))$ and it will generate $O(|V| \cdot (|V| + |E|))$ constraints.

## 3 Computing the Initial Layout

The algorithm given in the previous section requires an initial layout including node positions and poly-line routes for edges. One approach is to use a planarisation based technique [9]. These seek to find a maximal planar subgraph, draw this in a planar way, and then reinsert edges whilst producing few crossings. However, algorithms for drawing strictly planar graphs (or subgraphs) generally require further refinement and so our algorithm can be used for this purpose. An example is shown in Figure 8. We have also explored an alternative approach for obtaining the initial layout and edge routing. This is novel, so we describe it in more detail. It has four main steps.

```
procedure $generate\_x\_constraints(V, E)$
    $C \leftarrow \emptyset$
    $Events \leftarrow \{(top(v), Open, v), (bottom(v), Close, v) | v \in V\}$
        $\cup\{(top(e), Open, e), (bottom(e), Close, e) | e \in E\}$
    sort $Events$ by decreasing $y$ position
    # OpenNodes is maintained in order of each node's $x$ position
    $OpenNodes \leftarrow \emptyset$
    $OpenEdges \leftarrow \emptyset$ % OpenEdges is unordered
    for each $(y, type, obj) \in Events$ do
        if $obj$ is a node then
            $v \leftarrow obj$
            $l \leftarrow$ first node to left of $v$ in $OpenNodes$
            $r \leftarrow$ first node to right of $v$ in $OpenNodes$
            $Z \leftarrow Z \cup neighbourhood\_x\_constraints(v, l, r, y, OpenEdges)$
        if $type = Open$ then
            if $obj$ is a node then
                $insert(obj, OpenNodes, xpos(obj))$
            else if $obj$ is an edge then
                add $obj$ to $OpenEdges$
            endif
        else if $type = Close$ then
            delete $obj$ from $OpenNodes$ / $OpenEdges$
        endif
    endfor
return $Z$

procedure $neighbourhood\_x\_constraints(v, l, r, y, OpenEdges)$
    # If $l$ (or $r$) is unspecified we say $l$(or $r$) $= 0$ and include all edges to the left (or right) of $v$.
    $L \leftarrow \emptyset$, $minx = -\infty$
    if $l \neq 0$ then
        $L \leftarrow \{l\}$, $minx = xpos(l)$
    endif
    $L \leftarrow L \cup \{dummyNode(e, x, y) | e \in OpenEdges \wedge x \in xpos(e, y)$
        $\wedge\ minx < x < xpos(v) \wedge e$ is not connected to $l$ or $v\} \cup \{v\} \cup \ldots$
    ... similarly add dummy nodes for edges between $v$ and $r$ including $r$ (if $r \neq 0$)
    $u_A \leftarrow u_B \leftarrow u_C \leftarrow 0$
    for each $w \in L$ in ascending $xpos$ order do
        A: if $edge(w) \wedge \neg end(w) \in \{v, l, r\}$ then
                if $u_A \neq 0$ then
                    $Z \leftarrow Z \cup \{xpos(u_A) + (width(u_A) + width(w))/2 \leq xpos(w)\}$
                $u_A \leftarrow w$
            else if $node(w)$ then
                $u_A \leftarrow 0$
            endif
        B: if $edge(w)$ then
                if $isend(u_B, edge(w))$ then
                    $skipList \leftarrow skipList \cup \{w\}$
                else
                    for each $s \in skipList$ do
                        $Z \leftarrow Z \cup \{xpos(s) + (width(s) + width(w))/2 \leq xpos(w)\}$
                        $skipList \leftarrow \emptyset$
                endif
            else if $node(w)$ then
                $skipList \leftarrow \{w\}$
                $u_B \leftarrow w$
            endif
    for each $w \in L$ in descending $xpos$ order do
        C: # symmetrical to B
    # May need to also generate constraints $l, v$ and $v, r$ if necessary
return $Z$
```

**Fig. 4.** Vertical scan algorithm to create a set $Z$ of constraints to prevent node/edge or node/node overlap when moving nodes horizontally. The procedures *generate_y_constraints* and *neighbourhood_y_constraints* for the horizontal scan are symmetrical. Constraints generated for the three distinct cases $A$, $B$ and $C$ are illustrated in Figure 3(b).

The first step is to position the nodes by performing a force-directed layout on the graph, ignoring edge routing. A method such as [18] can be used to remove node overlap, allowing edges to be routed between neighbouring nodes.

Step two is to use the incremental poly-line connector routing library described in [12] to compute poly-line routes for each edge, which minimise edge length and amount of bend. This is done by constructing a visibility graph for the nodes, itself containing a node for each vertex of the bounding box of each node in the original graph. The visibility graph contains an edge between two nodes iff they are mutually visible, i.e., there is no intervening obstacle. Next, the edge paths are routed using an $A^\star$ based-search to find the best route for each edge. The cost of routing each edge is $O((|E| + |V|) \log |V|)$ where $|E|$ is the number of edges in the visibility graph.

Step three uses a simple greedy heuristic to reduce the number of edge crossings. Each edge with crossings is considered once, in decreasing order of crossings. Again we use an $A^\star$ based-search to find the best route for each edge, though this time the cost includes a penalty for each crossing. The cost of routing each edge taking into account edge-crossings is currently $O(|S|(|E|+|V|) \log |V|)$ where $|E|$ is the number of edges in the visibility graph and $|S|$ the number of segments in the routed edges.

The penalty for an edge route is simply the sum of the penalties for its segments where the penalty for an edge segment is given by:

$$cost = l + \alpha s + \frac{\beta a \log(a+1)}{10} + \gamma scc \tag{6}$$

where: $\alpha, \beta, \gamma$ are user specifiable penalties for, respectively, the segment penalty, the angle penalty and the crossing penalty. $l$ is the length of the segment. $a$ is the angle away from a straight line that this segment makes with the previous segment, scaled to the range $0 \le a \le 10$, therefore $a = 0$ means the two segments make a straight line. If this is not the first segment and $a > 0$, then $s = 1$, otherwise $s = 0$. Finally, $scc$ is the number of crossings for the segment.

The penalty function incorporates the three main features of poly-line edge routing that have been shown to affect user comprehension: edge length, number of bends and degree of bendiness [14]. The ability to use a flexible penalty function allows us to adjust the initial routing to match the desired combination of aesthetic criteria and their tradeoffs. Setting a very high penalty for crossings will produce routings with few crossings but this is not always ideal. By reducing the penalty we produce more pleasing routings such as the one shown in Figure 7(b). In this case, the four crossings at the perimeter are avoided but the one in the middle is allowed since it would otherwise result in a path of significantly greater length and amount of bendiness.

Finding poly-line edge crossings for a graph is not as simple as just determining the intersections between the segments of all edge paths. It is common for edges to bend around the same node corner or to share paths for part of their route, i.e., running along the same paths in the visibility graph. In these cases we want to distinguish between situations where they cross and where they only

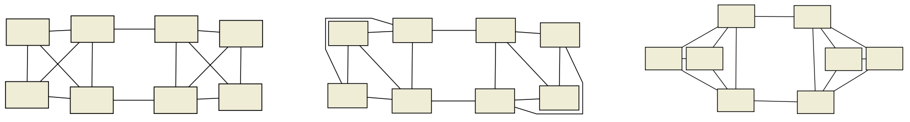

(a) Initial force-directed layout (b) Edges rerouted to reduce crossings (c) After straightening edges

**Fig. 6.** A small example showing the main steps in our layout process

touch or run parallel. We do this by comparing the order of edges entering and leaving shared paths or common bend points.

Since the length of the shared path has no effect on whether the two edges cross, we can treat bend points equivalently to shared paths. We start by looking for cases where the segments of two edges have a single shared endpoint. This is the beginning of a shared path or a common bend point. Such bend points always pass around the corner of a node, so we determine the order of edges entering the shared path at this point by finding which edge runs closest to the node. From here we follow the common segments along the shared path until they diverge again. We then determine the order of edges leaving the shared path, taking into account features of the bends such as the winding directions. If the two orders are different then we can tell that the edges cross along the shared path, rather than running parallel.

Finally, in step four, line segments are adjusted to slightly separate edges routed around the same corner of a node. This involves nudging the bend points of edges along shared paths, or at the points they cross or touch, as shown in Figure 5. To do this a sorted order for each of these points and shared paths is kept when determining crossings. This nudging step prevents the creation of additional edge crossings during the layout step.

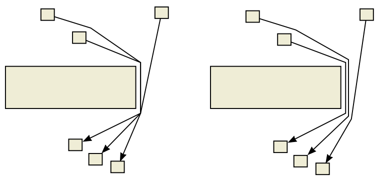


**Fig. 5.** An exaggerated example of the nudging we perform on shared edge bend points

## 4 Discussion

Figure 6 demonstrates how the edge routing and straightening procedures are applied in practice. We begin with the output of an unconstrained stress-majorisation layout in which edge routing is ignored. Edges are all close to their ideal length thus minimising (1). We apply edge routing as described above, penalising routes with crossings. A planar layout is obtained but with longer edges and a number of bends. Constraints are then generated using the algorithm from Figure 4 and the stress majorisation layout is rerun subject to these constraints. The result retains the planar layout but the edges are straightened.

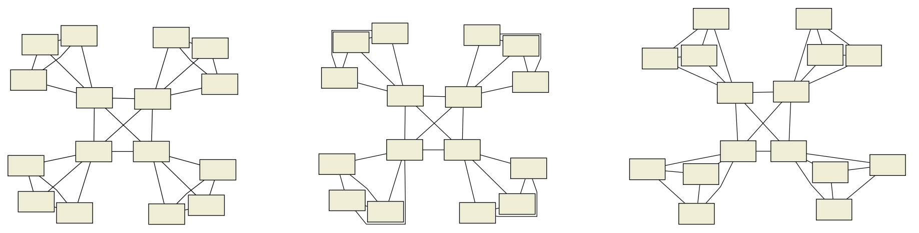

(a) Initial force-directed layout (b) Edges routed to penalise crossings (c) With edge straightening

**Fig. 7.** An example graph from Kamada-Kawai [2]

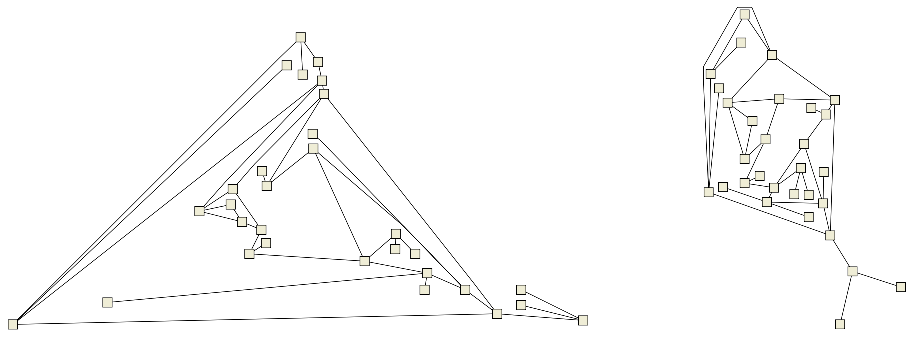
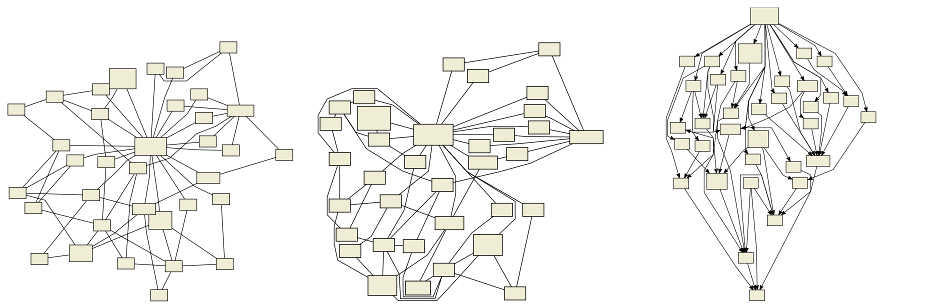

(a) Output of a planar graph layout algorithm (b) With edge straightening

**Fig. 8.** Applying edge straightening to the output of a planar layout algorithm

Figure 7 shows an example graph from Kamada and Kawai [2]. Note that the graph is planar, yet our routing allows one crossing to remain because removing the final crossing would require a very long edge. The result is a layout with more consistent edge length and greater symmetry than a completely planar drawing would permit.

Figure 8 shows the result of applying the edge straightening method directly to the output of a typical planar graph layout algorithm. The layout is considerably compacted by allowing edges to bend while the edge routing topology is preserved. Note that the layout is somewhat compressed by the requirement to preserve the ideal distance between two nodes on the outer face that have finished up on opposite sides of the drawing. This could perhaps be alleviated by a further refinement step that relaxed ideal distances between nodes connected by an edge following a long route.

Our final example in Figure 9 is more typical of the type of graphs encountered in actual applications, namely: nodes have variable size bounding boxes depending on the labels required; there are many more edges than our previous examples; and there are a small number of nodes of high degree, e.g., the two highest degree nodes have 19 and 8 connections. In Figures 9(b) and 9(c)



(a) Initial force-directed layout: 43 crossings

(b) With edges routed to penalise crossings and straightened: 18 crossings

(c) With additional constraints to require downward pointing edges: 26 crossings

**Fig. 9.** A more realistic graph visualisation application (a Bayesian net)

edges that share a common path have been separated using the method described in Section 3. Note that we have not applied such a separation where edges share a common end point—the main goal of nudging being to disambiguate routes that come together and then diverge. Figure 9(c) demonstrates how other constraints—in this case downward pointing directed-edges—can be used in addition to edge straightening constraints. This is also demonstrated in the example shown in Figure 1.

The entire process, including computation of the initial layout, routing of all edges and then the straightening phase takes only a few seconds for each of the examples shown.

## 5 Conclusion and Further Work

Extending stress majorisation techniques to handle separation constraints allows us to naturally handle a wide number of aesthetic criteria and drawing conventions. Here we have shown that separation constraints allow edge routing to be integrated into force-directed layout. The precise encoding is not obvious, and relies on using separation constraints in combination with a modification to the one-dimensional quadratic objective function to essentially model an arbitrary linear inequality.

Another contribution of the paper is to present a simple heuristic for finding poly-line edge routes that tries to minimise the number of edge crossing while still taking into account the edge length and degree of bendiness. There has been surprisingly little research into such heuristics and we believe that there is considerable scope for further work.

We think it is significant that our algorithm can be used to improve layouts obtained with planarization based methods. We plan to further explore how force-directed layout can be combined with such combinatorial techniques.

## References

1. Fisk, C.J., Isett, D.D.: ACCEL: automated circuit card etching layout. In: DAC'65: Proceedings of the SHARE design automation project, ACM Press (1965) 9.1–9.31
2. Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Information Processing Letters **31** (1989) 7–15
3. Bertault, F.: A force-directed algorithm that preserves edge crossing properties. In: Proc. 7th Int. Symp. on Graph Drawing (GD'99). Volume 1731 of LNCS., Springer (1999) 351–358
4. Brandes, U., Wagner, D.: Using graph layout to visualize train interconnection data. In: Proc. 6th Int. Symp. on Graph Drawing (GD'98). Volume 1547 of LNCS., Springer (1998) 44–56
5. Finkel, B., Tamassia, R.: Curvilinear graph drawing using the force-directed method. In: Proc. 12th Int. Symp. on Graph Drawing (GD'04). Volume 3383 of LNCS., Springer (2004) 448–453
6. Dwyer, T., Koren, Y., Marriott, K.: IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. In: Proc. IEEE Symp. on Information Visualisation (Infovis'06). (To appear 2006)
7. Gansner, E., Koren, Y., North, S.: Graph drawing by stress majorization. In: Proc. 12th Int. Symp. Graph Drawing (GD'04). Volume 3383 of LNCS., Springer (2004) 239–250
8. Fruchterman, T., Reingold, E.M.: Graph drawing by force-directed placement. Software - Practice and Experience **21** (1991) 1129–1164
9. Gutwenger, C., Mutzel, P., Weiskircher, R.: Inserting an edge into a planar graph. In: SODA '01: Proc. of the 12th Annual ACM-SIAM Symp. on Discrete Algorithms, Society for Industrial and Applied Mathematics (2001) 246–255
10. Garey, M.R., Johnson, D.S.: Crossing number is NP-complete. Journal of Algebraic Discrete Methods **4** (1983) 312–316
11. Harel, D., Sardas, M.: Randomized graph drawing with heavy-duty preprocessing. In: AVI '94: Proceedings of the Workshop on Advanced Visual Interfaces, New York, NY, USA, ACM Press (1994) 19–33
12. Wybrow, M., Marriott, K., Stuckey, P.J.: Incremental connector routing. In: Proc. 13th Int. Symp. on Graph Drawing (GD'05). Volume 3843 of LNCS., Springer (2006) 446–457
13. Purchase, H.C., Cohen, R.F., James, M.: Validating graph drawing aesthetics. In: Proc. 4th Int. Symp. on Graph Drawing (GD'96), Springer (1996) 435–446
14. Ware, C., Purchase, H., Colpoys, L., McGill, M.: Cognitive measurements of graph aesthetics. Information Visualization **1** (2002) 103–110
15. Dobkin, D.P., Gansner, E.R., Koutsofios, E., North, S.C.: Implementing a general-purpose edge router. In: Proc. 5th Int. Symp. on Graph Drawing (GD'97). Volume 1353 of LNCS., Springer (1997) 262–271
16. Freivalds, K.: Curved edge routing. In: Proc. of the 13th Int. Symp. on Fundamentals of Computation Theory (FCT '01). Number 2138 in LNCS, Springer (2001) 126–137
17. Preparata, F.P., Shamos, M.I. In: Computational Geometry. Springer (1985) 359–365
18. Dwyer, T., Marriott, K., Stuckey, P.: Fast node overlap removal. In: Proc. 13th Int. Symp. on Graph Drawing (GD'05). Volume 3843 of LNCS. (2006) 153–164

# Multipole-Based Force Approximation Revisited – A Simple but Fast Implementation Using a Dynamized Enclosing-Circle-Enhanced k-d-Tree

Ulrich Lauther

Siemens AG, CT SE 6
Otto-Hahn Ring 6, 81730 München, Germany
ulrich.lauther@siemens.com

**Abstract.** Most force-directed graph drawing algorithms depend for speed crucially on efficient methods for approximating repulsive forces between a large number of particles. A combination of various tree data structures and multi-pole approximations has been successfully used by a number of authors. If a multi-level approach is taken, in the late (and due to the large number of particles computationally intensive) steps, movements of particles are quite limited. We utilize this fact by basing force-calculations on an easy updatable tree data structure. Using explicit distance checks instead of relying on implicit guarantees provided by quadtrees and avoiding local expansions of the multi-pole expansion leads to a very simple implementation that is faster than comparable earlier methods. The latter claim is supported by experimental results.

## 1 Introduction

In several force-directed graph drawing methods [1,2,3,4] the nodes of a graph to be drawn are modeled as charged particles that repel each other and edges are represented as springs with some "ideal" length that attract or repel the two incident nodes, depending on their actual length. A placement of nodes that minimizes the total energy in the system often results in a nice straight line drawing of the underlying graph. This placement is achieved by iteratively computing the total force acting on each node and then moving the nodes accordingly.

In order to make this approach work in practice for large graphs, two considerations are crucial: Firstly, as a naive approach to calculating repulsive forces would take $O(N^2)$ steps per iteration for a graph with $N$ nodes, we need a fast method for approximating these forces with sufficiently high accuracy.

Secondly, to avoid an overly large number of iterations, we need a multi-level approach that first generates a series of coarsened graphs $G_1,\ldots,G_k$ from the initial graph $G_0$, generates a drawing for the small graph $G_k$, and then refines this drawing by constructing an initial placement for the nodes of $G_i$ from the node positions in $G_{i+1}$ which is then optimized by force-directed iterations. Note, that at each refinement level we start with a "reasonable" initial placement of nodes.

Though we have implemented such a coarsening/refinement approach (along the lines suggested by Hachul and Jünger [5]), it is not further discussed in this paper; its focus is on a simple and efficient approach to approximation of repulsive forces.

## 2 Previous Work

Beginning with Barnes and Hut [6] a number of force approximation schemes have been developed and used in the context of graph drawing [7,8] that use quadtrees or variants thereof for hierarchical clustering of particles and approximating forces either by substituting a "group particle" at the center of gravity for the particles of a cluster [6] or by calculation of *multipole expansions* [9] for the clusters. These approaches differ in how the tree is constructed and whether explicit distance calculations are used to steer the following force calculations or if implicit properties of the quadtree's topology are used. Hachul [10] , after careful analysis of different quadtree construction methods, came up with the *reduced bucket quadtree* that can be constructed in $O(N \log N)$ steps for $N$ particles.

One drawback of quadtrees is that they divide the area containing the particles into subregions of equal size, not equal population. If particles are not uniformly distributed, they become unbalanced and quite involved algorithms [10] are needed to achieve $O(N \log N)$ complexity for tree construction.

## 3 The Enclosing-Circle-Enhanced Modified k-d-Tree

As opposed to quadtrees, k-d-trees introduced by Bentley [11] partition the set of particles based on population, not on space.

### 3.1 Definition of the Tree Data Structure

We use a variant of a 2-d-tree, defined as follows: The root node represents the set of all $N$ particles. Each node representing $k$ particles has two children, where the left child contains the $k/2$ particles with lowest coordinates and the right child the remaining $k - k/2$ particles. (Note: throughout the paper we use integer division). If the bounding rectangle of a node with dimensions $dx$ and $dy$ has $dx > dy$, i.e., it is "horizontally oriented", the x-coordinate is used for splitting, otherwise the y-coordinate. (This is different from Bentley's k-d-trees, where coordinate directions are used in a strict round robin fashion; it helps to avoid long skinny regions which group points together that are far away from each other).

The leafs of the tree are buckets, i.e., they contain a list of particles. The bucket size is bounded by a predefined parameter $B$. Each bucket contains $b$ particles with $B/2 \leq b \leq B$. (Unless pathologically $N < B/2$).

In each leaf we store center and radius of the smallest circle enclosing all particles of its bucket. The interior nodes too contain enclosing circles of their subtree, either exact values or upper bounds, depending on the tree construction method, see below.

## 3.2 Tree Construction

Such a tree can be constructed in $O(N \log N)$ steps as follows:

First, we build two singly linked lists of particles and sort one list by x-coordinates, the other one by y-coordinates. This takes $O(N \log N)$ steps using list merge sort or any other appropriate method.

We create the root node, determine its appropriate splitting direction and append the node, its splitting direction, and the two initial lists to a queue of nodes to be processed.

Then, in the main loop of the algorithms, a node, its lists, and its splitting direction are popped from the queue and the node is split if it contains more than $B$ particles. Splitting a node proceeds as follows: we traverse one of the two sorted lists up to the median element, mark the traversed elements, and transfer them to a list $L1$ (this takes $k/2$ steps for a node representing $k$ particles). The remaining particles are transferred to list $L2$ (list concatanation, one step). Next, we split the second list (the other coordinate direction) based on the marks just made: elements are popped from the list and depending on their mark appended to one or the other of two result lists. Note, that the lists stay sorted during this step. This costs another $k$ steps. Finally, two child nodes are created and appended to the queue together with the four lists just created. If the node taken from the queue contains not more than $B$ nodes, a leaf is created containing one of the associated particle lists. By construction the resulting tree is fully balanced. The work at each level of the tree is $O(N)$. Thus, with $O(\log N)$ tree levels and including the initial sorting step, the overall complexity is $O(N \log N)$, independent of how particles are distributed in the plane.

It remains to discuss how the enclosing circles are created. For the leafs, we could employ any algorithm without affecting the overall complexity, as the size of the buckets is bounded. However, luckily, there is a very efficient method that computes the smallest enclosing circle for a set of $n$ points in the plane in $O(n)$ expected time, introduced by Welzl [12]. If we used this method during tree construction, when the sets of particles for each node and leaf are readily available, the overall *expected* time complexity became $O(N \log N)$.

However, there is a faster method - both from the complexity point of view and in practice -, if we sacrifice some accuracy. We proceed in four passes: In the first pass, the tree is constructed as described. In the second pass, smallest enclosing circles are calculated for leafs, using Welzl's algorithm. In the third pass, traversing the tree from the leafs bottom up, the bounding circles of nodes are initialized with the largest leaf-circle in their subtree, which is easily found by selecting the larger circle of the two children of a node. In the last pass, for each leaf, node-circles in the path from the leaf up to the root are modified - if needed - such that they fully overlap with the leaf-circle. (Passes two to four could be replaced by just combining pairs of circles bottom up, but this gives inferior results). Passes one to three take $O(N)$ steps, the last pass takes

$O(N \log N)$ steps, as the paths from leaf to root have $O(\log N)$ length. In general, the bounding circles calculated by this method will be 10 to 15% larger than the smallest enclosing circle of the set of particles in the subtree, but this method is significantly faster than the exact method and - as experiments have shown - does not significantly degrade the speed of force calculations carried out based on the tree.

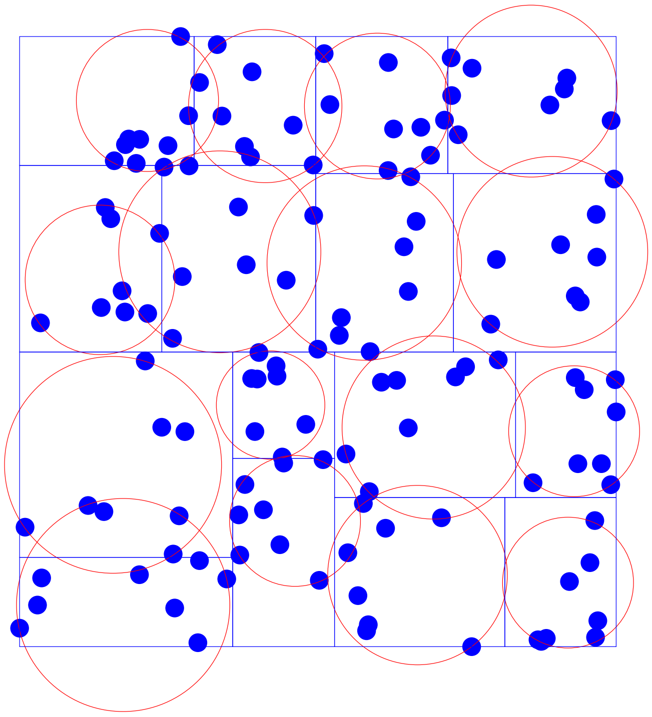


**Fig. 1.** k-d-tree like partioning of a set of particles into leaf rectangles and corresponding bounding circles

Figure 1 shows for a small example the partitioning of the plane into rectangular areas achieved by building the 2-d-tree from a set of particles. We see also the enclosing circles for the leaf nodes of the tree. Note that in general, these circels are smaller than the circumscribed circle of the corresponding leaf-rectangle, that had to be used for distance calculations if our circles were not available.

## 3.3 Tree Update

In the main loop of the force-directed spring embedder, particles are moved according to the total force acting on them. Due to the multi-level approach taken, these movements are small because particle positions are initialized with reasonable values derived from the previous level in the hierarchy of coarsened graphs. Therefore, there is no need to create the tree from scratch for each iteration. (It *is* created from scratch for each level of the hierarchy). Instead, we keep the tree structure and just update the enclosing circles in leafs and interior nodes, provided the average movement of particles in the last iteration was small (less than 10% of of the average leaf radius). For each particle, we follow the path from its leaf to the root of the tree and check if it is still inside the enclosing circle of the node. If not, the enclosing circle is appropriately enlarged. Otherwise, we are done with this particle as nodes further up cannot be affected. Typically, this loop terminates already at the bottom level and the typical runtime is 10 times smaller than that for building the tree from scratch. Nevertheless, we may rebuild the graph every, say 10th, iteration in order to avoid too much inaccuracy of bounding circles, without affecting speed significantly.

This speedup strategy is especially valuable at the last of the multi-level steps where the full graph is processed and node movements are small.

### 3.4 Tree Storage

As the size of the tree is known from the very beginning and its structure does not change, it is convenient to store the tree (actually pointers to interior tree nodes and leafs) in an array, just as a binary heap is conventionally stored: for a node at position $i$ we find its predecessor at position $i/2$ and its children at positions $2i$ and $2i+1$. Thus we save three pointers per node.

## 4 Calculation of Repulsive Forces

As invented by Greengard [9] and thoroughly discussed by Hachul [10] our calculation of repulsive forces is based on $p-$*term multipole expansions* of the potential energy due to groups of charged particles represented by nodes of our tree.

### 4.1 Calculation of p-Term Multipole Expansions

In what follows, coordinates, forces, and coefficients are complex numbers. For coordinates, we can convert between two-dimensional vectors and complex numbers in the obvious way, for forces we have to take the conjugate first. Formulas are taken from [10] where also proofs have been given.

The p-term multipole expansion with parameter $p$

$$e(z) = a_0 \log(z - z_0) + \sum_{k=1}^{p} \frac{a_k}{(z-z_0)^k}$$

with coefficients

$$a_0 = m \quad \text{and} \quad a_k = -\sum_{i=1}^{m} \frac{(p_i - z_0)^k}{k}$$

approximates the potential energy at point $z$ with $|z - z_0| > r$ due to $m$ particles with unit charge located in the plane at points $p_i$ within a circle with radius $r$ around $z_0$.

From the p-term multipole expansion we can derive the approximate force $f(z)$ that acts on a particle with unit charge at position $z$ outside the circle $(z_0, r)$ as

$$f(z) = \frac{a_0}{(z-z_0)} - \sum_{k=1}^{p} \frac{k \cdot a_k}{(z-z_0)^{k+1}}$$

For the efficient calculation of p-term multipole expansions in the nodes we also need to know that a p-term multipole-expansion around center $z_0$ can be shifted to a new position $z_1$. The coefficients $b_i$ for the shifted expansion are obtained by

$$b_0 = a_0 \quad \text{and} \quad b_l = \frac{-a_0(z-z_1)^l}{l} + \sum_{k=1}^{l} a_k (z_0 - z_1)^{l-k} \binom{l-1}{k-1}$$

Using these formulas, we first calculate the coefficients $a_0 \ldots a_p$ of the p-term multipole expansion for each leaf of the tree. Then, traversing the tree bottom up, we shift the expansions of the children of a node to the node's center and add the two sets of coefficients. The whole process takes $O(p^2 N \log N)$ steps.

In [10] in addition to multipole expansions so called *local expansions* are defined and used for force calculations. To simplify the implementation we do not, however, use local expansions.

## 4.2 Using p-Term Multipole Expansions

The multipole expansions (i.e., their coefficients) are calculated once for each iteration of the force-directed placement, after the tree of particles has been built or updated. Force-calculation is then a quite simple step. We loop over all leafs L1 of the tree and proceed as follows:

– For the $b$ particles in the leaf's bucket, we calculate the mutual repulsive forces directly; this takes $b(b-1)/2$ steps.
– Then we need to calculate the forces from the other particles in the tree. We use a stack of interior nodes or leafs to be processed which is initialized with the root node. While the stack is not empty we pop an interior node or leaf and compare it with the current leaf L1. If the popped element is sufficiently far away, we use its multipole expansion to find the forces acting on the particles of the current leaf. Otherwise, if it is an interior node, its two children are pushed on the stack for further processing. If it is a leaf L2, we loop over the particles in L1 and either, if L2 is sufficiently far away from a particle, calculate the forces acting on it due to the multipole expansion of L2, or else calculate the forces acting on the particle from those of L2 directly. In the first case we need $p$ calculations per particle, in the latter $b$ calculations, with $p$ the parameter of the p-term-multipole expansion and $b$ the number of particles in leaf L2.

The partial forces calculated as described are accumulated in the respective particles, so that we end up with the total repulsive force acting on each particle.

It remains to specify what "sufficiently far away" means. A particle is considered sufficiently far away from a node with radius $R$ of its bounding circle if its distance from the node's center is $\geq 2R$. A leaf with radius $r$ is considered sufficiently far away if the same condition holds for all points enclosed by the leaf's bounding circle, i.e., if for the distance $d$ between the two centers we have $d > 2R + r$. We tried to parameterize these conditions as in [6], as weaker distance requirements would speed up calculations, but this leads to a rapid degradation of accuracy if the node's circular bounds have been calculated exactly. If, however, the faster approximation is used for the calculation of node circles (that leads to larger bounds), we can compensate for this by multiplying node radii by 0.9 without degrading accuracy of force calculations too much.

As we have seen, we use explicit distance calculations between leafs and/or nodes to steer the algorithm instead of relying on implicit distance guarantees that come with quadtrees but require evolved bookkeeping of possibly interacting nodes [10]. We gain in simplicity and - as we will see, in speed - but we loose the possibility to give complexity results for this step. Thus, the efficiency of the method is shown by experimental results.

## 5 Experimental Result

The proposed algorithms have been implemented in C++ using our own C++-class library of basic algorithms and data structures [13] and as part of a spring embedder project. As the runtime of a spring embedder depends on many factors, e.g., local cooling schemes, termination conditions, translation of forces into particle movements, we focus here on the runtime for building and updating the particle tree and for the calculation of repulsive forces. Hachul [10] has made thorough comparisons of his reduced bucket quadtree based implementation with a number of other approximative methods and has shown his approach to be superior in all cases. It seems therefore justified to compare with his results only. For this purpose, we use the same particle distributions and numbers. We also use a similar hardware- and software platform (Intel Pentium 4 running Linux), however with a different clock rate. To make results comparable, we scaled our CPU-times such that running times for a naive $O(N^2)$ force calculation become identical.

### 5.1 Particle Distributions

We use these particle distributions as defined by Hachul [10]:

**Uniform.** Particles are uniformly distributed within the square $[0, 1] \times [0, 1]$.

**Non-uniform.** 20% of the particles are uniformly distributed within the square $[0, 1] \times [0, 1]$. The remaining particles are to equal parts distributed within circles with their center at $(\frac{1}{2}, \frac{1}{2})$ and radius $\frac{1}{4}$, $\frac{1}{16}$, $\frac{1}{64}$, and $\frac{1}{256}$, respectively.

**Quasi-converging.** Here particles are distributed on the line connecting (0,0) and $P = (0, 10^{25})$. Particle $i$ is placed at position $\frac{\frac{3}{4}P}{2^i-1}$ for $i = 1, 2, \ldots$ until $x_i$ becomes $< 10^{-25}$. The remaining particles are uniformly distributed on the line connecting (0,0) and $P = (10^{-25}, 10^{-25})$.

## 5.2 Running Times for Tree Construction

For the distributions described and particle numbers between 8000 and 128000 the particle tree was built and compared with the faster tree construction method $TC_b$ of [10]. The bucket size was fixed at 16 for all runs. As expected, the running times are independent of the distribution, with one exception: due to the correlation between x- and y-coordinates in the quasi-convergent distribution, the second pass of sorting in our method becomes very fast. The measured running times confirm the expected complexity of $O(N \log N)$ for all distributions. In general, our tree construction method is slower than that for the quadtree, with the exception of the quasi-convergent distribution, where the quadtree is significantly slower. However, in the spring-embedder application, in most cases the tree can be updated instead of building it from scratch, which is about 10 times faster.

To clarify, the running times in Table 1 show the time for building the tree once from scratch. This has to be done once per iteration in the implementation of [10], but is done only every 10th iteration in our approach; so the actual time spent on tree building is significatly smaller than the table seems to indicate.

**Table 1.** Running time for building the tree data structure (Note that the time needed for dynamically updating the tree after small movements of particles is by a factor of about 10 lower.)

| Distribution | Number of particles | Hachul's | Our's |
|---|---:|---:|---:|
| uniform | 8000 | 0.01 | 0.03 |
| | 16000 | 0.03 | 0.08 |
| | 32000 | 0.06 | 0.17 |
| | 64000 | 0.12 | 0.38 |
| | 128000 | 0.30 | 0.83 |
| non-uniform | 8000 | 0.02 | 0.03 |
| | 16000 | 0.03 | 0.08 |
| | 32000 | 0.08 | 0.17 |
| | 64000 | 0.15 | 0.38 |
| | 128000 | 0.34 | 0.84 |
| quasi-converging | 8000 | 0.22 | 0.02 |
| | 16000 | 0.44 | 0.06 |
| | 32000 | 0.71 | 0.13 |
| | 64000 | 1.49 | 0.27 |
| | 128000 | 2.75 | 0.64 |

### 5.3 Running Times for Force-Calculations

Now we compare the running times for calculation of repulsive forces. In [10], measurements were made with different parameter settings giving low, medium, and high accuracy, the latter being defined by an approximation error below $10^{-4}$. We restrict our comparisons to this high accuracy case, which is achieved - as in [10] - by setting the parameter $p$ of the multipole expansion equal to 6, 7, and 8 for the uniform, non-uniform, and quasi-converging distributions, respectively. The bucket size of the tree was again fixed at 16. Note that the times given refer just to the force calculations, excluding tree building and updating.

**Table 2.** Running time for calculation of repulsive forces

| Distribution | Number of particles | Hachul's | Our's | Exact |
|---|---:|---:|---:|---:|
| uniform | 8000 | 0.29 | 0.13 | 7.99 |
| | 16000 | 0.50 | 0.28 | 34.92 |
| | 32000 | 1.29 | 0.66 | 142.16 |
| | 64000 | 2.26 | 1.37 | 568.64 |
| | 128000 | 5.54 | 3.14 | 2274.56 |
| non-uniform | 8000 | 0.32 | 0.27 | 7.80 |
| | 16000 | 0.61 | 0.57 | 35.14 |
| | 32000 | 1.40 | 1.18 | 142.16 |
| | 64000 | 2.58 | 2.55 | 568.64 |
| | 128000 | 5.88 | 5.54 | 2274.56 |
| quasi-converging | 8000 | 0.29 | 0.16 | 7.83 |
| | 16000 | 0.60 | 0.34 | 35.08 |
| | 32000 | 1.06 | 0.71 | 142.16 |
| | 64000 | 2.13 | 1.45 | 568.64 |
| | 128000 | 4.00 | 3.13 | 2274.56 |

We see that our running times for force calculations for the uniform and for the quasi converging distribution are significantly lower than in the quadtree based approach and slightly smaller for the non-uniform distribution.

## 6 Conclusions

We have introduced a new particle tree variant, that stores smallest enclosing circles in its nodes and can be built fully balanced in $O(N \log N)$ steps independent of the particle distribution. After small movements of particles the tree can efficiently be updated, without any changes to its structure. Using explicit distance calculations between leafs and nodes and avoiding the calculation of local expansions, we achieve an easy to implement multipole expansion based approximation method for calculation of repulsive forces that compares - concerning speed and implementation efford - favorably with earlier work.

**Acknowledgements.** We are indebted to Stephan Hachul for many fruitful discussions of his work and our own evolving implementation. Thanks goes also to referees for helpful comments and suggestions.

## References

1. Eades, P.: A heuristic for graph drawing. Congressus Numerantium **42** (1984) 149–160
2. Kamada, T., Kawai, S.: An Algorithm for Drawing General Undirected Graphs. Information Processing Letters **31** (1989) 7–15
3. Fruchterman, T., Reingold, E.: Graph Drawing by Force-directed Placement. Software–Practice and Experience **21** (1991) 1129–1164
4. Davidson, R., Harel, D.: Drawing Graphs Nicely Using Simulated Annealing. ACM Transaction on Graphics **15** (1996) 301–331
5. Hachul, S., Jünger, M.: Drawing large graphs with a potential-field-based multilevel algorithm. In Pach, J., ed.: Graph Drawing. Volume 3383 of Lecture Notes in Computer Science., Springer (2004) 285–295
6. Barnes, J., Hut, P.: A hierarchical $\mathcal{O}(N \log N)$ force-calculation algorithm. Nature **324** (1986) 446–449
7. Quigley, A., Eades, P.: FADE: Graph Drawing, Clustering, and Visual Abstraction. In Marks, J., ed.: Graph Drawing 2000. Volume 1984 of Lecture Notes in Computer Science., Springer-Verlag (2001) 197–210
8. Tunkelang, D.: JIGGLE: Java Interactive Graph Layout Environment. In Whitesides, S.H., ed.: Graph Drawing 1998. Volume 1547 of Lecture Notes in Computer Science., Springer-Verlag (1998) 413–422
9. Greengard, L.: The Rapid Evaluation of Potential Fields in Particle Systems. ACM distinguished dissertations. The MIT Press, Cambridge, Massachusetts (1988)
10. Hachul, S.: A Potential-Field-Based Multilevel Algorithm for Drawing Large Graphs. PhD thesis, Institut für Informatik, Universität zu Köln, Germany (2005) URN: `urn:nbn:de:hbz:38-14097`, URL: `http://kups.ub.uni-koeln.de/volltexte/2005/1409`.
11. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18** (1975) 509–517
12. Welzl, E.: Smallest enclosing disks (balls and ellipses). New Results and New Trends in Computer Science **555** (1991) 359–370
13. Lauther, U.: The c++ class library turbo - a toolbox for discrete optimization. Software@Siemens (2000) 34–36

# SSDE: Fast Graph Drawing Using Sampled Spectral Distance Embedding

Ali Çivril, Malik Magdon-Ismail, and Eli Bocek-Rivele

Computer Science Department, RPI, 110 8th Street, Troy, NY 12180
{civria,magdon}@cs.rpi.edu, boceke@rpi.edu

**Abstract.** We present a fast spectral graph drawing algorithm for drawing undirected connected graphs. Classical Multi-Dimensional Scaling yields a quadratic-time spectral algorithm, which approximates the real distances of the nodes in the final drawing with their graph theoretical distances. We build from this idea to develop the linear-time spectral graph drawing algorithm SSDE. We reduce the space and time complexity of the spectral decomposition by approximating the distance matrix with the product of three smaller matrices, which are formed by sampling rows and columns of the distance matrix. The main advantages of our algorithm are that it is very fast and it gives aesthetically pleasing results, when compared to other spectral graph drawing algorithms. The runtime for typical $10^5$ node graphs is about one second and for $10^6$ node graphs about ten seconds.

## 1 Introduction

A graph $G = (V, E)$ is a pair where $V$ is the vertex set and $E$ is the edge set, which is a binary relation over $V$. The graph drawing problem is to compute an aesthetically pleasing layout of vertices and edges so that it is easy to grasp visually the inherent structure of the graph. In this paper, we only consider straight-line edge drawings for which a variety of aesthetic criteria have been studied: number of edge crossings; uniform node densities; symmetry. Depending on the aesthetic criteria of interest, various approaches have been developed, and a general survey can be found in [13,22].

For straight-line edge drawings, the graph drawing problem reduces to the problem of finding the coordinates of the vertices in two dimensions. A popular approach is to define an energy function or a force-directed model with respect to vertex positions, and to iteratively compute a local minimum of the energy function. The positions of the vertices at the local minimum produce the final layout. This approach is generally simple and easy to extend to new energy functions. Various energy functions and force models have been studied (see for example [6,12]) and there exist several improvements to handle large graphs, most of them concentrating on a multi-scale paradigm. Multi-scale approaches involve laying out a coarser level of the graph first, and then taking advantage of this coarse layout to compute the vertex positions at a finer level (see for example [9,24]).

*Spectral graph drawing* was first proposed by Hall in 1970 [8] and it has become popular recently. We use the term spectral graph drawing to refer to any approach that produces a final layout using the spectral decomposition of some matrix derived from the vertex and edge sets of the graph. A general introduction can be found in [14]. In this paper, we present the spectral graph drawing algorithm SSDE (Sampled Spectral Distance Embedding), using a similar formulation that was introduced in [5], which uses Classical Multi-Dimensional Scaling (CMDS) techniques for graph drawing. CMDS for graph drawing was first introduced in [17] and recently, a similar idea using CMDS technique, was proposed by Koren and Harel in [15] using a slightly different formulation. CMDS uses the spectral decomposition of the graph theoretical distance matrix to produce the final layout of the vertices. In the final layout, the pair-wise Euclidean distances of the vertices approximate the graph theoretical distances. The main disadvantage of this technique from the computational perspective is that one must perform an all-pairs shortest path computation, which takes $O(|V||E|)$ time. The space complexity of the algorithm is also quadratic since one needs to keep all the pair-wise distances. This prevents large graphs having more than $10,000$ nodes from being drawn efficiently.

SSDE uses an approximate decomposition of the distance matrix, reducing the space and time complexity considerably. Some theoretical properties of such matrix decompositions have been studied in [20]. The fact that the distance matrix is symmetric allows us to express the decomposition in a simpler way. SSDE consists of three main steps:

(i) *Sampling:* a constant number $c$ of nodes are sampled from the graph for which the graph theoretical distances to all other nodes are computed. Let the matrices $C$ and $R$ denote the corresponding rows and columns of the distance matrix that have been computed, where $R = C^T$. The complexity of this step is $O(c|E|)$ for unweighted graphs, using BFS for each sampled node.

(ii) *Computing $\Phi^+$:* Based on the information in $C$ and $R$, we form $\Phi$, which is a $c \times c$ matrix keeping the entries which are common in $C$ and $R$. Since we need its pseudo-inverse $\Phi^+$, the complexity of this step is $O(c^3)$, which involves computing a pseudo-inverse via the singular value decomposition (SVD) of $\Phi$.

(iii) *Spectral Decomposition:* We find the optimal rank-$d$ spectral reconstruction of the product $C\Phi^+R$, to embed in $d$-dimensions. The complexity of this step is $O(cd|V|)$, using the power iteration, which finds the largest eigenvalues of a matrix and its associated eigenvectors.

SSDE can be used to produce a $d$-dimensional embedding, the most practical being $d = 2,3$. We focus on $d = 2$ in this paper. We present the results of our algorithm through several examples, including run-times and embedding errors. Compared to similar techniques, we observe that our algorithm is fast enough to handle graphs up to $10^6$ nodes in about 10 seconds. A comparison of SSDE with two popular spectral graph drawing algorithms (HDE and ACE) is given in Figure 1: SSDE produces very good drawings of almost every mesh-like graph we have tried, with comparable or better running times. One of the main exceptions is tree-like graphs or more generally graphs with low algebraic connectivity, which are problematic for all three spectral graph drawing techniques mentioned.

The problem our algorithm addresses is that of embedding a finite metric space in $\mathbb{R}^2$ under the $l_2$-norm [19]. Most research in this area of mathematics has focused on

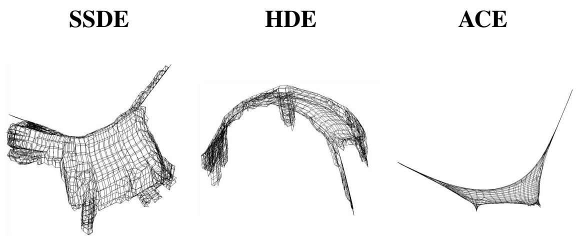


**Fig. 1.** Comparison of SSDE with other spectral methods (HDE and ACE) on the finite element mesh of a cow with $|V| = 1820, |E| = 7940$

determining what kinds of finite metric spaces are embeddable using low-distortion embeddings. Our work does not provide any guarantees on the distortion of the resulting embedding, which is an active area of research. We do, however, give the intuition behind why our algorithm constructs a good embedding using limited data on the distances between the points. Another paper using this kind of approach is [21], which introduces a different formulation via the Nystrom approximation.

## 1.1 Related Work

There are general methods to draw graphs, and detailed information about different approaches can be found in [13,22]. Our algorithm is based on spectral decomposition which yields the problem of computing the eigenvalues and eigenvectors of certain matrices related to the structure of the graph. The formulation is mathematically clean, in that exact solutions can be found, because eigenvectors and eigenvalues can be computed exactly in $O(|V|^3)$ time. Our work falls within the category of fast spectral graph drawing algorithms, which is the related work we elaborate upon.

High-Dimensional Embedding (HDE) described in [10] by Harel and Koren embeds the graph in a high dimension (typically 50) with respect to carefully chosen pivot nodes. One then projects the coordinates into two dimensions by using a well-known multivariate analysis technique called principal component analysis (PCA), which involves computing the first few largest eigenvalues and eigenvectors of the covariance matrix of the points in the higher dimension.

ACE (Algebraic multigrid Computation of Eigenvectors) [16] minimizes Hall's Energy function $E = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2$ in each dimension, modulo some non-degeneracy and orthogonality constraints ($n$ is the number of nodes, $x_i$ is the one-dimensional coordinate of the $i^{th}$ node and $w_{ij}$ is the weight of the edge between $i$ and $j$). This minimization problem can be reduced to obtaining the eigen-decomposition of the Laplacian of the graph. A multi-scaling approach is also used, creating coarser levels of the graph and relating them to the finer levels using an interpolation matrix.

Both of the methods described above are fast due to the small sizes of the matrices processed. Specifically, the running time of ACE depends on the structure of the graph while HDE provides better image quality and run-times. But, they may result in aesthetically unpleasant drawings of certain graphs and some of these problems are illustrated in Figure 1.

### 1.2 Notation

We use $i, j, k, \ldots$ for indices of vectors and matrices, bold uncapitalized letters $\mathbf{x}, \mathbf{y}, \mathbf{z}$ for vectors in $\mathbb{R}^d$ and bold capitalized letters for matrices. Typically, $\mathbf{M}, \mathbf{N}$ are used to represent $n \times n$ matrices and $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ for $n \times d$ matrices, which represent $n$ vectors in $\mathbb{R}^d$. $\mathbf{A}_{(i)}$ denotes the $i^{th}$ row of the matrix $\mathbf{A}$ and $\mathbf{A}^{(i)}$ denotes its $i^{th}$ column. The pseudo-inverse of a matrix $\mathbf{A}$ is denoted as $\mathbf{A}^+$. The norm of a vector $\|\mathbf{x}\|$ is the standard Euclidean norm. The transpose of a vector or a matrix is denoted as $\mathbf{x}^T, \mathbf{M}^T$.

## 2 Spectral Decomposition of the Distance Matrix and CMDS

Given a graph $G = (V, E)$ with $n$ nodes, let $V = \{v_1, v_2, \ldots, v_n\}$. The distance matrix $\mathbf{D}$ is the symmetric $n \times n$ matrix containing all the pair-wise distances, i.e., $D_{ij}$ is the length of the shortest path between $v_i$ and $v_j$. Suppose that the position at which vertex $v_i$ is placed is $\mathbf{x}_i$. We are seeking a positioning that approximates the graph theoretical distances with the Euclidean distances, i.e,

$$\|\mathbf{x}_i - \mathbf{x}_j\| \approx D_{ij}, \qquad \text{for } i, j = 1, 2, \ldots, n. \tag{1}$$

A suitable algebraic manipulation as presented in [4] on (1) yields the following equation, which is the basic idea of CMDS:

$$\mathbf{Y}\mathbf{Y}^T \approx -\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma} = \mathbf{M}. \tag{2}$$

where $\mathbf{Y}$ is an $n \times d$ matrix containing the coordinates of the points, $\mathbf{L}$ is the $n \times n$ matrix keeping the squares of the distances between nodes and $\boldsymbol{\gamma} = \frac{1}{n}\mathbf{I}_n - \mathbf{1}_n\mathbf{1}_n{}^T$ is a projection matrix. We want to approximate $\mathbf{M}$ as closely as possible. The metric that CMDS chooses is the spectral norm, so we wish to find the best rank-$d$ approximation to $\mathbf{M}$ with respect to the spectral norm. This is a well-known problem, which is equivalent to finding the largest $d$ eigenvalues of $\mathbf{M}$. The final centralized coordinates are then given by $\mathbf{Y} = [\sqrt{\lambda_1}\mathbf{u}_1, \ldots, \sqrt{\lambda_d}\mathbf{u}_d]$, where $\lambda_1, \ldots, \lambda_d$ are the first $d$ eigenvalues of $\mathbf{M}$ and $\mathbf{u}_1, \ldots, \mathbf{u}_d$ are the associated eigenvectors.

```
CMDS(G)
  1: Compute D using an APSP algorithm on G
  2: Define matrix L such that $L_{ij} = D_{ij}^2$.
  3: return Y = PowerIteration($-\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma}, \varepsilon$)
```

**Fig. 2.** The spectral graph drawing algorithm based on CMDS

Finding a rank-$d$ approximation of $\mathbf{M} = -\frac{1}{2}\boldsymbol{\gamma}\mathbf{L}\boldsymbol{\gamma}$, which corresponds to computing the largest $d$ eigenvalues and eigenvectors, is performed by a standard procedure typically referred to as the power iteration, rather than by an exact algorithm which would have $O(|V|^3)$ time complexity.

## 3 Approximate Distance Matrix Reconstruction

The running time of the CMDS technique is quadratic in terms of the number of nodes even for sparse graphs since one needs to compute and store all-pairs' shortest path lengths. In this section, we will first briefly explain the intuition behind SSDE, which breaks the quadratic complexity of this technique and actually yields a fast, linear time algorithm. Then, we will present the mathematical formulation.

### 3.1 Intuition

SSDE tries to construct an approximation to the distance matrix without computing all the entries in it. In the previous section, we noted that the distance matrix might have rank larger than $d$. But, the rank of the distance matrix is expected to be small in terms of the number of nodes in the graph, even if it is larger than $d$. This intuitive reasoning stems from a famous result from low distortion embedding theory. In 1984, Johnsson and Lindenstrauss [11] proved that $n$ points in high dimension can be embedded into $O(\frac{log\,n}{\varepsilon^2})$ dimensions with $\varepsilon$ distortion. This means, roughly speaking that the set of points can be reconstructed in low dimension while preserving all the pair-wise distances and hence that the effective rank of the distance matrix is much smaller than its full dimension. This suggests that one can extract much of the information about the matrix by performing computations on matrices having much smaller ranks.

Specifically, SSDE approximates the distance matrix with the product of three smaller matrices, which have linear size in terms of the number of nodes in the graph. In order to do this, reasoning from the fact that the distance matrix has low rank, the columns of the distance matrix can approximately be expressed as a linear combination of a small number of its columns. The algorithm essentially consists of choosing this small number of columns, constructing the whole matrix appropriately and computing the coordinates of the vertices via the spectral decomposition of this matrix. A variant of the particular approximation that we will use has been studied in [20]. In [20], the sampling approach used assumes that the whole matrix is known using one pass. Since, this would lead to a quadratic time algorithm, our approach must use online sampling. One can either sample the columns randomly or use a simple greedy algorithm, which seems to give a better set of columns.

### 3.2 Formulation

Let $i_1, i_2, \ldots, i_c$ be a set of distinct indices where $c$ is a predefined positive integer smaller than $n$ and $1 \leq i_k \leq n$ for $k = 1, \ldots c$. Let $\mathbf{C} = [\mathbf{L}^{(i_1)}, \mathbf{L}^{(i_2)}, \ldots, \mathbf{L}^{(i_c)}]$. If $\mathbf{C}$ is chosen carefully, under the assumptions mentioned above, any column $\mathbf{L}^{(i)}$ can approximately be written as a linear combination of the columns of $\mathbf{C}$, i.e.

$$\mathbf{L}^{(i)} \approx \mathbf{C}\boldsymbol{\alpha}^{(i)} \qquad \text{for } i = 1, 2, \ldots, n, \tag{3}$$

where $\boldsymbol{\alpha}^{(i)}$ is a $c \times 1$ vector. Denoting $\boldsymbol{\alpha} = [\boldsymbol{\alpha}^{(1)}, \boldsymbol{\alpha}^{(2)}, \ldots, \boldsymbol{\alpha}^{(n)}]$, we have

$$\mathbf{L} \approx \mathbf{C}\boldsymbol{\alpha} \tag{4}$$

Let $\boldsymbol{\Phi}$ be the $c \times c$ matrix such that $\boldsymbol{\Phi}_{jk} = \mathbf{L}_{i_j i_k}$ for $j,k = 1, \ldots c$. Note that since we also have $\mathbf{C}^T = [\mathbf{L}_{(i_1)}, \mathbf{L}_{(i_2)}, \ldots, \mathbf{L}_{(i_c)}]$, $\boldsymbol{\Phi}$ can be interpreted as the intersection of $\mathbf{C}$ and $\mathbf{C}^T$ on the matrix $\mathbf{L}$. Now, since the columns of $\mathbf{L}$ can approximately be expressed as a linear combination of the columns of $\mathbf{C}$, the columns of $\mathbf{C}^T$ can also be expressed as a linear combination of the columns of $\boldsymbol{\Phi}$. This gives

$$\mathbf{C}^T \approx \boldsymbol{\Phi}\boldsymbol{\alpha} \tag{5}$$

where $\boldsymbol{\alpha}$ is the same matrix as we defined above. If $\boldsymbol{\Phi}$ has full rank, (5) yields $\boldsymbol{\alpha} = \boldsymbol{\Phi}^{-1}\mathbf{C}^T$. Combining this with (4), we have $\mathbf{L} = \mathbf{C}\boldsymbol{\Phi}^{-1}\mathbf{C}^T$. More generally, we do not assume that $\boldsymbol{\Phi}$ has full rank, so we have

$$\mathbf{L} \approx \mathbf{C}\boldsymbol{\Phi}^{+}\mathbf{C}^T \tag{6}$$

where $\boldsymbol{\Phi}^{+}$ is the pseudo-inverse of $\boldsymbol{\Phi}$ (See [7] for the definition of pseudo-inverse). The last expression indicates that we can approximate the distance matrix $\mathbf{L}$ by the multiplication of three smaller matrices, which all have at most linear size in terms of $n$. Note that $\mathbf{C}$ is $n \times c$ and $\boldsymbol{\Phi}$ is $c \times c$.

## 4 The Algorithm SSDE

The algorithm SSDE, which uses the procedures that we will define shortly is summarized in Figure 5. As stated in the introduction, the algorithm consists of three main steps:

(1) *Sampling:* The first step of the algorithm is to compute the columns that define $\mathbf{C}$ and $\boldsymbol{\Phi}$. This is equivalent to choosing a particular set of nodes and computing the graph theoretical distances to all other nodes in the graph. We propose two methods to sample $c$ nodes:

   (i) *Random Sampling:* The $c$ nodes are sampled uniformly at random.

   (ii) *Greedy Sampling:* The first node is chosen uniformly at random. Then, at each step, we choose the furthest node to the set of nodes that have already been chosen until $c$ nodes are chosen.

   Note that, the second method stated above is also known to be a 2-approximation algorithm to the $k$-center problem [23]. This method was also used in [9] and [10]in different contexts. The procedure for performing these operations is presented in Figure 3. Even though $c$ can be treated as a parameter to the algorithm, we have experienced that setting $c = 25$ is enough for getting good results on practically all graphs we have tried. The sampling step, overall requires $O(c|E|)$ time as we initiate a BFS from $c$ nodes in the graph.

```
ComputeCandPhi(G, method, $c$)
 1: if method = random then
 2:     Select $c$ vertices uniformly at random
 3:     for $k = 1$ to $c$ do
 4:         $\mathbf{C}^k \leftarrow dist(i_k, V)$   // BFS
 5:     end for
 6: else if method = greedy then
 7:     $i_1 \leftarrow unifrnd(1, |V|)$ // Choose uniformly at random
 8:     $\mathbf{C}^1 \leftarrow dist(i_1, V)$   // BFS
 9:     for $k = 2$ to $c$ do
10:         $i_k \leftarrow \max_{1 \le j \le n} \min_{1 \le l \le k} \{\mathbf{C}_{jl}\}$  // Choose the furthest node
11:         $\mathbf{C}^k \leftarrow dist(i_k, V)$   // BFS
12:     end for
13: end if
14: Compute $\boldsymbol{\Phi}$   // $\boldsymbol{\Phi}_{kj} = \mathbf{C}_{i_k j}$
15: return $(\mathbf{C}, \boldsymbol{\Phi})$
```

**Fig. 3.** The procedure computing the matrices **C** and $\boldsymbol{\Phi}$

(2) *Computing $\Phi^+$:* We find the pseudo-inverse $\boldsymbol{\Phi}^+$ by first computing the singular value decomposition $\boldsymbol{\Phi} = \mathbf{U\Sigma V}^T$, which can be performed in $O(c^3)$ time using standard procedures (see for example [7]). The pseudo-inverse can then be computed by the expression $\boldsymbol{\Phi}^+ = \mathbf{V\Sigma}^+\mathbf{U}^T$. Here, $\boldsymbol{\Sigma}^+$ is the diagonal matrix keeping the reciprocals of the non-zero singular values, which are stored in $\boldsymbol{\Sigma}$. Unfortunately, in order to get numerically stable results, it is not enough to compute the reciprocals of the singular values, since the small singular values which are close to zero should actually be ignored, as they may be the result of numerical imprecision and will result in huge instability in $\boldsymbol{\Sigma}^+$. To prevent such instability, we use a *regularization* method that was presented in [18], which uses the expression

$$\frac{\sigma_i}{{\sigma_i}^2 + \alpha / {\sigma_i}^2} \tag{7}$$

for the reciprocals in $\boldsymbol{\Sigma}^+$, where $\sigma_i$ is the $i^{th}$ diagonal entry in $\boldsymbol{\Sigma}$. The parameter $\alpha$ is the regularization parameter, which must be chosen judiciously in order not to distort the reciprocals of the large singular values too much. On the other hand, it should result in values close to zero for the small singular values. Our experiments revealed that $\alpha = {\sigma_1}^3$ is good enough for practical purposes where $\sigma_1$ is the largest singular value. However, we keep it as a parameter of the procedure.

(3) *Spectral Decomposition:* Having computed the pseudo-inverse of $\boldsymbol{\Phi}$, we compute $\hat{\mathbf{L}} = \mathbf{C}\boldsymbol{\Phi}^+\mathbf{C}^T$ from which we obtain $\hat{\mathbf{M}} = -\frac{1}{2}\boldsymbol{\gamma}\hat{\mathbf{L}}\boldsymbol{\gamma}$. Then, analogous to (2), we obtain the coordinates of the points in the embedding using the spectral decomposition of $\hat{\mathbf{M}}$, which approximates $\mathbf{M}$. This requires computing the top $d$ eigenvalues and eigenvectors, for which we use a standard procedure called the power iteration

(See Figure 4). In the power iteration, the main computational task is to repetitively multiply a randomly chosen vector with the matrix whose eigenvalues and eigenvectors are sought. In our power iteration, starting from the right, the matrix-vector multiplications (line 5 and line 15) can be performed using $O(c|V|)$ scalar additions and multiplications. The total number of iterations until a predefined convergence condition holds, depends on the matrix processed. But, since the convergence is exponential (see for example [7]), in practice, a constant number of iterations is enough. Overall, the running time of the power iteration step of the algorithm is $O(c|V|)$.

```
PowerIteration($\mathbf{C}$, $\boldsymbol{\Phi}^+$, $\varepsilon$)
 1: $current \leftarrow \varepsilon$;  $\mathbf{y}_1 \leftarrow \mathbf{random}/\|\mathbf{random}\|$
 2: repeat
 3:    $prev \leftarrow current$
 4:    $\mathbf{u}_1 \leftarrow \mathbf{y}_1$
 5:    $\mathbf{y}_1 \leftarrow -\frac{1}{2}\boldsymbol{\gamma}\mathbf{C}\boldsymbol{\Phi}^+\mathbf{C}^T\boldsymbol{\gamma}\mathbf{u}_1$
 6:    $\lambda_1 \leftarrow \mathbf{u}_1 \cdot \mathbf{y}_1$   % compute the eigenvalue
 7:    $\mathbf{y}_1 \leftarrow \mathbf{y}_1/\|\mathbf{y}_1\|$
 8:    $current \leftarrow \mathbf{u}_1 \cdot \mathbf{y}_1$
 9: until $|current/prev| \leq 1+\varepsilon$
10: $current \leftarrow \varepsilon$;  $\mathbf{y}_2 \leftarrow \mathbf{random}/\|\mathbf{random}\|$
11: repeat
12:    $prev \leftarrow current$
13:    $\mathbf{u}_2 \leftarrow \mathbf{y}_2$
14:    $\mathbf{u}_2 \leftarrow \mathbf{u}_2 - \mathbf{u}_1(\mathbf{u}_1 \cdot \mathbf{u}_2)$   % orthogonalize against $\mathbf{u}_1$
15:    $\mathbf{y}_2 \leftarrow -\frac{1}{2}\boldsymbol{\gamma}\mathbf{C}\boldsymbol{\Phi}^+\mathbf{C}^T\boldsymbol{\gamma}\mathbf{u}_2$
16:    $\lambda_2 \leftarrow \mathbf{u}_2 \cdot \mathbf{y}_2$   % compute the eigenvalue
17:    $\mathbf{y}_2 \leftarrow \mathbf{y}_2/\|\mathbf{y}_2\|$
18:    $current \leftarrow \mathbf{u}_2 \cdot \mathbf{y}_2$
19: until $|current/prev| \leq 1+\varepsilon$
20: return $(\sqrt{\lambda_1}\mathbf{y}_1 \ \ \sqrt{\lambda_2}\mathbf{y}_2)$
```

**Fig. 4.** The power iteration method for finding eigenvectors and eigenvalues ($d = 2$)

The embedding is obtained directly from the eigenvectors and eigenvalues, which are returned by the power iteration.

## 5 Results

We have implemented our algorithm in C++, and Table 1 gives the running time results on a Pentium 4HT 3.0 GHz processor system with 1 GB of memory. We present the results of running the algorithm on several graphs of varying sizes up to about $2,000,000$ nodes. We set $c = 25$, since our experiments have revealed that this is enough to get good

```
SSDE(G, method)
1: $(\mathbf{C}, \boldsymbol{\Phi}) \leftarrow$ ComputeCandPhi(G, method, $c$)
2: $(\mathbf{U}, \boldsymbol{\Sigma}, \mathbf{V}^T) \leftarrow$ SVD($\boldsymbol{\Phi}$)
3: $\boldsymbol{\Sigma}^+ \leftarrow$ Regularize($\boldsymbol{\Sigma}$, $\alpha$)
4: $\boldsymbol{\Phi}^+ \leftarrow \mathbf{V}\boldsymbol{\Sigma}^+\mathbf{U}^T$
5: return $\mathbf{Y}$ = PowerIteration($\mathbf{C}, \boldsymbol{\Phi}^+, \varepsilon$)
```

**Fig. 5.** The spectral graph drawing algorithm SSDE

drawings. For the power iteration, we set the tolerance $\varepsilon = 10^{-7}$. The running times in Table 1 do not include the file I/O that is used to access and store the coordinates of the nodes. In Table 1, we present the results for CMDS and SSDE with $c = 25, 50$ and greedy sampling. Along with the running time, we also give the Frobenius norm of the relative error matrix for the embedding, $\boldsymbol{\varepsilon}$, where $\varepsilon_{ij} = 1 - D'_{ij}/D_{ij}$ and $\boldsymbol{D}, \boldsymbol{D'}$ are the true distance matrix and distance matrix implied by the embedding respectively. The normalized Frobenius errors computed in Table 1 are defined as

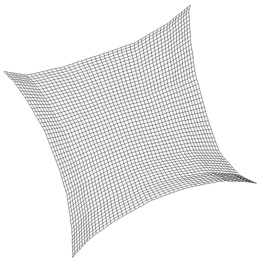
$$\|\boldsymbol{\varepsilon}_{F'}\| = \sqrt{\frac{1}{n^2}\sum_{i\neq j}(1 - \frac{D'_{ij}}{D_{ij}})^2}. \tag{8}$$

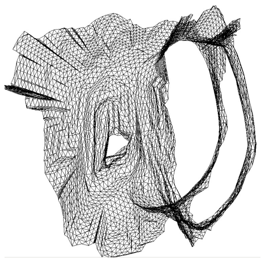
These errors might be interpreted as a quantification of the quality of the embedding, and can be used to compare SSDE to CMDS. As can be inferred from Table 1, SSDE is a good approximation to CMDS, which becomes more so as $c$ increases.

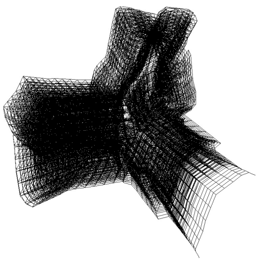
SSDE is able to draw graphs up to $10^6$ nodes in about ten seconds, which is comparable to the other fast spectral methods. The last three graphs in the table are road maps of states [1]. As is empirically verified from these graphs, the asymptotic running time of the algorithm is linear. Figure 6 demonstrates the quality of the drawings for some benchmark graphs. In all the graphs except finan512, we used the greedy sampling method. Random sampling seems to work better for finan512 because of its special structure. We have observed that the algorithm is able to reveal the general structure of almost all the graphs we tested, as well as the finer structure of some of the graphs successfully, where other spectral methods have difficulty. An example is the finan512 graph, where the overall structure is clearly visible, and one can also see the finer structure of the small "towers" attached to the main cycle. Figure 7 compares the results of the exact algorithm CMDS, and SSDE, which is approximate but far more efficient. We demonstrate the results of SSDE for both random and greedy sampling. The figure shows that SSDE does not sacrifice much in the way of picture quality as compared to CMDS. For all the drawings mentioned, it is important to note that exact pictures may change depending on which specific nodes are sampled, but the typical structure is consistent. The quality of the drawing for random and greedy sampling also doesn't differ much, but our experiments showed that the greedy sampling tends to give more consistent results.

**Table 1.** Running time and embedding errors of CMDS and SSDE for several graphs. (Most of these graphs can be downloaded from [1], [2] and [3]). Missing entries are graphs where it was too costly to compute the entire distance matrix.

| Graph | \|V\| | \|E\| | CMDS | | SSDE(c=25) | | SSDE(c=50) | |
|---|---|---|---|---|---|---|---|---|
| | | | $\|\|\varepsilon_{F'}\|\|$ | Time(sec) | $\|\|\varepsilon_{F'}\|\|$ | Time(sec) | $\|\|\varepsilon_{F'}\|\|$ | Time(sec) |
| **3elt** | 4720 | 13722 | 0.382 | 8.47 | 0.432 | 0.015 | 0.398 | 0.04 |
| **sierpinski08** | 9843 | 19683 | 0.17 | 24.72 | 0.203 | 0.03 | 0.19 | 0.07 |
| **Grid 100x100** | 10000 | 19800 | 0.17 | 29.73 | 0.192 | 0.03 | 0.186 | 0.06 |
| **crack** | 10240 | 30380 | 0.085 | 45.00 | 0.103 | 0.045 | 0.09 | 0.10 |
| **4elt2** | 11143 | 32818 | 0.252 | 48.77 | 0.291 | 0.07 | 0.283 | 0.14 |
| **4elt** | 15606 | 45878 | 0.308 | 133.33 | 0.375 | 0.13 | 0.342 | 0.25 |
| **sphere** | 16386 | 49152 | 0.291 | 136.69 | 0.334 | 0.14 | 0.312 | 0.27 |
| **finan512** | 74752 | 261120 | - | - | - | 0.68 | - | 1.43 |
| **ocean** | 143437 | 409593 | - | - | - | 1.65 | - | 3.56 |
| **144** | 144649 | 1074393 | - | - | - | 2.85 | - | 6.03 |
| **wave** | 156317 | 1059331 | - | - | - | 2.40 | - | 4.78 |
| **auto** | 448695 | 3314611 | - | - | - | 9.96 | - | 21.67 |
| **Florida** | 1048506 | 1330551 | - | - | - | 10.04 | - | 23.45 |
| **California** | 1613325 | 1989149 | - | - | - | 17.91 | - | 36.13 |
| **Texas** | 2073870 | 2584159 | - | - | - | 21.69 | - | 45.89 |


(a)


(b)


(c)

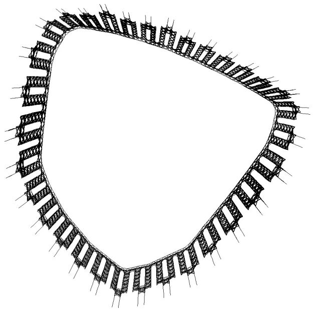

(d)

**Fig. 6.** Layouts of (a) 50x50 grid with $|V| = 2500, |E| = 4900$, (b) 3elt with $|V| = 4720, |E| = 13722$, (c) cti with $|V| = 16840, |E| = 48232$, (d) finan512 with $|V| = 74752, |E| = 261120$

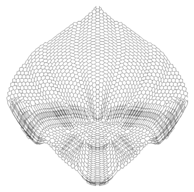
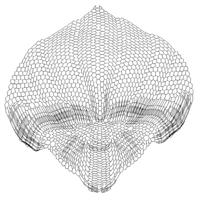
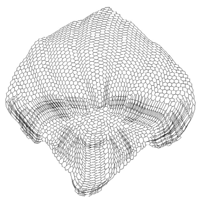
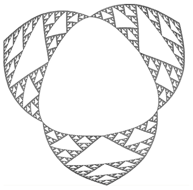
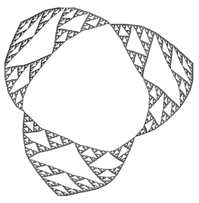
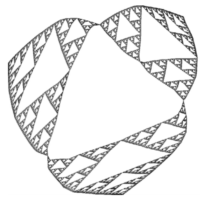
| | **CMDS** | **SSDE** Greedy | **SSDE** Random |
|---|---|---|---|
| 4970 $\vert V\vert = 4970$ $\vert E\vert = 7400$ |  |  |  |
| | running time = 5.04 sec. | running time = 0.01 sec. | running time = 0.01 sec. |
| sierpinski08 $\vert V\vert = 9843$ $\vert E\vert = 19683$ |  |  |  |
| | running time = 24.72 sec. | running time = 0.03 sec. | running time = 0.03 sec. |

**Fig. 7.** Comparison of pure CMDS and SSDE

## 6 Conclusion and Future Work

We have presented a fast spectral graph drawing algorithm, which significantly improves the idea of Classical Multi-Dimensional Scaling (CMDS), by using sampling techniques over nodes to reduce the time complexity of computing the distance matrix. We use a sparse approximation to the distance matrix obtained through sampling. The spectral decomposition of this sampled matrix yields the desired embedding. The running time of our algorithm is mainly governed by the shortest path computations for the sampled nodes and the power iteration procedure where we compute the coordinates of the points via the spectral decomposition, which in total is linear in the size of the graph. SSDE gives competitive running times with very good drawings for a broad range of graphs, and at the same time it does not sacrifice quality as compared to CMDS.

The typical graphs for which SSDE is not suited are graphs with low algebraic connectivity (such as trees for which special purpose algorithms exist) and dense graphs which are difficult to visualize anyway. Usually, as the graph gets denser, the sampled nodes cannot extract enough information about the spectrum of the distance matrix. We would like to mention that this shortcoming of SSDE applies to many real world graphs. However, these are issues faced by all the fast spectral methods discussed here.

The rigorous mathematical analysis of sampling methods and specifically their implications on the error of the difference between the real distance matrix and the approximation is the context of future work. The sampling step intuitively tries to pick a set of columns whose volume in $|V|$ dimensions is as large as possible, which implies a better approximation to the distance matrix. An interesting problem would be to consider the performance of greedy sampling with respect to the optimal choice of samples.

## References

1. http://www.dis.uniroma1.it/~challenge9/data/tiger/.
2. http://wwwcs.uni-paderborn.de/fachbereich/AG/monien/RESEARCH/PART/graphs.html.
3. http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/.
4. I. Borg and P. Groenen. *Modern Multidimensional Scaling*. Springer-Verlag, 1997.
5. A. Çivril, M. Magdon-Ismail, and E. Bocek-Rivele. SDE: Graph drawing using spectral distance embedding. In *GD'05*, pages 512–513, 2005.
6. T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice And Experience*, 21(11):1129–1164, 1991.
7. G. H. Golub and C.H. Van Loan. *Matrix Computations*. Johns Hopkins U. Press, 1996.
8. K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17: 219–229, 1970.
9. D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. In *GD'00*, volume 1984, pages 183–196, 2000.
10. D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *GD'02*, 2002.
11. W. Johnson and J. Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemp. Math.*, 26:189–206, 1984.
12. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.
13. M. Kaufmann and D. Wagner, editors. *Drawing Graphs: Methods and Models*. Number 2025 in LNCS. Springer-Verlag, 2001.
14. Y. Koren. On spectral graph drawing. In *COCOON 03*, volume 2697, pages 496–508, 2003.
15. Y. Koren. One dimensional layout optimization, with applications to graph drawing by axis separation. *Computational Geometry: Theory and Applications*, 32:115–138, 2005.
16. Y. Koren, D. Harel, and L. Carmel. Drawing huge graphs by algebraic multigrid optimization. *Multiscale Modeling and Simulation*, 1(4):645–673, 2003. SIAM.
17. J. B. Kruskal and J. B. Seery. Designing network diagrams. In *Proc. First General Conference on Social Graphics*, 1980.
18. J. Maeda and K. Murata. Restoration of band-limited images by an iterative regularized pseudoinverse method. *Journal of Optical Society of America*, 1(1):28–34, 1984.
19. J. Matousek. Open problems on embeddings of finite metric spaces. *Discr. Comput. Geom.*, to appear.
20. P.Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo algorithms for matrices III: Computing a compressed approximate matrix decomposition. *SIAM Journal on Computing*, 36(1):184–206, 2006.
21. J. C. Platt. FastMap, MetricMap, and landmarkMDS are all Nystrom algorithms. In *Proc. 10th Int. Workshop on Artificial Intelligence and Statistics*, pages 261–268, 2005.
22. I. G. Tollis, G. Di Battista, P. Eades, and R. Tamassia. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
23. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.
24. C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *GD'00*, volume 1984, 2000.

# Eigensolver Methods for Progressive Multidimensional Scaling of Large Data

Ulrik Brandes and Christian Pich

Department of Computer & Information Science, University of Konstanz, Germany
{Ulrik.Brandes,Christian.Pich}@uni-konstanz.de

**Abstract.** We present a novel sampling-based approximation technique for classical multidimensional scaling that yields an extremely fast layout algorithm suitable even for very large graphs. It produces layouts that compare favorably with other methods for drawing large graphs, and it is among the fastest methods available. In addition, our approach allows for progressive computation, i.e. a rough approximation of the layout can be produced even faster, and then be refined until satisfaction.

## 1 Introduction

The term multidimensional scaling (MDS) refers to a family of techniques for dimensionality reduction that are used to represent high-dimensional data in low-dimensional space while approximately preserving distances. For drawing graphs, methods based on the objective function of *distance scaling* are used widely, but the *classical scaling* approach has only occasionally been recognized as a useful alternative [7,21,24]. Indeed the computational complexity of this method is quadratic in the input size and thus prohibitive for large graphs.

In this paper we propose a sampling-based approximation technique to overcome this restriction and to reduce time and space complexity essentially to linearity. The proposed algorithm is simple to implement, yet extremely fast and therefore applicable to very large graphs. Moreover, it allows for progressive computation by very quickly producing a rough approximation of the layout, which can then be improved by successive refinement.

This paper is organized as follows. Background on multidimensional scaling and derived methods is provided in Section 2. In Section 3 we introduce two variants of the eigensolver approach, which are evaluated and compared to each other in Section 4. Section 5 concludes our contribution.

## 2 Related Work

The first MDS algorithm is due to Torgerson [30] and nowadays referred to as *classical MDS* or *classical scaling*. Its objective is a low-dimensional representation of high-dimensional data by fitting inner products; it has a global optimum which can be directly computed by spectral decomposition. The method we propose in this paper is an efficient approximation of classical scaling.

Another MDS variant best known and most widely used today has been proposed by Kruskal [22] and is sometimes distinguished as *distance scaling*. The objective is to directly fit Euclidean distances in the drawing to the given graph-theoretical distances, typically by minimizing a stress measure. It is performed by iterative replacement according to a spring model of attracting and repelling forces or an energy model, as widely known in the graph drawing literature [18], or by iterative algebraic techniques [12]. Due to their time and space complexity, straightforward implementations of distance scaling methods are restricted to data sets of moderate cardinality.

In the graph drawing literature, methods based on linear algebra have become popular in recent years. Examples are High-Dimensional Embedding (HDE) [15], fast multiscale approaches based on eigenvectors of the Laplacian [20], subspace-restricted layout [19], and stress majorization [12].

Poor scalability to large data sets due to quadratic complexity is a well-known problem of all MDS algorithms. It was addressed as early as in the 1960s [23], and since then, many approaches to speeding up spring-force computations have been devised [6,16,25,26]. Likewise, methods for speeding up the spectral methods have been proposed [11,31]. Closest to our approach is Landmark MDS [10]; we give an experimental comparison in Sect. 4. Relationships between these approaches are discussed in [2,27]. For more general surveys on sparse techniques for dimensionality reduction and related spectral methods see [5,28].

MDS seems to have been the first computerized layout method used for drawing social networks [17] towards the end of the 1960s. Even in this restricted application domain there are many extensions and variants, such as incremental or interactive MDS [1,4,8,32]. For further information about MDS, its history, and other applications we refer the reader to recent textbooks [3,9].

## 3 Multidimensional Scaling and Its Approximation

Let $\Delta \in \mathbb{R}^{n \times n}$ denote a symmetric matrix of metric *dissimilarities* or *distances* $\delta_{ij}$ between items $i, j \in \{1, \ldots, n\}$. The goal of multidimensional scaling is to find positions $x_i \in \mathbb{R}^d$ in $d$-dimensional space, $d \ll n$, such that $\|x_i - x_j\| \approx \delta_{ij}$, i.e. distances are represented well in this low-dimensional space. Note that for notational convenience we write positions $x_i$ as column vectors, and that $d \in \{2, 3\}$ for visualization purposes. With $\Delta^{(2)}$ we denote matrix $\Delta$ with squared entries, i.e. $[\Delta^{(2)}]_{ij} = [\Delta]_{ij}^2$.

In graph drawing and network analysis, $\Delta$ frequently consists of shortest-path distances (see, e.g., [8] for an alternative graph distance). In other contexts it is often induced by a high-dimensional feature space with an associated distance function.

In this section, we briefly describe a standard technique for multidimensional scaling, a recently introduced method for its fast approximation, and our new variant of this approximation. It turns out that, technically, our method is very similar to one of the fastest algorithms for drawing large graphs [15], but eliminates some of its shortcomings. This is outlined in Sect. 3.5.

## 3.1 Classical MDS

We briefly describe the scaling method known as Classical MDS [30]. Recall that we are looking for an embedding in $d$-dimensional space, i.e. a matrix $X \in \mathbb{R}^{n \times k}$ with $X = [x_1, \dots, x_n]^T$, such that $\delta_{ij} \approx \|x_i - x_j\|$. Since this implies

$$\delta_{ij}^2 \approx \|x_i - x_j\|^2 = (x_i - x_j)^T(x_i - x_j) = x_i^T x_i - 2x_i^T x_j + x_j^T x_j,$$

consider the matrix $B = XX^T$ of inner products $b_{ij} = x_i^T x_j$. While we do not know $X$, it can be shown that

$$b_{ij} = -\frac{1}{2}\left(\delta_{ij}^2 - \frac{1}{n}\sum_{r=1}^{n}\delta_{rj}^2 - \frac{1}{n}\sum_{s=1}^{n}\delta_{is}^2 + \frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n}\delta_{rs}^2\right),$$

so that $B$ can also be obtained by double-centering the matrix of squared dissimilarities $\Delta^{(2)}$, i.e. each column and each row of $B$ sums to zero.

Knowing $B$, positions $X$ are reasonably reconstructed using the eigendecomposition $B = V\Lambda V^T$, where $\Lambda$ is the diagonal matrix of the eigenvalues of $B$, and $V$ is the orthonormal matrix of its eigenvectors. Simply let

$$X = V(d)\Lambda_{(d)}^{1/2} ,$$

where $\Lambda_{(d)} \in \mathbb{R}^{d \times d}$ is the diagonal matrix of the $d$ largest eigenvalues of $B$ and $V(d) \in \mathbb{R}^{n \times d}$ is an $n \times d$ matrix of associated eigenvectors. Thus, the essence of classical scaling is to fit inner products rather than distances as in distance scaling.

It is important to note that the two or three required eigenvectors can be computed by power iteration, i.e. by repeatedly multiplying a starting vector $x \in \mathbb{R}^n$ with $B$. The iterate is periodically normalized; further eigenvectors are found by orthogonalization against previously computed eigenvectors. See, e.g., [13] for background on matrix computations.

The running time for drawing an unweighted graph with $n$ vertices and $m$ edges by performing classical MDS on its matrix $\Delta$ of shortest-path distances is thus $\mathcal{O}(nm)$ for computing $\Delta$ using breadth-first search, $\Theta(n^2)$ for constructing $B$, and another $\mathcal{O}(n^2)$ per iteration. Running times and also storage requirements are therefore prohibitive for large graphs.

## 3.2 Landmark MDS

Landmark MDS (LMDS) [10] is a fast method for approximating the results of Classical MDS using a sparsification of the transformed distance matrix. It is based on distinguishing a few items as *landmarks*, and computing the eigendecomposition only on the double-centered matrix of squared distances among those landmarks. Positions of non-landmarks are then determined as linear combinations of landmark positions, i.e. items are placed in the weighted barycenter of all landmarks where the weights are derived from the original distances.

The rationale is that a set of appropriate reference points is sufficient to determine the projection into low-dimensional space. To be representative, the $k$ landmarks, $d < k \ll n$, should be distributed well. Common experience shows that a *MaxMin* strategy, in which the next landmark maximizes the minimum distance to the previous landmarks, yields satisfactory results. Note that this corresponds to a well-known 2-approximation of the $k$-center problem in facility location. We have tried other simple strategies such as *MaxSum*, random selection, and hybrids, but none proved to be superior consistently. More advanced techniques are proposed in [29].

Time and space complexity of LMDS are significantly smaller than for Classical MDS. Landmark selection and distance computations are carried out in $\mathcal{O}(k \cdot |E|)$ time, each power iteration step requires only $\mathcal{O}(k^2)$ time, and the final positioning is done in $\mathcal{O}(kn)$ time. Since, in general, choosing $k < 100$ yields satisfactory results on most practical instances, LMDS can be regarded a linear-time algorithm. Moreover, it is only necessary to store the $\Theta(kn)$ distances to landmarks.

## 3.3 Pivot MDS

We now introduce a new variant of sparse MDS which we call *Pivot MDS* (PMDS). It is motivated by a potential shortcoming of the LMDS strategy to position landmarks only with respect to each other: it is possible that the (already available) distance information to non-landmarks can be utilized to improve the quality of the result.

Recall that Classical MDS is based on an eigendecomposition of the double-centered $n \times n$-matrix of squared distances $B$, and that Landmark MDS is based on the corresponding decomposition of the double-centered $k \times k$-submatrix of squared distances among selected items only. Pivot MDS is based on the double-centered $n \times k$-submatrix $C$ of squared distances from every item to those selected, having entries

$$c_{ij} = -\frac{1}{2}\left(\delta_{ij}^2 - \frac{1}{n}\sum_{r=1}^{n}\delta_{rj}^2 - \frac{1}{k}\sum_{s=1}^{k}\delta_{is}^2 + \frac{1}{nk}\sum_{r=1}^{n}\sum_{s=1}^{k}\delta_{rs}^2\right),$$

where $i \in \{1,\ldots,n\}$ and $j \in \{1,\ldots,k\}$, and thus contains all distance information available.

Note that the $n$-dimensional left singular vectors of $C \in \mathbb{R}^{n\times k}$ are equal to the eigenvectors of $CC^T \in \mathbb{R}^{n\times n}$. If they are computed using power iteration, an iteration consists of two steps: first, positions of pivots are determined using the current positions of all items (multiplication with $C^T \in \mathbb{R}^{k\times n}$), and then all items are positioned relative to the pivots (multiplication with $C \in \mathbb{R}^{n\times k}$).[1]

[1] This interpretation motivates the name "pivot," in contrast to "landmarks" which are first assigned their final location and then used to determine the position of all other items.

An intuitive interpretation is that the eigenvectors of $CC^T$ approximate the eigenvectors of $B^2$, and thus of $B$. This follows from the assumption

$$\left[B^2\right]_{ij} = \left[BB^T\right]_{ij} = \sum_{\ell=1}^{n} b_{i\ell} b_{j\ell} \approx \sum_{\ell=1}^{k} c_{i\ell} c_{j\ell} = \left[CC^T\right]_{ij},$$

so matrix entries $[B^2]_{ij}$ and $[CC^T]_{ij}$ represent the same type of transformed distance sums, though in the latter case with a truncated list of intermediaries. If these are sufficiently well distributed, the relative size of entries in $CC^T$ is representative for those in $B^2$.

At face value the iteration time of PMDS is $\mathcal{O}(kn)$. However, we can rewrite $(CC^T)^i = C(C^TC)^{i-1}C^T$ so that the iteration is performed only on the $k \times k$-matrix $C^TC$. The initial multiplication with $C^T$ can be omitted (in Sect. 3.4 we will argue, though, that it is sometimes desirable), since the starting vector is arbitrary. The final multiplication with $C$ is similar to the final projection step of LMDS. The algorithm is summarized in Alg. 1.

Except for the additional $\mathcal{O}(kn+k^2n)$ cost of double-centering and computing $C^TC$, the running time is therefore essentially the same as in LMDS.

**Algorithm 1**: Pivot MDS

```
Input: undirected graph $G = (V, E)$, number $k \in \mathbb{N}$ of pivots
Output: coordinates $x, y \in \mathbb{R}^n$

select $k$ pivots from $V$
for $i \in \{1, \ldots, k\}$ do
    $i$-th column of $\Delta(k) \leftarrow$ BFS($i$-th pivot)
$C \leftarrow$ doublecenter$\left(\Delta(k)^{(2)}\right)$
$(v_1, v_2) \leftarrow$ poweriterate$(C^TC)$      // 2 largest eigenvectors
$x \leftarrow Cv_1, y \leftarrow Cv_2$
```

## 3.4 Progressive MDS

When using pivot approximation there is a natural trade-off between running time and memory usage; users might have to experiment with various numbers of pivots and different strategies. Instead of iteratively re-executing the algorithm with a larger set of pivots for layout improvement, we propose to use a progressive form of MDS computation that we shall describe in the following.

Let $\Delta(k) \in \mathbb{R}^{n \times k}$ denote a submatrix of the matrix of pairwise distances, and let $x \in \mathbb{R}^n$ be a component in the placement computed by PMDS based on it. To improve approximation quality, $\Delta(k)$ can be extended by a certain number of new pivot columns to $\Delta(k') \in \mathbb{R}^{n \times k'}$ $(k' \geq k)$. Note that all operations for computing the new columns in $\Delta(k')$, double-centering of $\Delta(k')^{(2)}$ to obtain $C'$, and determination of matrix ${C'}^T C'$ can be implemented to run in $\mathcal{O}\left((k'-k) \cdot |E|\right)$. The new vector $x' \in \mathbb{R}^n$ is computed by replacing $C$ with $C'$ in Algorithm 1.

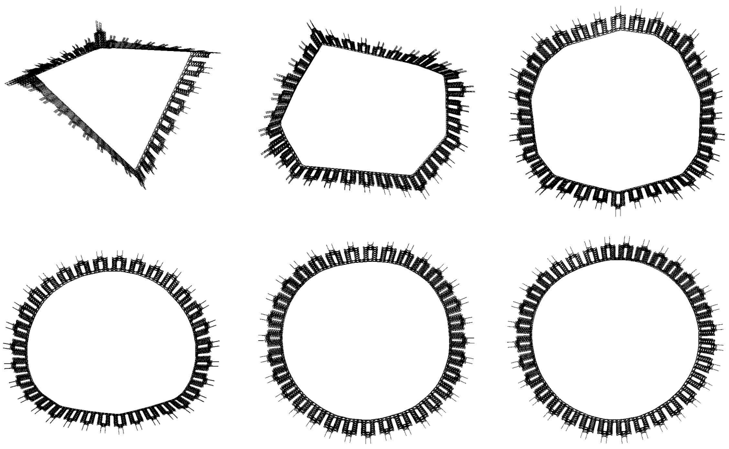


**Fig. 1.** Progressively drawing the finan512 graph ($|V| = 74752, |E| = 261120$) with increasing pivot set ($k = 3, 6, 12, 25, 50, 100$) using the minmax strategy

To prevent artificial effects through rotation and reflection in the transition from $x$ to $x'$ due to indeterminacies in the basis of the eigenspace of ${C'}^T C'$, the initial solution $y \in \mathbb{R}^k$ for the power iteration is derived from the previous layout by $y = {C'}^T x$. Compared to random initialization, the iteration process for computing the new layout $x'$ is thus more likely to converge towards a solution close to $x$, and we have observed that transitions between intermediate layouts tend to become smoother and visually more pleasing.

For smaller graphs, pivots may be added in batches before computing the layout, while it can make more sense for very large graphs to extend $\Delta(k)$ column by column, after each insertion computing the layout anew. In Sect. 4 our experiments indicate that most of the running time of Pivot MDS is consumed by the distance computations, while a layout based on these distances can be computed quickly. It is thus worthwhile to progressively compute the layout until the quality does not improve significantly.

### 3.5 Pivot MDS vs. HDE

In retrospect, our proposed method is reminiscent of another fast algorithm for drawing large graphs, the high-dimensional embedder (HDE) of [15].

HDE proceeds as follows: From a set of $k$ selected nodes (the pivots), distances to all other nodes are determined. These distances are, however, neither squared nor double-centered, but directly interpreted as coordinates in a $k$-dimensional space. In this space, they are centered to place the mean at zero coordinate, and yield a high-dimensional embedding $X \in \mathbb{R}^{n \times k}$. This $k$-dimensional embedding is then projected into two dimensions by Principal Component Analysis (PCA), i.e. by computing the two largest eigenvectors of the covariance matrix $\frac{1}{n}X^TX \in \mathbb{R}^{k\times k}$. The final coordinates are then obtained by matrix multiplication with $X$ analog to PMDS.

While this appears technically similar to PMDS, it is important to note that both approaches are motivated by different intuitions and produce different results: HDE transforms $k$ possibly correlated variables (the embedding $X$) into two uncorrelated variables (the layout). In contrast, the objective of PMDS is to directly find low-dimensional coordinates with inner products complying with the given dissimilarities $\delta_{ij}$. More details about the fundamental differences of the two approaches and experiments can be found in [21].

Both PMDS and HDE have approximately the same running time complexity of $\mathcal{O}(k \cdot |E| + k^2n)$, while PMDS appears to yield drawings of superior quality.

## 4 Evaluation

The algorithms were implemented in Java SDK 1.4.1. All experiments were conducted under MS Windows XP Version 2002 SP2 on an Intel Pentium-M CPU with 1.6GHz and 512MB of main memory.

We used a set of test graphs for drawing and for evaluating the scalability of our approach. We measured the CPU running times for distance computation and the layout algorithm. Descriptions of the test graphs are given in [14,15].

### 4.1 Running Time

Figure 2 shows for both Pivot and Landmark MDS that the running time for the breadth-first searches for matrix $C$ in $\mathcal{O}(km)$ time indeed dominates over the computation times for spectral decomposition of $C^TC$ and the final coordinates, which together are in $\mathcal{O}(k^3 + k^2n)$ time. The larger the graph and the smaller $k$ in relation to $n$, the more apparent this effect becomes. LMDS is slightly faster than Pivot MDS because it does not require the construction of $C^TC$.

We have used straightforward, non-optimized implementations for distance computations and matrix operations. Therefore, we expect that using specialized libraries with sophisticated algorithms and data structures yields significant improvements on the absolute values of the measured times.

One important consequence from our observation is that the number of pivots used for the approximation and the pivot strategy can be crucial for the ratio between quality and running time. The next subsection gives more details on the quality of the approximation relative to (full) Classical MDS.

### 4.2 Quality

To assess their approximation quality, we compared the approximated layouts with those given by full Classical MDS. *Procrustes analysis* (see, e.g., [9]), a technique popular in data analysis and statistics, is used to assess how similar two configurations $X, Y \in \mathbb{R}^{n\times d}$ with $X = [x_1, \ldots, x_n]^T, Y = [y_1, \ldots, y_n]^T$ are up to translation, dilation, and rotation. It is the sum of squared distances

| name | $\lvert V\rvert$ | $\lvert E\rvert$ | BFS | layout | total |
|---|---|---|---|---|---|
| ug380 | 1104 | 3231 | 0.05 | 0.03 | 0.08 |
| fidap006 | 1651 | 23914 | 0.16 | 0.01 | 0.17 |
| esslingen1 | 2075 | 4769 | 0.07 | 0.01 | 0.08 |
| 3elt | 4720 | 13722 | 0.22 | 0.05 | 0.27 |
| power | 4941 | 6594 | 0.17 | 0.05 | 0.22 |
| add32 | 4960 | 9462 | 0.15 | 0.02 | 0.17 |
| bcsstk33 | 8738 | 291583 | 1.96 | 0.05 | 2.01 |
| whitaker3 | 9800 | 28989 | 0.34 | 0.04 | 0.38 |
| crack | 10240 | 30380 | 0.45 | 0.05 | 0.50 |
| 4elt2 | 11143 | 32818 | 0.41 | 0.06 | 0.47 |
| 4elt | 15606 | 45878 | 0.78 | 0.08 | 0.86 |
| sphere | 16386 | 49152 | 0.81 | 0.09 | 0.90 |
| fidap011 | 16614 | 537374 | 3.52 | 0.08 | 3.60 |
| bcsstk31 | 35588 | 572914 | 4.36 | 0.19 | 4.54 |
| bcsstk32 | 44609 | 985046 | 7.09 | 0.25 | 7.34 |
| finan512 | 74752 | 261120 | 4.11 | 0.40 | 4.50 |
| ocean | 143437 | 409593 | 10.24 | 0.82 | 11.06 |

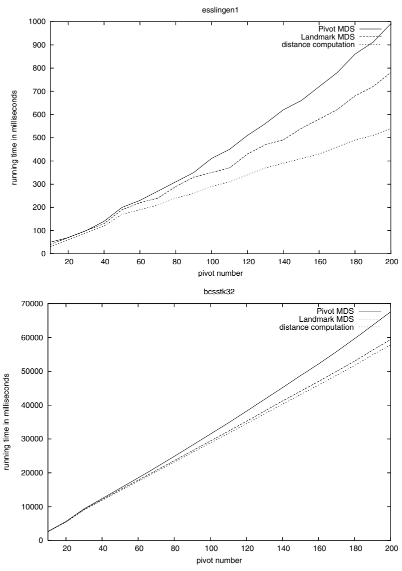


**Fig. 2.** Left: Pivot MDS running times in seconds using 50 pivots, measured for the set of test graphs. Right: Total running times required by distance computations, Pivot MDS, and Landmark MDS (the latter two including distance computation) with increasing pivot numbers, as measured for ESSLINGEN1 and BCSSTK32.

$$R^2 = \sum_{i=1}^{n} (x_i - y_i)^T (x_i - y_i) \,,$$

where both configurations consist of two-dimensional coordinates (i.e., $d = 2$). Procrustes analysis translates, dilates, and rotates $X$ such that $R^2$ is minimized with respect to $Y$. It can be shown (see, e.g., [3]) that $0 \leq R^2 \leq 1$ and that the minimum value is given by the *Procrustes statistic*

$$R^2 = 1 - \frac{\left(\mathrm{tr}(X^T Y Y^T X)^{1/2}\right)^2}{\mathrm{tr}(X^T X)\mathrm{tr}(Y^T Y)} \,,$$

which is the sum the squared distances between $X$ after the best possible transformation (with respect to $Y$), and $Y$. If the two configurations can be perfectly matched, $R^2 = 0$; if they cannot be matched at all by any transformation, $R^2 = 1$. We may assume that both configurations have the centroid in the origin.

We computed the $R^2$ value for the ESSLINGEN1 graph with Pivot and Landmark MDS, using the maxmin and the random pivot strategy, as depicted in Figure 3 with respect to the layout by full MDS. It can be seen that our method is almost consistently superior to Landmark MDS and that it seems to give more stable results. An interesting observation for both algorithms is that using the minmax pivot strategy yields good results with a small number of pivots, while,

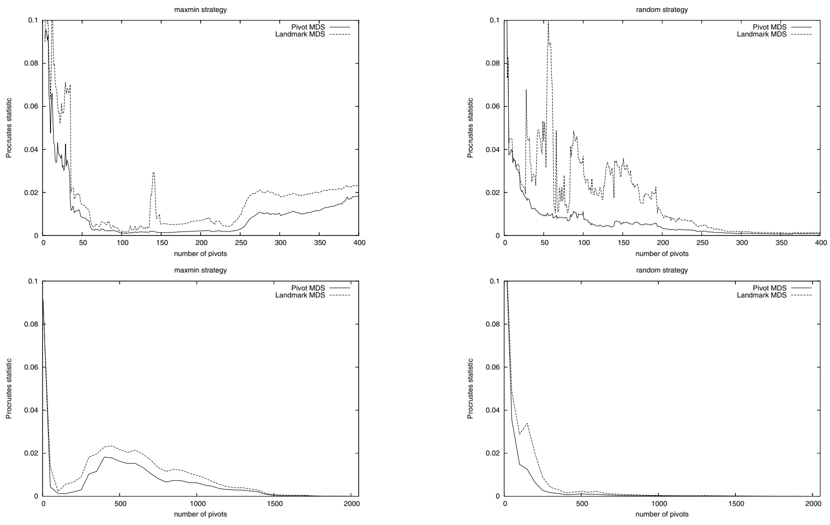


**Fig. 3.** Procrustes statistic vs. number of pivots for ESSLINGEN1. Upper row: Quality of PMDS and LMDS for practical use ($3 \leq k \leq 400$). Lower row: the same for the full scope ($3 \leq k \leq n$, larger step size in the plot). All curves reach 0 at $k = n$.

starting from a certain point, systematic pivot selection creates an unbalanced approximation leading to deterioration of quality. In contrast, using a random pivot strategy initially requires a larger number of pivots to obtain the same approximation quality, but displays a more monotonic behavior.

As the Procrustes statistic can be computed efficiently, it is suitable for comparing intermediate layouts when increasing the number of pivots, and may be used as a termination criterion. Progression may be stopped when the value of $R^2$ for consecutive layouts falls below a given threshold, indicating that little to no quality improvement can be expected by adding more pivots.

It is important to note that there are graphs for which Classical MDS (even without approximation) may be of poor quality due to the fact that the two dimensions in the layout are not sufficient for expressing the higher-dimensional structure of the data. In contrast, graphs with a very regular structure, such as finite-element meshes, often have a direct relation between coordinates in a low-dimensional space and graph-theoretical distances, and therefore almost surely yield useful layouts.

This is frequently referred to as the *intrinsic dimensionality* of the data. It can be estimated by the eigenvalue distribution: Few large positive and a large number of "almost zero" (hence rather uninformative) eigenvalues suggest a small number of intrinsic dimensions (which can be captured in a low-dimensional representation well); many large positive eigenvalues indicate a high intrinsic dimensionality and that there is little hope to get a feasible low-dimensional layout with any distance-based method.

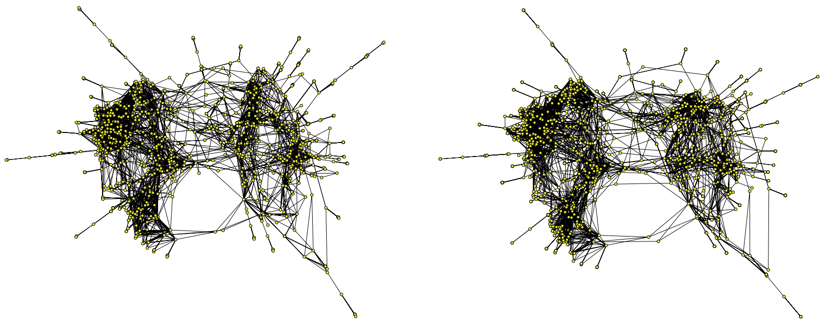


**Fig. 4.** Layouts of the ESSLINGEN1 graph using Pivot MDS approximation with 50 pivots (left), and by full Classical MDS (or, equivalently, 2075 pivots). The Procrustes statistic yields $R^2 = 0.0085$, indicating an excellent "fit".

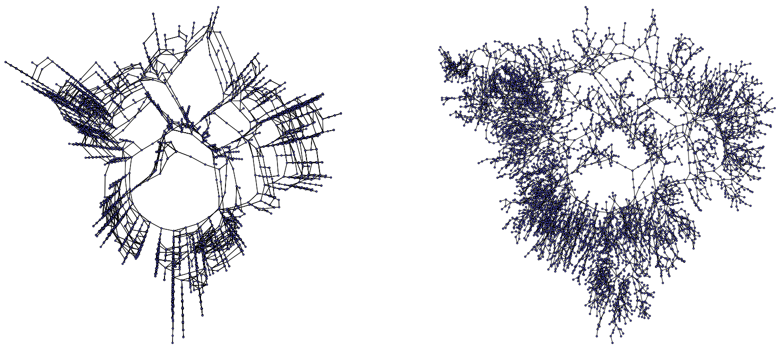


**Fig. 5.** The US power grid graph ($|V| = 4941, |E| = 6594$). Left: Pivot MDS using 100 pivots. Right: The same after postprocessing by a spring embedder. Pivot MDS appears to give a better layout of the grid structure, while the spring embedder displays regional density better. This suggests the use of our method for efficient generation of initial placements for further processing, which is crucial for many algorithms.

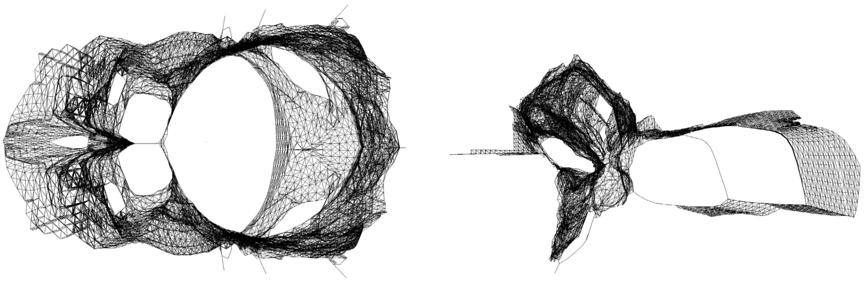


**Fig. 6.** Drawings of the graphs BCSSTK31 ($|V| = 35588, |E| = 572914$) and BCSSTK32 ($|V| = 44609, |E| = 985046$) with 200 pivots. In the experimental study of [14] these graphs posed serious difficulties for most methods.

## 5 Conclusion

We have proposed a simple and efficient method for drawing very large undirected graphs based on MDS. With pivot approximation it can be implemented to run in linear time and with linear memory.

The graph layout can be made progressive by extending the set of pivots incorporated in the layout computation. This allows for quick generation and display of a decent preview layout, which can then be refined by further computation carried out in the background.

In our experiments, we found that generally a very small number of pivots is sufficient and that running time for computing the eigenvectors was negligible with respect to setting up the distance-submatrix $C$. The essential difference to LMDS is that $C^T C$ contains more relations than just those between landmarks. LMDS and PMDS are therefore equally efficient in practice. We also noted, however, that PMDS indeed requires fewer pivots in general to reach the same quality level, while offering greater overall stability.

Even though our prototypical implementation is written in Java, and we did not perform any optimization, the running times compare favorably with the fastest methods available, and are likely to be reduced significantly in a dedicated implementation.

## References

1. W. Basalaj. Incremental multidimensional scaling method for database visualization. In *Proc. VDA*, pages 149–158, 1999.
2. Y. Bengio, J.-F. Paiement, P. Vincent, O. Delalleau, N. Le Roux, and M. Ouimet. Out-of-sample extensions for LLE, Isomap, MDS, eigenmaps, and spectral clustering. In *NIPS*, pages 307–311, 2004.
3. I. Borg and P. Groenen. *Modern Multidimensional Scaling.* Springer, 2005.
4. A. Buja and D. F. Swayne. Visualization methodology for multidimensional scaling. *J. Classification*, 19:7–43, 2002.
5. C. J. C. Burges. Geometric methods for feature extraction and dimensional reduction. Technical report, Microsoft Research, 2004.
6. M. Chalmers. A linear iteration time layout algorithm for visualizing high-dimensional data. In *Proc. InfoVis*, pages 127–132. IEEE, 1996.
7. A. Civril, M. Magdon-Ismail, and E. Bocek-Rivele. SDE: Graph drawing using spectral distance embedding. In *Proc. Graph Drawing*, pages 512–513, 2005.
8. J. D. Cohen. Drawing graphs to convey proximity. *ACM Transactions on Computer-Human Interaction*, 4(3):197–229, 1997.
9. T. Cox and M. Cox. *Multidimensional Scaling.* CRC/Chapman and Hall, 2001.
10. V. de Silva and J. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Proc. NIPS*, pages 721–728, 2003.
11. C. Faloutsos and K. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc. ACM SIGMOD*, pages 163–174, 1995.
12. E.R. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In *Proc. Graph Drawing*, pages 239–250, 2004.

13. G. H. Golub and C. F. van Loan. *Matrix computations.* Johns Hopkins University Press, 1996.
14. S. Hachul and M. Jünger. An experimental comparison of fast algorithms for drawing general large graphs. In *Proc. Graph Drawing*, pages 235–250, 2005.
15. D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *Proc. Graph Drawing*, pages 388–393, 2002.
16. F. Jourdan and G. Melançon. Multiscale hybrid MDS. In *Proc. IV*, pages 388–393. IEEE, 2004.
17. Charles Kadushin. Personal communication.
18. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
19. Y. Koren. Graph drawing by subspace optimization. In *Proc. VisSym*, pages 65–74, 2004.
20. Y. Koren, L. Carmel, and D. Harel. ACE: A fast multiscale eigenvectors computation for drawing huge graphs. In *Proc. InfoVis*, pages 137–144. IEEE, 2002.
21. Y. Koren and D. Harel. One-dimensional layout optimization, with applications to graph drawing by axis separation. *Computational Geometry: Theory and Applications*, 32:115–138, 2005.
22. J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.
23. J. B. Kruskal and R. E. Hart. A geometric interpretation of diagnostic data from a digital machine: Based on a study of the Morris, Illinois, Electronic Central Office. *Bell Sys. Tech. J.*, 45(8):1299–1338, 1966.
24. J. B. Kruskal and D. Seery. Designing network diagrams. In *Proc. First General Conference on Social Graphics*, pages 22–50, 1980.
25. A. Morrison and M. Chalmers. Improving hybrid MDS with pivot-based searching. In *Proc. InfoVis*, pages 85–90. IEEE, 2003.
26. A. Morrison, G. Ross, and M. Chalmers. A hybrid layout algorithm for subquadratic multidimensional scaling. In *Proc. InfoVis*, pages 152–158. IEEE, 2002.
27. J. C. Platt. FastMap, MetricMap, and Landmark MDS are all Nyström Algorithms. Technical report, Microsoft Research, 2004.
28. L. K. Saul, K. Q. Weinberger, J. H. Ham, F. Sha, and D. D. Lee. Spectral methods for dimensionality reduction. In B. Schölkopf, O. Chapelle, and A. Zien, editors, *Semi-Supervised Learning.* MIT Press, 2006. To appear.
29. J. G. Silva, J. S. Marques, and J. M. Lemos. Selecting landmark points for sparse manifold learning. In *Proc. NIPS*, 2005.
30. W. S. Torgerson. Multidimensional scaling: I. Theory and Method. *Psychometrika*, 17:401–419, 1952.
31. J. T.-L. Wang, X. Wang, K. Lin, D. Shasha, B. A. Shapiro, and K. Zhang. Evaluating a class of distance-mapping algorithms for data mining and clustering. In *Proc. KDD*, pages 307–311, 1999.
32. M. Williams and T. Munzner. Steerable, progressive multidimensional scaling. In *Proc. InfoVis*, pages 57–64. IEEE, 2004.

# Angle and Distance Constraints on Tree Drawings

Ulrik Brandes and Barbara Schlieper[⋆]

Department of Computer & Information Science, University of Konstanz

**Abstract.** We consider planar drawings of trees that must satisfy constraints on the angles between edges incident to a common vertex and on the distances between adjacent vertices. These requirements arise naturally in many applications such as drawing phylogenetic trees or route maps. For straight-line drawings, either class of constraints is always realizable, whereas their combination is not in general. We show that straight-line realizability can be tested in linear time, and give an algorithm that produces drawing satisfying both groups of constraints together in a model where edges are represented as polylines with at most two bends per edge or as continuously differentiable curves.

## 1 Angle and Distance Constraints

We are interested in planar drawings of simple undirected graphs $G = (V, E)$. Throughout this paper, we assume that $G$ is planar and let $n = |V|$ denote the number of vertices and $m = |E|$ the number of edges. We are particularly interested in drawing *trees* $T = (V, E)$, i. e. graphs that are connected and acyclic. Denote by $T(e, v)$ the tree obtained from splitting $T$ by removing $e \in E$ and choosing the component that contains $v \in V$. For any $r \in V$ let $T_r$ be the tree *rooted* at $r$, and $T_r(v)$ the subtree of all descendants of $v$ (including $v$ itself). Clearly, $T_r(r) = T$ and $T_r(v) = T(v, e)$ if $v \neq r$ and $e$ is the unique first edge on the path from $v$ to $r$. With $\{v, w\}$ we refer to the undirected edge incident to $v$ and $w$.

The implications of the following two types of constraints on drawings of a graph $G$ are investigated.

**Distance constraints:** A drawing of a graph $G = (V, E)$ satisfies *distance constraints* $\delta : E \rightarrow \mathbb{R}^+$, if all pairs of adjacent vertices $\{v, w\} \in E$ are exactly at distance $\delta(v, w)$.

**Angle constraints:** A graph is said to be *embedded* (combinatorially), if the, say, counterclockwise cyclic ordering of edges incident to the same vertex is prescribed. For an embedded graph $G = (V, E)$, let $A \subseteq E \times E$ be the *angle set*, where $(e_1, e_2) \in A$ iff both edges share a vertex $v$ and $e_2$ is the counterclockwise next edge after $e_1$ around $v$.

[⋆] To whom correspondence should be directed: `schliepe@inf.uni-konstanz.de`. This author gratefully acknowledges financial support from the state of Baden-Württemberg (Landesgraduiertenförderungsgesetz scholarship).

A drawing of an embedded graph $G = (V, E)$ satisfies *angle constraints* $\alpha : A \to (0, 2\pi]$, if the angle between all pairs $(e_1, e_2) \in A$ is exactly $\alpha(e_1, e_2)$. Note that $\alpha$ is frequently called an *angle assignment*.
A necessary requirement for angle constraints to be satisfiable is that they sum to $2\pi$ around every vertex, and to $(d_G(f) - 2)\pi$ around every inner face $f$ with $d_G(f)$ vertices. Such a set of angle constraints is called *locally consistent*, and we assume that all given angle constraints are.

A graph with angle and/or distance constraints is called *realizable in the straight-line model* (or *straight-line realizable* for short), if there exists a planar straight-line drawing in the plane, such that all constraints are satisfied.

## 2 Straight-Line Realizability

Testing straight-line realizability is known to be $\mathcal{NP}$-complete for both distance-constrained graphs (even if all edges are constrained to have unit length) [5] and angle-constrained graphs [7]. For trees, arbitrary distance and angle constraints can be satisfied, though not necessarily in the same drawing.

**Theorem 1.** *For any tree $T = (V, E)$ with locally consistent angle constraints $\alpha$ (or distance constraints $\delta$), a planar straight-line drawing satisfying $\alpha$ (or $\delta$) can be determined in linear time.*

*Proof.* A drawing of $T$ satisfying any locally consistent angle constraints, can be determined in linear time using a simple postorder traversal to create a balloon layout [9]: starting with an empty circle of arbitrary radius around each leaf, parent edges are stretched such that the enclosing circles of subtrees rooted at siblings do not intersect when satisfying the angle constraints.

An algorithm for drawing any distance-constrained tree in linear time is given in [1]. $\square$

Note that straight-line drawings of trees with both angle and distance constraints are completely determined (up to translation and rotation).

**Theorem 2.** *Straight-line realizability of trees with both angle and distance constraints can be tested in linear time.*

*Proof.* We show that straight-line realizability testing is equivalent to testing simplicity of polygonal chains, which can be done in linear time [3].

Since a polygonal chain can be viewed as a tree with angle and distance constraints, trees cannot be tested faster than polygonal chains. On the other hand, an embedded tree with $l$ leaves can be covered by $l$ paths $p_1, \ldots, p_l$ connecting each leaf with the first leaf encountered in a right-first search. Due to the given constraints, each path corresponds to a polygonal chain, and the tree is realizable if and only if all $p_i$, $1 \leq i \leq l$ are simple. Since the union of these paths corresponds to an Euler tour around the tree, the total size of the polygonal chains is $2m = 2n - 2$. $\square$

## 3 Polyline Representation

If an angle and distance constrained tree is not straight-line realizable, it can still be drawn without edge intersections by allowing polylines. In the remainder of the section, we will prove the following theorem.

**Theorem 3.** *For a tree, a planar polyline drawing that satisfies locally consistent angle and distance constraints and has at most two bends per edges can be determined in linear time.*

In the first step we calculate an initial layout of $T_r$ for an arbitrary root $r \in V$ with the length-preserving algorithm of [1] and given edge lengths $\delta$. We exploit an invariant characteristic of the layouts computed.

**Theorem 4.** *For a tree with given vertex positions a planar polyline drawing that satisfies locally consistent angle constraints and has at most two bends per edge can be determined in linear time, if the given vertex positions are such that disjoint subtrees are contained in disjoint wedges.*

This is a special case of the problem to find a drawing of a planar graph with fixed vertices and pre-specified angles between the edges incident to the same vertex [2]. A related problem is embedding a planar graph on a fixed set of points in the plane. The graph can be drawn without edge intersections using at most two bends per edge in polynomial time, if the mapping between the vertices $V$ and the points $P$ is not fixed [8]. However, if the mapping is fixed i. e., each vertex has a fixed position such that the straight-line drawing is not necessarily planar, up to $O(n)$ bends per edge can be needed to guarantee planarity and this bound is known to be asymptotically optimal in the worst case [10]. Note that these strategies do not yield drawings that satisfy angle constraints. Angle constraints have to be satisfied for example when drawing graphs with good angular resolution. A planar graph can be embedded and drawn planar with at most one bend per edge, such that for each $(e_1, e_2) \in A$ sharing a vertex $v \in V$ it is $\alpha(e_1, e_2) \geq \frac{1}{d(v)}$ where $d(v)$ denotes the degree of $v$ [4]. For not necessarily planar graphs angular resolution and the number of edge crossings can be improved modifying a force-directed graph drawing algorithm into an algorithm for drawing graphs with curved edges [6]. Note that for these drawings no distances constraints are to be satisfied.

In our drawings edges will be represented as polylines. The polyline of an edge $\{v, w\}$ will be determined by the endpoints of two control segments incident to $v$ and $w$.

We guarantee the planarity in two steps: For a vertex $v \in V$

- we determine the direction of each initial control segment.
- we determine the length for each initial control segment.

In Sect. 3.1 we define a rotation angle $\beta$ to guarantee certain situations regarding the angles of the initial control segments incident to a vertex $v$ and those of the straight lines from $v$ to the corresponding neighbors. In Sect. 3.2

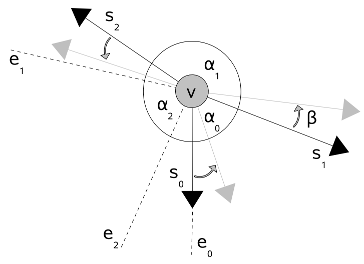


**Fig. 1.** Rotating the angle template

we determine the control segment lengths to avoid intersections in the remaining situations.

We will focus on each vertex a constant number of times and look at all incident edges so the overall running time is linear.

Let $v$ be a vertex we focus on with $k$ incident control segments $s_0 \dots s_{k-1}$ (in counterclockwise order) belonging to the $k$ polylines of edges $e_0 \dots e_{k-1}$ to the $k$ neighbors $w_0 \dots w_{k-1}$ of $v$. We refer to the absolute angle of a control segment $s_i$ with $\gamma_i = \gamma_0 + \sum_{t=0}^{k-1} \alpha_t$ and $l_i$ its length. Further, let $s'_i$ be the control segment for $e_i$ incident to $w_i$ with angle $\gamma'_i$ and length $l'_i$. Let $p_i$ denote the target point of $s_i$ and $p'_i$ the target point of $s'_i$, $\lambda_i$ the angle of the line from $v$ to $w_i$ and $\lambda'_i$ the angle of the same line, but from $w_i$ to $v$.

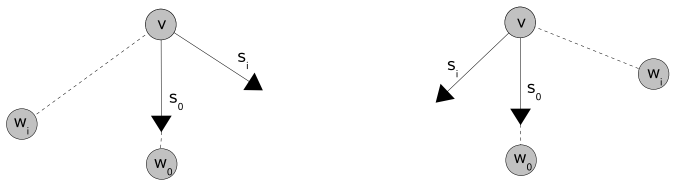
## 3.1 Control Segment Angles

Since only the relative angles between consecutive control segments incident to $v$ are given, to determine the final layout we have to choose the absolute angle for one of the control segments. We start with $\gamma_0 = \lambda_0$ and then rotate the whole angle template by an angle $\beta$ i.e., update each angle $\gamma_i = \gamma_i + \beta$. For a neighbor $w_i$ of $v$ the *angular deviation* is $\theta(i) = (\gamma_i - \lambda_i) \bmod 2\pi$. We say that $w_j$ *lies between* $s_i$ and $w_i$, if $(\gamma_i - \lambda_j) \bmod 2\pi < (\gamma_i - \lambda_i) \bmod 2\pi$.

We assume that for each neighbor $w_i$ the endpoint $p'_i$ of the control segment incident to $w_i$ lies on the same side of the line $(v, p_i)$ and will assure this when determining the control segment lengths (see (7),(8)). The following two situations will cause intersections in the starting configuration, because $e_i$ will be intersecting with $e_0$ (see Fig. 2 for illustration). For a pair $s_i, w_i$ we say that

$$\begin{array}{c} i \text{ and } 0 \text{ are } \textit{clockwise crossing} \text{ if} \\ \theta(i) \leq \pi \text{ and } w_0, s_0 \text{ lie between } s_i \text{ and } w_i, \end{array} \tag{1}$$

$$\begin{array}{c} i \text{ and } 0 \text{ are } \textit{counterclockwise crossing} \text{ if} \\ \theta(i) \geq \pi \text{ and } w_0, s_0 \text{ lie between } w_i \text{ and } s_i. \end{array} \tag{2}$$



(a) clockwise crossing (b) counterclockwise crossing

**Fig. 2.** Two cases in the start situation

**Lemma 1.** *At any vertex $v \in V$, there are either clockwise or counterclockwise crossings, if any.*

*Proof.* Assume $i_1$ fulfills Equation (1) and $i_2$ Equation (2), then $s_0$ lies between $s_{i_1}$ and $s_{i_2}$ and $w_0$ lies between $w_{i_2}$ and $w_{i_1}$. The orders of the control segments and of the neighbors are fixed, hence $i_1 < i_2$ and $i_1 > i_2$, which is a contradiction. □

If a vertex $v$ has a neighbor $w_i$ such that $i$ and 0 are clockwise crossing we rotate $v$'s angle template counterclockwise. If $i$ and 0 are counterclockwise crossing for an $0 \leq i \leq k-1$ this is the mirrored case and can be solved by rotating clockwise. After rotating the angle template we will have (see Fig. 3):

**Lemma 2 (counterclockwise sheering).** *For $0 \leq i, j \leq k-1$, $w_j$ lies between $w_i$ and $s_i$ with angular deviation $\theta(i) > \pi$, if and only if also $s_i$ lies between $w_j$ and $s_j$ with $\theta(j) > \pi$.*

**Lemma 3 (clockwise sheering).** *For $0 \leq i, j \leq k-1$, $w_j$ lies between $s_i$ and $w_i$ with angular deviation $\theta(i) < \pi$, if and only if also $s_i$ lies between $s_j$ and $w_j$ with $\theta(j) < \pi$.*

In these two situations we can avoid intersections by determining the control segment lengths.

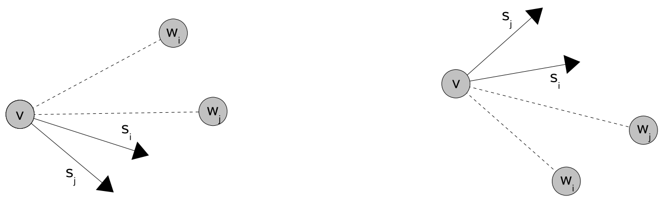
We say $s_j$ is *pulled between* $s_i$ and $w_i$, if it was $(\lambda_i - \gamma_j) \bmod 2\pi < \pi$ before rotation, but is $(\lambda_i - \gamma_j) \bmod 2\pi > \pi$ and $s_j$ lies between $s_i$ and $w_i$ after rotation. We say $s_i$ is *pushed over* $w_j$, if it was $(\lambda_j - \gamma_i) \bmod 2\pi < \pi$ before rotation, but is $(\lambda_j - \gamma_i) \bmod 2\pi > \pi$ and $w_j$ lies between $s_i$ and $w_i$ after rotation.

Of all pairs $s_i, w_i$ for which $i$ and 0 are clockwise crossing let $i_{\min}$ be the one for which the angular deviation $\theta(i)$ is minimal and of all pairs $s_i, w_i$ for which $i$ and 0 are not clockwise crossing but $\theta(i) < \pi$ let $i_{\max}$ be the one for which $\theta(i)$ is maximal.

We rotate the angle template counterclockwise by an angle $\beta$ with

$$0 \leq \pi - \theta(i_{\min}) < \beta < \pi - \theta(i_{\max}) \leq \pi \qquad (3)$$

**Lemma 4.** *For every vertex $v$ there is a feasible rotation $\beta$, i. e. $\theta(i_{\min}) > \theta(i_{\max})$.*



(a) counterclockwise sheering (b) clockwise sheering

**Fig. 3.** Two solvable situations

*Proof.* For $i_{\max} > i_{\min}$ the vertex $w_{i_{\max}}$ lies between $w_0$ and $w_{i_{\min}}$ because the order of neighbors is fixed. $i_{\max}$ and 0 are not clockwise crossing, hence $s_{i_{\max}}$ lies between $w_0$ and $w_{i_{\max}}$. So it must be $\theta(i_{\min}) > \theta(i_{\max})$ because both $s_{i_{\max}}$ and $w_{i_{\max}}$ lie between $s_{i_{\min}}$ and $w_{i_{\min}}$ with $\theta(i_{\max}) < \pi$. The proof for the case $i_{\max} > i_{\min}$ is analogous. □

**Lemma 5.** *These bounds are sharp.*

*Proof.* For $\beta \leq \pi - \theta(i_{\min})$ it would still be $\theta(i_{\min}) \leq \pi$ after rotation and both $w_0$ and $s_0$ would still lie between $s_{i_{\min}}$ and $w_{i_{\min}}$, hence $e_{i_{\min}}$ would be intersecting with $e_0$.

For $\beta \geq \pi - \theta(i_{\max})$ it would be $\theta(i_{\max}) \geq \pi$ after rotation and both $w_{i_{\min}}$ and $s_{i_{\min}}$ would lie between $w_{i_{\max}}$ and $s_{i_{\max}}$, hence $e_{i_{\min}}$ would be intersecting with $e_{i_{\max}}$. □

Within these bounds we can now optimize any objective function, for example the sum over all squared angular deviations $\theta(i)^2$ for $0 \leq i \leq k-1$. See [2] for other reasonable objective functions.

To proof Lemma 2 and Lemma 3 we first proof the following:

**Lemma 6.** *If $\theta(i) > \pi$ after rotation for $0 \leq i \leq k-1$, before and after rotation neither $w_0$ nor $s_0$ can lie between $w_i$ and $s_i$.*

*Proof.* If $\theta(i) > \pi$ before rotation, $w_0$ and $s_0$ cannot have lain between $w_i$ and $s_i$, because $i$ and 0 would have been counterclockwise crossing. We rotate counterclockwise by an angle $\beta < \pi$, hence they cannot after rotation as well.

If $\theta(i) \leq \pi$ before rotation but $\theta(i) > \pi$ after, $i$ and 0 must have been clockwise crossing before rotation, so $w_0$ and $s_0$ cannot lie between $w_i$ and $s_i$ after rotation. □

**Lemma 7.** *If $\theta(i) < \pi$ after rotation for $0 \leq i \leq k-1$, not both $w_0$ and $s_0$ can lie between $s_i$ and $w_i$.*

*Proof.* Before rotation $i$ and 0 cannot have been clockwise crossing, so by rotating counterclockwise by an angle $\beta < \pi$ from the start situation $\lambda_0 = \gamma_0$, it can only happen that either $s_i$ is pushed over $w_0$ or $s_0$ is pulled between $s_i$ and $w_i$. □

**Corollary 1.** *For the angular deviation of a vertex $w_i$, $\theta(i) = \pi$ is impossible for any $0 \leq i \leq k-1$.*

*Proof.* of Lemma 2 (Lemma 3 can be proven with the same ideas)

"$\Rightarrow$" Neither $s_0$ nor $w_0$ can lie between $w_i$ and $s_i$ (Lemma 6) so $w_0$ cannot lie between $w_i$ and $w_j$ and we have $j < i$. Because the order of control segments is fixed $s_0$ cannot lie between $s_i$ and $s_j$, hence $s_0$ must lie between $s_j$ and $w_i$ and $s_0$ lies between $s_j$ and $w_j$. If it was $\theta(j) \leq \pi$ $s_0$ must have been pulled between $s_j$ and $w_j$ (Lemma 7) so $s_0$ must have lain between $w_i$ and $s_i$ which is a contradiction to Lemma 6.

"$\Leftarrow$" Neither $s_0$ nor $w_0$ can lie between $w_j$ and $s_j$ (Lemma 6) so $s_0$ cannot lie between $s_i$ and $s_j$ and we have $j < i$. Because the order of neighbors is fixed $w_0$ cannot lie between $w_i$ and $w_j$, hence $w_0$ must lie between $s_j$ and $w_i$ and $w_0$ lies between $s_i$ and $w_i$. If it was $\theta(i) \leq \pi$ $s_i$ must have pushed over $w_0$ (Lemma 7), so also $s_j$ must have pushed over $w_0$, hence $w_0$ must have lain between $w_j$ and $s_j$ which is a contradiction to Lemma 6.

$\square$

## 3.2 Control Segment Lengths

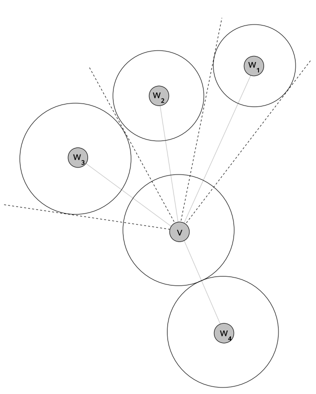
In the remaining situations we can avoid intersections by determining first the radius $r_v$ of the circle containing all control segments incident to a vertex $v$ and then the lengths $l$ and $l'$ of an edge $e$'s control segments. We have to make sure that none of the three segments of an edge $e$'s polyline can intersect with any segment of another polyline. The maximal possible radius $r_v$ is determined such that:

- control segments incident to different neighbors of $v$ can not intersect
- the control segments incident to $v$ can not intersect with a control segment incident to any of $v$'s neighbors

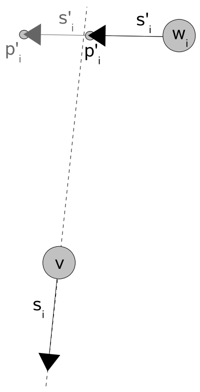
For an edge $e = \{v, w\}$ the maximal possible length of $e$'s control segment $s$ incident to $v$ is determined such that:

- neither the middle segment of $e$ nor the control segment $s$ can intersect with a control segment incident to another neighbor
- neither the middle segment of $e$ nor the control segment $s$ can intersect with the middle segment of another edge

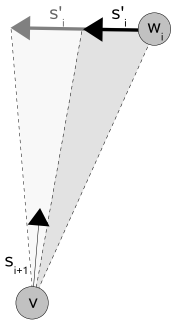
When we computed the initial layout of $T_r$ a wedge with size $\omega_v$ was assigned to each vertex $v$ in which $v$'s subtree $T(v)$ was lying and that was divided among the children. The children's wedges are rooted in $v$. We now look at the unrooted tree $T$ and all the wedges of $v$'s neighbors are rooted in $v$. The vertex $w_0$, that had been $v$'s parent in $T_r$, is now lying in an opposed wedge with size $\omega_{w_0} = 2\pi - \omega_v$



(a) wedges for vertex $v$



(b) line $(v, p_i)$



(c) range of $w_i$

**Fig. 4.**

(see Fig. 4(a)). To guarantee that the control segments incident to one neighbor are not intersecting with the control segments of another neighbor, we determine for $0 \leq i \leq k-1$:

$$r_{w_i} < \begin{cases} \sin \frac{\omega_{w_i}}{2} \cdot \parallel w_i - v \parallel_2 & \text{if } \omega_{w_i} < \pi \\ \sin(\pi - \frac{\omega_{w_i}}{2}) \cdot \parallel w_i - v \parallel_2 & \text{otherwise} \end{cases} \tag{4}$$

We also have to guarantee that the control segments incident to $v$ are not intersecting with the control segments incident to one of $v$'s neighbors:

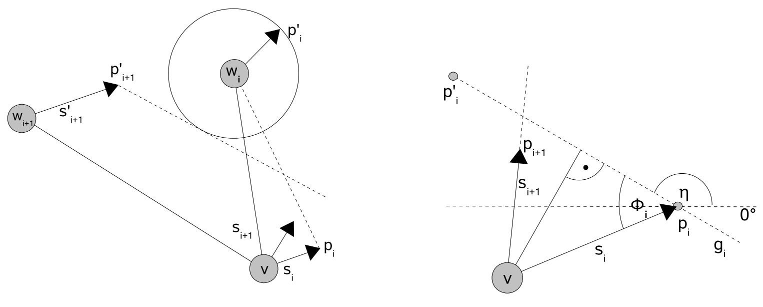
$$r_v \leq \frac{1}{2} \min\{\delta\{e_i\}\}_{0 \leq i \leq k-1} \tag{5}$$

The computations in (4) and (5) will be done first for each vertex $v \in V$ to determine $r_v$, which we will need in the following.

A neighbor $w_i$ cannot be involved both in a clockwise and a counterclockwise sheering. Let $i$ and $(i+1) \bmod k$ (we will write $i+1$ in the remainder) be counterclockwise sheering. We have to further determine the length $l_{i+1}$ of the control segment $s_{i+1}$.

The polyline of an edge $e_{i+1}$ might intersect with $e_i$ or another edge incident to $w_i$ if a line through $p'_{i+1}$ and $p_{i+1}$ would be intersecting with the circle containing all control segments incident to $w_i$. We can avoid this by choosing $l_{i+1}$ such that $s_{i+1}$ is not intersecting with a tangent line to $w_i$'s circle through $p'_{i+1}$. Further, $s_{i+1}$ must not intersect with the line $g_i = (p_i, p'_i)$. We use the maximal length $r_{w_i}$ here to determine possible coordinates of $p'_i$. If we focus on $w_i$ later, $s'_i$ might have to be shortened, hence $s_{i+1}$ must not intersect with the line $(p_i, w_i)$ and the tangent line to $w_i$'s circle through $w_{i+1}$. See Figure 5(a) for illustration.

The computations for a length $l_{i+1}$ such that $s_{i+1}$ is not intersecting with one of these lines, are all very similar. We show the computation here for the line



(a) Lines (b) length of $s_{i+1}$

**Fig. 5.** Bounding control segments

$g_i = (p_i, p'_i)$ (note that the length $l_i$ has to be fixed already) with absolute angle $\eta_i$ and $\phi_i$ the angle between $g_i$ and $s_i$ in (6) (see Figure 5(b) for illustration):

$$l_{i+1} < \frac{\sin\phi_i \cdot l_i}{\sin(\pi - \phi_i - \gamma_{i+1} + \gamma_i)} \tag{6}$$

In Sect. 3.1 we assumed that the neighbor $w_i$ and the endpoint $p'_i$ of the control segment incident to $w_i$ lie on the same side of the line $(v, p_i)$. We assure this by choosing $l'_i$, the length of $s'_i$, such that $s'_i$ does not intersect with the line $(v, p_i)$ (see Fig. 4(b)):

If $(\gamma'_i - \lambda'_i) \bmod 2\pi > \pi$ and $\theta(i) > \pi$:

$$l'_i < \frac{\sin(\gamma_i + \pi - \lambda_i) \cdot \delta(e_i)}{\sin(-\gamma_i + \lambda_i - \lambda'_i + \gamma'_i)} \tag{7}$$

If $(\gamma'_i - \lambda'_i) \bmod 2\pi < \pi$ and $\theta < \pi$:

$$l'_i < \frac{\sin(\lambda_i - \gamma_i - \pi) \cdot \delta(e_i)}{\sin(-\lambda_i + \gamma_i - \gamma'_i + \lambda'_i)} \tag{8}$$

Let $w_i$ be one neighbor of $v$. We call the (smallest) sector of $w_i$'s wedge, in which the control segment $s'_i$ incident to $w_i$ is lying, the *range* of $w_i$. If another control segment $s_j$ incident to $v$ is lying within this range, the polylines of $e_i$ and $e_j$ can be intersecting without $i$ and $j$ clockwise or counterclockwise sheering. We avoid this by choosing the length $l'_i$ of $s'_i$ such that $s'_i$ is not intersecting with the line $(v, p_j)$ (see Fig. 4(c)), the computation is analog to (7) and (8).

With these control segments lengths we can draw the edges' polylines without intersections (see also Lemma 9). In Fig. 7(a) a tree after determining the vertex positions in displayed with an arbitrary rotation angle for each vertex $v$ and control segment lengths smaller than $r_v$. In Fig. 7(b) the rotation angle is determined like shown in Sect. 3.1 and the control segment lengths like in Sect. 3.2.

## 4 Curve Representation

Instead of using polylines we can also represent the edges as smooth curves. A cubic Bézier curve is determined by two endpoints $b_0, b_3$ and two inner control points $b_1, b_2$. We call the segments $\overline{b_0b_1}$ and $\overline{b_3b_2}$ the (initial) control segments, while $\overline{b_1b_2}$ is called inner segment. A Bézier curve is contained in the convex hull of its defining points, and the tangents at its endpoints are collinear with the initial control segments, so the outgoing angle of a Bézier curve is the angle of its control segment. For the Bézier curve of an edge we use the control segments of the corresponding polyline as initial control segments. We refer to the Bézier curve from $v$ to $w_i$ for $0 \leq i \leq k-1$ with $c_i$ and $H_i$ the convex hull of its control points. In the remainder of this section we will proof the following theorem.

**Theorem 5.** *For a tree, a planar drawing that satisfies locally consistent angle and distance constraints while representing edges as continuously differentiable curves consisting of at most two cubic Bézier curves and a straight line segment can be determined in linear time.*

Even if the polylines of two edges are not intersecting, the hulls of the corresponding Bézier curves can intersect. Three borders of a curve's hull are the line segments of the corresponding polyline, and when determining the control segment lengths we avoided most intersections, but in case of a clockwise or counterclockwise sheering we have to split an edge's curve by adding control segments.

Let $i$ and $i+1$ be counterclockwise sheering. When all the control segment lengths in the tree are determined, we split a curve $c_i$. Let $q_i$ be the intersection point of the lines $(v, p_{i+1})$ and $g_i$. The splitting point $m_i$ must lie on $g_i$ between the point $q_i$ and $p_i$. Rooted at $m_i$ we add two diametral opposed control segments $s_i^{m1}$ and $s_i^{m2}$ on $g_i$, with $s_i^{m1}$ pointing to $p_i$ and $s_i^{m2}$ pointing to $p'_i$. (See Fig. 6 for illustration.) Now we can describe $c_i$ by two Bézier curves smoothly attached (continuously differentiable, because the angle at $m_i$ has size $\pi$) one by $s_i$ and $s_i^{m1}$ and one by $s'_i$ and $s_i^{m2}$. When we later focus on $w_i$, we might have to split $c_i$ to avoid intersections with curves incident to $w_i$. If then the splitting point cannot lie between $q_i$ and $p_i$, we will add a second splitting point on $g_i$, otherwise one splitting point will be sufficient.

This procedure induces that $l_i$ has to be calculated first. Therefore we create a vertex list $L^+$ sorted by increasing index and $\theta(i) < \pi$ for each $w_i \in L^+$.

If for $v$ there is a pair $i$ and $(i-1) \bmod k$ clockwise sheering this is symmetrically the same situation and will be solved with the same strategy. We will need a vertex list $L^-$ sorted by decreasing index and $\theta(i) > \pi$ for each $w_i \in L^-$. We can create both types of vertex lists testing all neighbors $w_i$ for $0 \leq i \leq k-1$ in counterclockwise order.

**Lemma 8.** *Neither the vertex list $L^+$ nor $L^-$ will contain all neighbors of $v$.*

*Proof.* We start with the angular deviation $\theta(0) = 0$. If we do not have to rotate $v$'s angle template the neighbor $w_0$ will be in none of the vertex lists. Iff

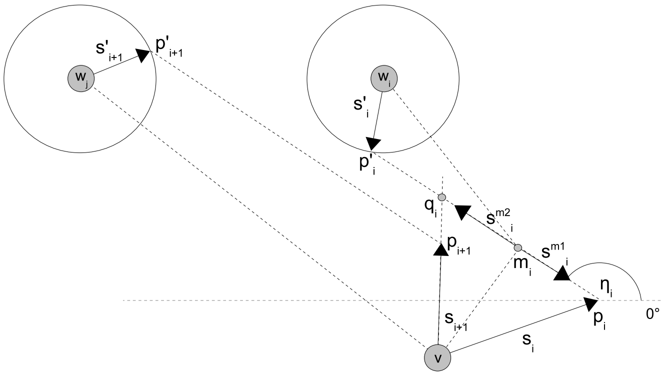


**Fig. 6.** Splitting a curve

there is a neighbor $w_i$ of $v$ such that $i$ and 0 are clockwise crossing we rotate counterclockwise by an angle $\beta < \pi$ until $\theta(i) > \pi$. After rotation it will be $\theta(0) < \pi$, hence $w_0$ and $w_i$ will be in different vertex lists. Rotating clockwise is the mirrored case. □

**Lemma 9.** *The resulting tree layout is planar.*

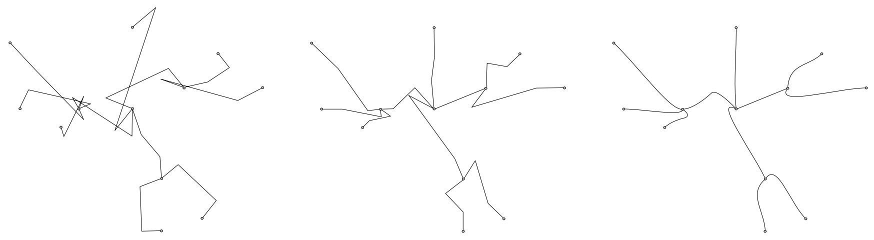
*Proof.* For any vertex $v$ the curve or polyline of an edge $e_{i+1}$ incident to $v$ can cause problems, if the hull $H_{i+1}$ is intersecting with the wedge of another neighbor of $v$. With the strategy presented previously we made sure, that $H_i$ and $H_{i+1}$ are not intersecting.

By determining the length of $s_{i+1}$ in (6) we also made it impossible for $H_{i+1}$ to intersect with any wedge or control segment of one of $w_i$'s neighbors different from $v$. With this, $H_{i+1}$ can not intersect with any line from a point in $w_i$'s circle to a point in the wedge of one of $w_i$'s neighbors different from $v$, thus $H_j$ cannot intersect with any hull of a curve incident to $w_i$ or any vertex in the subtree $T(w_i, \{w_i, v\})$.

Hull $H_{i+1}$ intersecting with the wedge of the neighbor $w_{i+2}$ is the mirrored case. □

## 5 Discussion

We presented efficient algorithms for drawing trees with constraints on distances between adjacent vertices and angles between incident edges. There is plenty of opportunity for further work. For instance, we would like to address angle and distance constraints together to improve both vertex placements and angle rotations, and enlarge the class of graphs on which our methods work. A major challenge is to implement the rotation method of [2] so that planarity is maintained always.



(a) random rotation (b) polylines (c) curves

**Fig. 7.** Drawings of a tree with fixed vertices and angles

## References

[1] C. Bachmaier, U. Brandes, and B. Schlieper. Drawing phylogenetic trees. In *Proc. 16th Intl. Symp. Algorithms and Computation (ISAAC '05)*, volume 3827 of *LNCS*, pages 1110–1121. Springer, 2005.

[2] U. Brandes, G. Shubina, and R. Tamassia. Improving angular resolution in visualizations of geographic networks. In *Data Visualization: Proc. 2nd Joint EUROGRAPHICS and IEEE TCVG Symp. Visualization, VisSym*, pages 23–33. Springer, 29–30 2000.

[3] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[4] C. C. Cheng, C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Drawing planar graphs with circular arcs. In *Proc. Graph Drawing 1999*, volume 1731 of *LNCS*, pages 117–126. Springer, 1999.

[5] P. Eades and N. C. Wormald. Fixed edge-length graph drawing is $\mathcal{NP}$-hard. *Discrete Applied Mathematics*, 28:111–134, 1990.

[6] B. Finkel and R. Tamassia. Curvilinear Graph Drawing Using the Force-Directed Method. In *Proc. Graph Drawing 2004*, volume 3383 of *LNCS*, pages 448–453. Springer, 2004.

[7] A. Garg. On drawing angle graphs. In *Proc. Graph Drawing 1994*, volume 894 of *LNCS*, pages 84–95. Springer, 1994.

[8] M. Kaufmann and R. Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *Journal of Graph Algorithms and Applications*, 6(1):115–129, 2002.

[9] G. Melançon and I. Hermann. Circular drawing of rooted trees. Technical report 9817, Reports of the Center for Mathematics and Computer Science, 1998.

[10] J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. In *Proc. Graph Drawing 1998*, volume 1547 of *LNCS*, pages 263–274. Springer, 1998.

# Schematisation of Tree Drawings

Joachim Gudmundsson[1], Marc van Kreveld[2], and Damian Merrick[1,3]

[1] National ICT Australia[*], Sydney, Australia
{joachim.gudmundsson,damian.merrick}@nicta.com.au
[2] Department of Computer Science, Utrecht University, The Netherlands
marc@cs.uu.nl
[3] School of Information Technologies, University of Sydney, Australia

**Abstract.** Given a tree $T$ spanning a set of points $\mathcal{S}$ in the plane, we study the problem of drawing $T$ using only line segments aligned with a fixed set of directions $\mathcal{C}$. The vertices in the drawing must lie within a given distance $r$ of each original point $p \in \mathcal{S}$, and an objective function counting the number of bends must be minimised. We propose five versions of this problem using different objective functions, and algorithms to solve them. This work has potential applications in geographic map schematisation and metro map layout.

## 1 Introduction

A schematic drawing is one in which line segments conform to a restricted set of orientations, often for the purpose of increasing readability. Schematic drawings are widely used in many application areas, including electronics, software engineering and visualising geographical networks.

There has been extensive research into orthogonal graph drawing, where edges of a graph are drawn as sequences of alternating horizontal and vertical line segments; see Eiglsperger et al. [5] for a survey. Lauther and Stübinger [8], and subsequently Brandes et al. [2], investigated the problem of generating orthogonal schematic plans from sketch drawings. Hong et al. [7], Stott and Rodgers [13] and Nöllenburg and Wolff [12] looked at the layout of metro maps, in which lines generally conform to horizontal, vertical and diagonal (45° and 135°) orientations.

A popular topic in cartography is polygonal line simplification, and this has also been investigated in a setting of restricted orientations [10,11]. Cabello et al. [3] presented an algorithm to construct various types of schematisations of geographical networks, in which vertex positions remain unmodified but edges are redrawn in a schematised way.

Particularly of interest are certain point placement problems in computational geometry. Cabello and van Kreveld [4] study the problem of aligning as many points as possible in a given set of orientations. If a planar graph is defined on

[*] National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

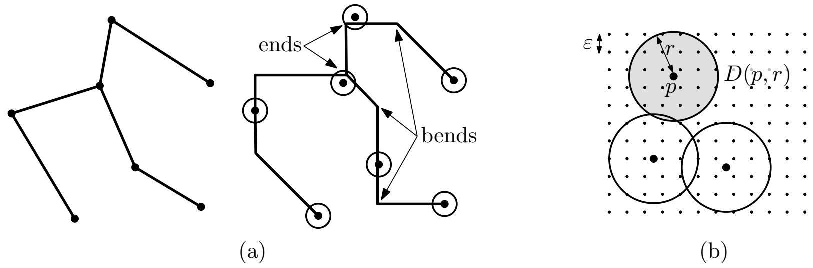


**Fig. 1.** (a) A tree (left) and a schematisation of the tree (right). Some bends (along edges) and ends (at vertices) are marked. (b) The grid $G_\varepsilon$.

the points, and only points connected in the graph are considered, they prove the problem NP-hard, even when only one orientation is used for alignment. They show that discrete forms of the problem can be solved more efficiently, however. This motivates us to discretise our own problems; we detail this later in this section.

In this paper, we aim to produce schematic drawings of trees from initial sketch, geographical or automatic layouts. We propose five problems. Each takes as input a tree $T$ spanning a point set $\mathcal{S}$, and the output should be a *schematisation* of the tree's initial layout (See Fig. 1(a)).

To be considered a schematisation, we require that line segments in the drawing align with certain orientations, and that they pass within a certain distance of each original point $p \in \mathcal{S}$. In addition, we want to minimise an objective function $M(p)$ defined for each point $p \in \mathcal{S}$. $M(p)$ measures the complexity of the lines from $p$ to all of $p$'s children in $T$.

We formally define the concept of a *tree schematisation* as follows.

**Definition 1.** *Given a tree $T$ spanning a point set $\mathcal{S}$, a set of directions $\mathcal{C}$ and a real number $r > 0$, a* tree schematisation *$T'$ of $T$ is a layout of $T$ such that:*

1. *every line segment of $T'$ is aligned with a direction $c \in \mathcal{C}$,*
2. *for every $p \in S$, the corresponding point $p'$ in the schematisation lies inside the disc $D(p, r)$ centred at $p$ with radius $r$*
3. *$\sum_{p \in \mathcal{S}} M(p)$ is minimised.*

The complexity of a tree schematisation can be measured in a number of ways. The primary difference between the five problems we propose lies in the objective function $M(p)$. The choice of $M(p)$ defines how to calculate a cost for each point of a given schematisation. Each problem bases this choice on a particular assumption of how to measure complexity. Some of the problems also take additional input in the form of a set of coloured paths covering $T$, which in turn affects the objective function.

Henceforth, we call the five problems $\pi_1$ - $\pi_5$. The objective functions associated with each of these are denoted $M_1(p)$ - $M_5(p)$. We refer to the value of these functions for a schematisation as the $M_1$ cost, $M_2$ cost, etc.

Each objective function measures the number of *bends* and *ends* in a given schematisation, as illustrated in Fig. 1(a). We use the term *bend* to mean a change in direction of a line between two vertices in the tree $T$. We use *end* to mean a bend at a vertex; that is, a line that does not continue through a vertex in a single direction. What counts as an end depends on the particular problem; we detail this in the following sections.

In this paper, we assume that the set of directions $\mathcal{C}$ is constructed by dividing the angle $2\pi$ by a given even integer $m$ of desired directions, i.e. the $i$th direction in $\mathcal{C}$ is defined by an angle of $\frac{2\pi i}{m}$. $m$ must be even to ensure that for every direction $c \in \mathcal{C}$, the opposite direction $\overline{c}$ is also in $\mathcal{C}$. We look at a discretised version of the general problem of constructing tree schematisations. We place a finite set of points inside each disc $D(p, r)$, and the constructed schematisation must choose one of these points for the position of the vertex $p \in T$. We use a global grid $G_\varepsilon$ with $\varepsilon$ horizontal and vertical spacing between grid points to obtain the set of points $G_\varepsilon(p)$ for each disc $D(p, r)$. $G_\varepsilon(p)$ is the set of grid points in $G_\varepsilon$ that lie inside $D(p, r)$. There are $\mathcal{O}(\frac{r^2}{\varepsilon^2})$ such points (See Fig. 1(b)).

Sections 2 and 3 define $\pi_1$ - $\pi_5$ in detail. Section 4.1 presents a general method for computing tree schematisations on the grid $G_\varepsilon$. This method runs in time exponential in the degree of the tree. In many applications, the degree may be considered constant. In this case, the time complexity of the method is $O(\frac{r^4}{\varepsilon^4}|\mathcal{C}|^{(3+\Delta(T))}n)$, where $\Delta(T)$ is degree of $T$ and $n$ is the number of vertices. However, $\pi_1$ - $\pi_5$ can be solved in time polynomial in the degree of the tree; Sections 4.2 - 4.5 give algorithms for this, based on the general method.

## 2 Coloured Tree Problems

Problems $\pi_1$ and $\pi_2$ involve coloured trees; they take as input a set $\mathcal{P}$ of elementary paths covering the tree $T$, with each path identified by a distinct colour (See Fig. 2).

**Problem $\pi_1$: Single edges only.** In $\pi_1$, we assume that the paths in the set $\mathcal{P}$ are edge-disjoint (as in Fig. 2(a)), and we want to minimise the number of bends and ends along each of the coloured paths independently.

Choose an arbitrary leaf of $T$ to be the root vertex. For any vertex $p$ in $T$, let $N(p)$ be the set of vertices adjacent to $p$ and let $\text{ch}(p) \subseteq N(p)$ be the set of $p$'s children. We define $\pi_1$'s objective function $M_1(p)$ as:

$$M_1(p) = \sum_{q \in N(p)} \text{ends}_1(q, p) + \sum_{q \in \text{ch}(p)} 2 \times \text{bends}(q, p)$$

where $\text{ends}_1(q, p) = 0$ if there is an edge of the same colour as (q, p) entering $p$ from the opposite side, or 1 otherwise, and $\text{bends}(q, p)$ is the number of bends along the edge between $q$ and $p$. Fig. 3 shows examples. The multiplication of bends by 2 equates the cost of a bend in an edge with two ends at a vertex, which may visually appear the same.

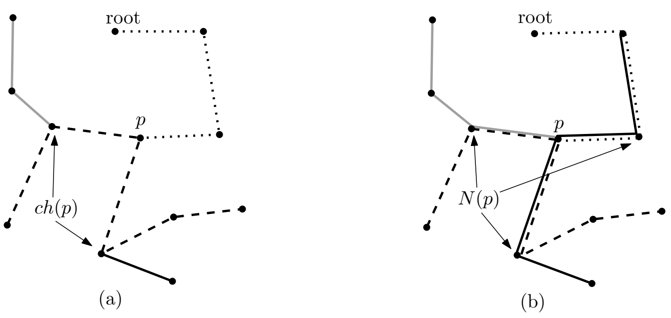


**Fig. 2.** An example of a "coloured tree", with solid, dashed, dotted and light grey lines representing four different coloured paths. The left example shows an edge-disjoint path cover ($\pi_1$), and the right shows a path cover with multiple edges allowed ($\pi_2$). The children ch($p$) and neighbourhood $N(p)$ of the point $p$ are illustrated.

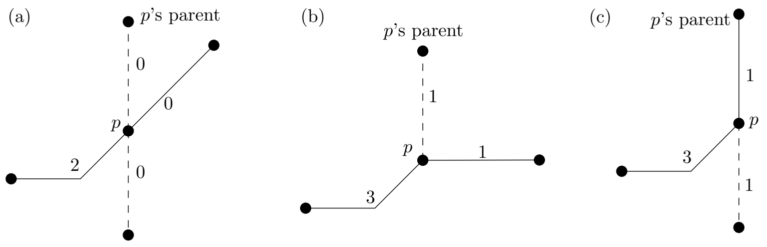


**Fig. 3.** Counting ends and bends in $\pi_1$. A bend costs the same as two ends. The $M_1(p)$ costs are (a) (1 bend) = 2, (b) (1 bend + 3 ends) = 5, and (c) (1 bend + 3 ends) = 5.

**Problem $\pi_2$: Multiple independent edges.** Problem $\pi_2$ removes the requirement for the coloured paths to be edge-disjoint; more than one path may contain an edge between the same two vertices (See Fig. 2(b)). Given multiple edges in the input, it holds that $M_2(p) = M_1(p)$. Every edge between the same pair of vertices counts towards the cost, where in $\pi_1$ there could only be one edge between each adjacent pair of vertices.

$\pi_2$ requires that the endpoints of multiple edges between two nodes must coincide in the schematisation, but not the number of bends in the edge nor the orientations of the line segments.

## 3 Uncoloured Tree Problems

The three remaining problems, $\pi_3$ - $\pi_5$, take uncoloured trees; they assume that no covering set of paths is given. This leads to ambiguity in how to count ends, particularly when many lines in the schematisation enter a vertex in the same direction. $\pi_3$ - $\pi_5$ model three different possibilities.

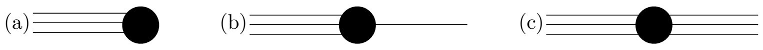


**Fig. 4.** Three lines entering a vertex in the same direction, with (a) no lines, (b) one line and (c) three lines in the opposite direction

**Problem $\pi_3$: Parallel lines remain separate.** $\pi_3$ assumes that multiple edges, or any lines entering a vertex in a single direction, are drawn separately. For every line entering a vertex in a certain direction, there must be another line entering the vertex in the opposite direction, otherwise the line is counted as an end. The examples shown in Fig. 4(a), (b) and (c) give $3, 2$ and $0$ ends, respectively. The objective function is:

$$M_3(p) = \sum_{c\in\mathcal{C}} \tfrac{1}{2} \times |\text{edges}(p,c) - \text{edges}(p,\overline{c})| + \sum_{q\in\text{ch}(p)} 2 \times \text{bends}(q,p)$$

where edges$(p,c)$ is the number of $c$-directed lines entering $p$, and edges$(p,\overline{c})$ is the number of lines entering $p$ in the opposite direction to $c$. The factor of $\frac{1}{2}$ ensures counting ends in each pair of opposing directions only once.

**Problem $\pi_4$: Parallel lines merge to line on opposite side.** Problem $\pi_4$ assumes that lines entering a vertex from the same direction are allowed to merge without cost inside the vertex, given the presence of at least one line in the opposing direction. If there is no line in a given direction, an end is counted for every line in the opposing direction. If one or more lines are placed in a pair of opposing directions, there are no ends.

$$M_4(p) = \sum_{q\in N(p)} \text{ends}_4(q,p) + \sum_{q\in ch(p)} 2 \times \text{bends}(q,p)$$

where $\text{ends}_4(q,p) = 0$ if there is any edge entering $p$ from the opposite direction to the edge $(q,p)$, or 1 otherwise. In this problem, the examples in Fig. 4(a), (b) and (c) produce $3, 0$ and $0$ ends respectively.

**Problem $\pi_5$: Parallel lines always merge.** The final problem, $\pi_5$, assumes that lines entering a vertex from the same direction are drawn as a single line; that is, parallel lines merge to a single line *before* a vertex. As soon as at least one line is present in the opposite direction, no ends are counted. $M_5(p)$ is equal to:

$$\tfrac{1}{2} \sum_{c\in\mathcal{C}} |\text{sign}(\text{edges}(p,c)) - \text{sign}(\text{edges}(p,\overline{c}))| + 2 \sum_{q\in\text{ch}(p)} \text{bends}(q,p)$$

where $\text{sign}(x) = 1$ if $x > 0$, or 0 otherwise. For the examples in Fig. 4, the number of ends are $1, 0$ and $0$ for (a), (b) and (c) respectively.

## 4 Algorithms for Tree Schematisation

In this section, we present an algorithm to produce a schematisation of a given tree, without considering the specific objective function being used. The algorithm is then modified for each of the problems $\pi_1$ - $\pi_5$ to provide a more efficient solution in each case. In all cases, we use the grid discretisation introduced in Section 1. First, we define an important concept used throughout this section.

**Definition 2.** *Given a set of directions $\mathcal{C}$, a $\mathcal{C}$-directed path between two points $p$ and $q$ is a polygonal chain from $p$ to $q$ in which every line segment is parallel to some direction in $\mathcal{C}$.*

## 4.1 A General Solution

Start by choosing an arbitrary leaf vertex to be the root vertex $p_{root}$ of the tree $T$. Perform a post-order traversal on $T$, visiting each point $p \in \mathcal{S}$ once from the leaves to the root. The algorithm finds optimal solutions for each subtree $T_p$ of $T$ rooted at $p$, and during its traversal of $T$ propagates these solutions upward, until a complete solution has been generated at the root vertex. We use the term *$M$ cost* to denote the value of the objective function $M(p)$. The $M$ cost of a subtree of $T$ is the summed $M$ cost of all points in the schematisation of the subtree.

For each vertex $p$, given a grid point $p' \in G_\varepsilon(p)$ and an direction $c \in \mathcal{C}$, let $M_{min}(p, p', c)$ be the minimum $M$ cost of the subtree of $T$ rooted at $p$, assuming that the schematisation passes through $p'$, and that the path from $p$ to $p$'s parent will leave $p'$ in direction $c$. We will refer to the direction $c$ as the *parent direction* from $p$. We assume in this description that there is only one edge from $p$ to $p$'s parent, and hence only one $\mathcal{C}$-directed path can be constructed in the schematisation. In the problem $\pi_2$, $p$ may have multiple edges to its parent, and the schematisation may use different directions for each. In this case, we perform the same steps several times, once for each edge. This process is further detailed in Section 4.2.

When the algorithm visits the vertex $p$, it computes and stores for every point $p' \in G_\varepsilon(p)$, and for every parent direction $c \in \mathcal{C}$ the value of $M_{min}(p, p', c)$, along with the corresponding solution. If $p$ has $k$ children, then $k$ paths must be stored for each solution. Thus $\mathcal{O}(\frac{|\mathcal{C}|kr^2}{\varepsilon^2})$ values are stored at each vertex $p$.

As introduced in Section 1, the objective function being minimised by the algorithm may be divided into a count of *bends* and *ends*. We can count the number of bends by considering each child $q \in \text{ch}(p)$ in turn.

Given two grid points $p' \in G_\varepsilon(p)$, $q' \in G_\varepsilon(q)$, let $\delta(q', p', c_1, c_2)$ be the minimum number of bends needed by a $\mathcal{C}$-directed path from $q'$ to $p'$ that starts in direction $c_1 \in \mathcal{C}$ and ends in direction $c_2 \in \mathcal{C}$. This value can computed in $O(1)$ time. Compute $\delta(q', p', c_1, c_2)$ for every pair of grid points $p' \in G_\varepsilon(p)$, $q' \in G_\varepsilon(q)$ and for each pair of directions $c_1, c_2 \in \mathcal{C}$. These values are the minimum number of bends required for any possible path from $q$ to $p$. Now add $M_{min}(q, q', c_1)$ to these and store the minimum solution for every child $q$, grid point $p'$, final path direction $c_2$, and parent direction $c$. To calculate $M_{min}(p, p', c)$, it remains only to count the number of ends.

Unlike bends, ends cannot generally be counted independently for each of $p$'s children. Let $k$ be the number of children of $p$. If $k \leq 1$, i.e. $p$ is a leaf vertex or a vertex with only one child, then only the edge between $p$ and $p$'s parent, or its interaction with the single child edge, needs to be considered in calculating the number of ends.

If $k > 1$ then the solution will, in general, be far more complicated, since the end cost of the $\mathcal{C}$-directed path from each child will depend on the paths from the other children. It can be shown that it is NP-hard to compute the solution with minimum end cost, even in a somewhat restricted case (See Section 4.5).

A brute force approach can be used to explore all solutions over the entire set of $p$'s children. There are $\binom{|\mathcal{C}|k}{k}$ such solutions, which is a number exponential in $k$. Sections 4.2 - 4.5 give methods to compute minimum end cost solutions for $\pi_1$ - $\pi_5$ that need only time polynomial in $k$.

Once $p_{root}$ has been processed, the entire tree has been traversed. Now find the minimum value of $M_{min}(p_{root}, p', c)$ over all grid points in $G_\varepsilon(p_{root})$. A schematisation can be traced back through the tree, from root to leaves, following stored solutions of minimum cost at each vertex.

Since we start by finding a locally optimal solution at each leaf vertex, and then augment these solutions to find an optimal solution for the subtree rooted at every parent vertex, it is clear by induction that the solution obtained at $p_{root}$ is optimal for the entire tree.

**Time complexity.** The time complexity of the algorithm depends on the specific problem being solved. However, a minimum time complexity of $\Omega(\frac{r^4}{\varepsilon^4}|\mathcal{C}|^3k)$ for processing each vertex $p$ can be noted, as in all five cases $\delta(q', p', c_1, c_2)$ values must be computed for the $k$ children of $p$. This equates to $\Omega(\frac{r^4}{\varepsilon^4}|\mathcal{C}|^3n)$ over an entire tree of $n$ vertices.

## 4.2 Problems $\pi_1$ and $\pi_2$

The objective function $M_1(p)$ counts bends and ends only along each individual coloured path. Hence, the solution for one path has no effect on the solution for another, and they may be treated independently. $M_2(p)$ is the same, but multiple edges must be taken into account.

**The algorithm.** For any coloured path containing the vertex $p$, one can consider all solutions in polynomial time, since there are at most two edges of that colour incident to $p$. Find the best solution for every colour, and sum the $M_1$ costs to get $M_{min}(p, p', c)$ over all of $p$'s children.

In $\pi_2$, there may be multiple coloured edges from $p$ to $p$'s parent. In this case, consider a different parent direction for each colour in turn, and repeat the above process. As the colours are independent, choose the best solution for each and sum them to get the total $M_2$ cost.

**Time complexity.** We compute the $M_{min}(p, p', c)$ values for each $p \in \mathcal{S}$ in time $\mathcal{O}(\frac{r^4}{\varepsilon^4}|\mathcal{C}|^3k)$. We visit each of the $n$ points in the tree only a constant number of times, so a total of $\mathcal{O}(\frac{r^4}{\varepsilon^4}|\mathcal{C}|^3n)$ time is needed to solve $\pi_1$. The time complexity for $\pi_2$ increases in the worst case by a factor of $|\mathcal{P}|$, giving $\mathcal{O}(\frac{r^4}{\varepsilon^4}|\mathcal{C}|^3|\mathcal{P}|n)$ overall.

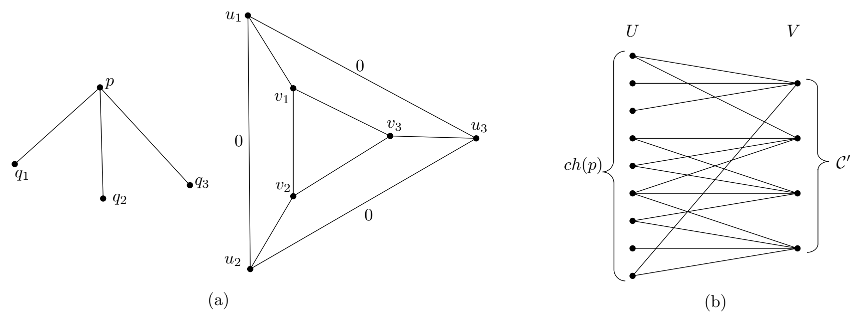


**Fig. 5.** (a) A vertex $p$ with three neighbours $q_1, q_2, q_3$ (left) and the corresponding matching graph $G = (V, E)$ (right). (b) The bipartite matching graph $G = (U, V, E)$ constructed for $\pi_4$. Every node in $V$ is connected by $|V| = |\mathcal{C}'|$ edges to nodes in $U$, representing the $|\mathcal{C}'|$ lowest cost paths entering $p$ in the corresponding direction.

## 4.3 Problem $\pi_3$

In order to minimise the cost function $M_3(p)$, we want to match each individual $\mathcal{C}$-directed path entering $p$ with a different path coming in to $p$ in the opposite direction. We only want to do this if the number of bends on the edges does not overcome the reduced end cost, however.

We propose a transformation from this problem to a *minimum cost maximum cardinality matching* problem. A matching on a graph is a set of vertex-disjoint edges. A maximum cardinality matching is such a set that is as large as possible. A minimum cost maximum cardinality matching is a maximum cardinality matching in which the sum of edge weights is minimal. A detailed introduction to graph matching can be found in the literature [9]. Gabow [6] details an algorithm to solve weighted matching problems for a graph $G = (V, E)$ in time $\mathcal{O}(|V|(|E| + |V| \log |V|))$.

**The algorithm.** Let $q_1, q_2, \ldots, q_{|N(p)|}$ be the neighbouring vertices of $p$ in $T$. Consider in turn each $q_i, 1 \leq i \leq |N(p)|$. For every direction $c_2 \in \mathcal{C}$, compute the minimum $M_3$ cost assuming that the minimum-bend $\mathcal{C}$-directed path from $q_i$ to $p$ enters $p$ in direction $c_2$ and that there is no matching path in the direction opposite to $c_2$. Store the minimum of these values, and the associated direction. Consider now every pair of $p$'s neighbours $q_i, q_j, 1 \leq i \leq |N(p)|, 1 \leq j \leq |N(p)|$, and similarly compute and store the minimum $M_3$ cost solution for every $c_2 \in \mathcal{C}$ assuming that the $\mathcal{C}$-directed path from $q_i$ enters $p$ in direction $c_2$ and the path from $q_j$ enters in the direction opposite to $c_2$. Ties for the minimum cost may be broken arbitrarily. If either $q_i$ or $q_j$ is $p$'s parent, consider only the parent direction $c$ for that path from that vertex.

Construct a graph $G = (V, E)$ as follows (See Fig. 5(a)). Add a node $v_i$ to $V$ for every $q_i$. Add an edge to $E$ between every pair of nodes $v_i, v_j$, with a weight of the minimum $M_3$ cost if the $\mathcal{C}$-directed paths from $q_i$ and $q_j$ enter $p$ from opposite directions. This allows each path to be matched in $G$, with the same cost as if they entered $p$ in opposite directions.

Next add a second node $u_i$ to $V$ for every $q_i$. Add an edge to $E$ between every pair of nodes $u_i, v_i$ with a weight of the minimum $M_3$ cost if the path from $q_i$ to $p$ is unmatched by a path entering $p$ in the opposite direction. Finally, add a zero weight edge between every node pair $u_i, u_j$.

Compute a minimum cost maximum cardinality matching $\eta$ on $G$. Transform back to the original problem by considering each pair of matched nodes in $\eta$. If both nodes correspond to the same child $q_i$, then the direction that gives the minimal $M_3$ cost independent of other children is taken for $q_i$'s path to $p$. Otherwise, the nodes correspond to two different children in the original problem, and the their paths are made to enter $p$ in the pair of opposing directions that gave minimal $M_3$ costs.

**Time complexity.** The graph $G$ takes $\mathcal{O}(|\mathcal{C}|k^2)$ time to generate, and contains $\mathcal{O}(k)$ nodes and $\mathcal{O}(k^2)$ edges. Computing the matching therefore takes $\mathcal{O}(k^3)$ time. Hence, the total time complexity is $\mathcal{O}(\frac{r^4}{\varepsilon^4}|\mathcal{C}|^3 n + \frac{r^2}{\varepsilon^2}\sum_{p\in\mathcal{S}}(|\mathcal{C}|d_T(p)^3 + |\mathcal{C}|^2 d_T(p)^2))$, where $d_T(p)$ is the degree of $p$ in $T$.

## 4.4 Problem $\pi_4$

In contrast to $\pi_3$, in $\pi_4$ we need only one path entering $p$ in a given direction to match many in the opposite direction.

Throughout this section, we say that a direction $c$ is *used* by a schematisation if there is a $\mathcal{C}$-directed path in the schematisation from at least one neighbour of $p$ that enters $p$ in direction $c$. The direction $c$ is *not used* if there is no such path.

Our approach again uses a transformation to a graph matching problem, this time matching edges to each of a subset $\mathcal{C}' \subseteq \mathcal{C}$ of directions. The matching problem assumes that an optimal solution exists that uses every direction in the given subset $\mathcal{C}'$, and does not use any other directions. To find a globally optimal solution, the algorithm processes all $2^{|\mathcal{C}|}$ subsets of $\mathcal{C}$ (assuming $k \geq |\mathcal{C}|$, otherwise only subsets of cardinality at most $k$ are considered). We expect $|\mathcal{C}|$ to be a small constant in most practical applications.

**The algorithm.** Process every subset $\mathcal{C}' \subseteq \mathcal{C}$ for which $|C'| \leq k$ as follows. Let $q_1, q_2, \ldots, q_k$ be the children of $p$ in $T$. For each $q_i, 1 \leq i \leq k$ and every direction $c \in \mathcal{C}'$, calculate the $M_4$ cost if the $\mathcal{C}$-directed path from $q_i$ enters $p$ in direction $c$, with no path entering $p$ in the opposite direction. For now, store the solution that gives the lowest $M_4$ cost, breaking ties arbitrarily.

Once all children have been processed, an initial schematisation is constructed from the set of stored solutions. If this schematisation uses every direction $c \in \mathcal{C}'$, then we are done; there is no possibility of reducing the end count, as an end will only be counted if there is an unused direction. If one or more directions are unused, we must satisfy our assumption of an optimal solution using exactly the set of directions $\mathcal{C}'$, by choosing some of the paths to enter $p$ in the unused directions.

Compute for every direction $c \in \mathcal{C}'$ the set of $|\mathcal{C}'|$ children whose $\mathcal{C}$-directed paths to $p$ give the lowest additional $M_4$ cost if required to enter $p$ in direction $c$. Call this set $\mathrm{ch}_c(p)$. The additional $M_4$ cost for each child $q \in \mathrm{ch}_c(p)$ will be the cost difference between the initial schematisation and the one obtained if the path from $q$ enters $p$ in direction $c$.

Now construct a bipartite graph $G = (U, V, E)$ for matching (See Fig. 5(b)). For each child $q_i$ that was added to $\mathrm{ch}_c(p)$ for any $c \in \mathcal{C}'$, add exactly one node $u_i$ to $U$. For every direction $c_j \in \mathcal{C}'$, add one node $v_j$ to $V$. Construct an edge between every pair of nodes $u_i \in U, v_j \in V$ for which it holds that $u_i \in \mathrm{ch}_{c_j}(p)$. Set the weight of the edge $u_i, v_j$ equal to the additional $M_4$ cost for the child $q_i$'s $\mathcal{C}$-directed path to enter $p$ in direction $c_j$.

$G$ now contains $\mathcal{O}(|\mathcal{C}'|^2)$ nodes and $\mathcal{O}(|\mathcal{C}'|^2)$ edges. Compute a matching from $U$ to $V$ as in Section 4.3. Once a matching is computed, modify the initial schematisation as follows. For each matched pair of nodes $u_i \in U, v_j \in V$, replace the initial path from $q_i$ to $p$ with a minimum-bend $\mathcal{C}$-directed path that enters $p$ in direction $v_j$. All other paths remain as in the initial schematisation. We now have a schematisation that uses every direction in the set $\mathcal{C}'$, and is of minimal cost for $\mathcal{C}'$.

After computing such a schematisation for all subsets $\mathcal{C}' \subseteq \mathcal{C}$, take the schematisation that produces the minimum $M_4$ cost, breaking ties arbitrarily; this is an optimal solution. Note that if $k < |\mathcal{C}|$, the algorithm needs only to consider those subsets $\mathcal{C}' \subseteq \mathcal{C}$ where $|\mathcal{C}'| \leq k$.

**Time complexity.** The algorithm explores $\mathcal{O}(2^{|\mathcal{C}|})$ subsets of directions. For each of these subsets, we can then populate the set $\mathrm{ch}_c(p)$ for every direction $c \in \mathcal{C}'$ in $\mathcal{O}(|\mathcal{C}|^2 k)$ time, since choosing the $i$th largest of $n$ numbers can be done in $\mathcal{O}(n)$ time [1]. The graph $G$ is constructed in $\mathcal{O}(|\mathcal{C}|^2)$ time, and $\mathcal{O}(|\mathcal{C}|^4 |\log|\mathcal{C}|)$ time is needed to compute a matching from $G$. We therefore need $\mathcal{O}(\frac{r^4}{\varepsilon^4}|\mathcal{C}|^3 k + \frac{r^2}{\varepsilon^2}2^{|\mathcal{C}|}(|\mathcal{C}|^6 \log|\mathcal{C}| + |\mathcal{C}|^4 k))$ time to process the vertex $p$, and $\mathcal{O}(\frac{r^4}{\varepsilon^4}|\mathcal{C}|^3 n + \frac{r^2}{\varepsilon^2}2^{|\mathcal{C}|}|\mathcal{C}|^6 \log|\mathcal{C}| n)$ time for $T$.

## 4.5 Problem $\pi_5$

$\pi_5$ counts 1 end for each direction used if the opposing direction is not used, or 0 otherwise. Let $\mu(p', q', q, c_1, c_j) = M_5(q, q', c_1) + \delta(q', p', c_1, c_j)$ be the $M_5$ cost calculated by the algorithm of Section 4.1, before counting ends.

In order to count ends, we need only consider directions $c_j \in \mathcal{C}$ that correspond to a cost of at most 1 more than the minimum $\mu(p', q', q, c_1, c_j)$ value. By using two opposite directions for the $\mathcal{C}$-directed paths from two children $q_1, q_2 \in \mathrm{ch}(p)$, we can save at most 2 from the $M_5$ cost. Hence, if either path entering $p$ in those directions had a $\mu(p', q', q, c_1, c_j)$ value of 2 or more greater than the optimal, there must be a pair of unopposed directions for the two paths with equal or lower $M_5$ cost.

Given this fact, $\pi_5$ appears somewhat simpler than $\pi_4$. However, it can be shown to be NP-hard. Let ChildPathPlacement be the problem of choosing a direction $c_j \in \mathcal{C}$ for the path from each $q_i \in \text{ch}(p)$ such that the corresponding schematisation gives a total $M_5$ cost of $m$.

**Theorem 1.** ChildPathPlacement *is NP-complete.*

The proof of Theorem 1 is by a straightforward reduction from SetCover; we omit it here due to space restrictions.

It is clear that ChildPathPlacement can be solved if the $\pi_5$ problem can be solved, which leads to our final result for this section.

**Corollary 1.** $\pi_5$ *is NP-hard.*

$\pi_5$ can be solved using the same algorithm as $\pi_4$.

## References

1. M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan. Time bounds for selection. Journal of Comp. and Sys. Sciences, vol. 7, no. 4, p. 448–461, 1973.
2. U. Brandes, M. Eiglsperger, M. Kaufmann and D. Wagner. Sketch-driven orthogonal graph drawing. In Proc. Graph Drawing 2002, p. 1–11, 2002.
3. S. Cabello, M. de Berg and M. van Kreveld. Schematization of networks. Computational Geometry and Applications, vol. 30, p. 223–238, 2005.
4. S. Cabello and M. van Kreveld. Approximation algorithms for aligning points. Algorithmica, vol. 37, p. 211–232, 2003.
5. M. Eiglsperger, S. P. Fekete and G. W. Klau. Orthogonal graph drawing. Drawing Graphs, p. 121–171, Springer-Verlag Berlin Heidelberg, 2001.
6. H. N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In Proc. ACM-SIAM Symp. Discrete Alg., p. 434–443, 1990.
7. S.-H. Hong, D. Merrick and H. A. D. do Nascimento. The metro map layout problem. In Proc. Graph Drawing 2004, p. 482–491, 2004.
8. U. Lauther and A. Stübinger. Generating schematic cable plans using springembedder methods. In Proc. Graph Drawing 2001, p. 465–466, 2002.
9. L. Lovász and M. D. Plummer. Matching Theory. Elsevier, Amsterdam, 1986.
10. D. Merrick and J. Gudmundsson. Path simplification for metro map layout. Submitted to Graph Drawing, June 2006.
11. G. Neyer. Line simplification in restricted orientations. In Proc. of the 6th International workshop on Algorithm s and Data Structures, p. 13–24, 1999.
12. M. Nöllenburg and A. Wolff. A mixed-integer program for drawing high-quality metro maps. In Proc. Graph Drawing 2005, p. 321–333, 2006.
13. J. M. Stott and P. Rodgers. Metro map layout using multicriteria optimization. In Proc. Information Visualisation, p. 355–362, 2004.

# Trees with Convex Faces and Optimal Angles

Josiah Carlson and David Eppstein

Computer Science Department, University of California, Irvine
{jcarlson,eppstein}@uci.edu

**Abstract.** We consider drawings of trees in which all edges incident to leaves can be extended to infinite rays without crossing, partitioning the plane into infinite convex polygons. Among all such drawings we seek the one maximizing the angular resolution of the drawing. We find linear time algorithms for solving this problem, both for plane trees and for trees without a fixed embedding. In any such drawing, the edge lengths may be set independently of the angles, without crossing; we describe multiple strategies for setting these lengths.

## 1 Introduction

Suppose we wish to draw a tree in the plane by first setting the slopes of its edges, and then independently setting the edge lengths. For what choices of slopes are we guaranteed that the drawing will be non-self-crossing, no matter what lengths are chosen?

To answer this question, we define a *convex arch* to be a polygonal chain spanning a range of angles of at most $\pi$, so that the edges occur on the chain in sorted order by their angles within that range (Figure 1). We say that a tree drawing has *convex faces* if the path between all consecutive pairs of leaves (in the radial order of leaves around the root of the tree) is a convex arch.

If a tree drawing has convex faces, then the edges incident to the leaves may be extended to infinite rays, transforming each arch into an infinite convex polygon. These polygons partition the plane, similarly to the partition formed by a farthest neighbor Voronoi diagram (Figure 2; see also [1] for Voronoi related tree drawing algorithms). There can be no crossings within any of these convex faces, so the drawing is planar, regardless of the drawing's edge lengths. Conversely, any partition of the plane into finitely many infinite convex polygonal faces comes from a tree drawing in this way.

Tree drawings with convex faces, such as the one in Figure 3, can be visually appealing. For instance, the convexity of the faces makes it easy to visually separate vertices in different subtrees of the tree. However in terms of the *angular resolution* of a drawing (the minimum angle between any two edges incident on the same vertex [2]) drawings with convex faces may require much sharper angles than nonconvex drawings, and we wish to alleviate this fault by using as wide angles as possible. This motivates the main problem we consider here:

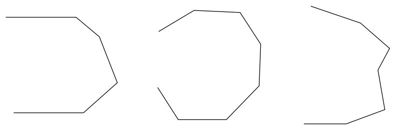


**Fig. 1.** Left: a convex arch. Center: not a convex arch, as the angles span a range larger than $\pi$. Right: not a convex arch, as the angles of the segments are not in sorted order.

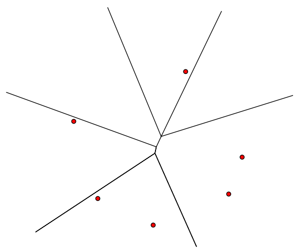


**Fig. 2.** The farthest point Voronoi diagram partitions the plane into infinite polygonal cells, much like the partition formed by extending the leaf edges of a tree drawing with convex faces

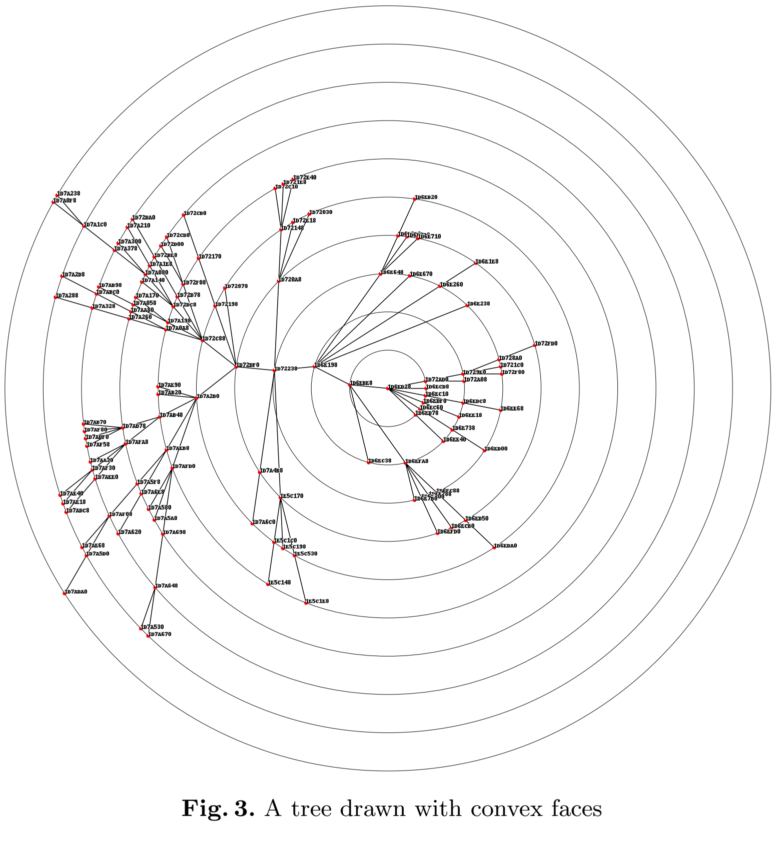


**Fig. 3.** A tree drawn with convex faces

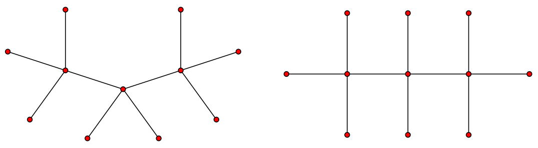


**Fig. 4.** Different embeddings of a tree can have different optimal angular resolutions

**Tree Drawing With Convex Faces and Optimal Angles:**
Given as input a tree $T$, find a drawing of $T$ with convex faces, having the maximum angular resolution possible among all such drawings.

We consider two different versions of this problem. In one version a plane embedding of $T$ is given by ordering the edges at each vertex; this ordering must be respected by the drawing. In the other version, the edges at each node of $T$ may be permuted arbitrarily. The optimal angular resolution may differ depending on which version of the problem we consider; for instance, in the tree shown in Figure 4, the embedding on the left has optimal angular resolution $2\pi/5$ while the right embedding has optimal angular resolution $\pi/2$. Our main results are that the optimal angular resolution, and a drawing achieving that resolution, may be found in linear time, both for the plane case and the unembedded case.

Our tree drawing algorithms set the slopes of all edges in the drawing, before setting the lengths of the edges and placing the vertices. For the slope-setting phase, the definition of having convex faces and the angular resolution do not depend on the choice of root for $T$, so we may assume that initially $T$ is unrooted. However, it will be useful to choose a particular root, depending on the structure of $T$. After the slopes are set, we may return to the original root of $T$ (if it has one) and use that information when we set edge lengths and place vertices.

The problem of choosing slopes so that any setting of edge lengths is non-crossing can also be solved by drawings in which the faces are non-convex, as long as paths between consecutive leaves have ranges of angles of at most $\pi$; for instance the nonconvex path on the right of Figure 1 is always non-crossing. However, this additional generality does not allow for improved angular resolution, so we restrict our attention to drawings with convex faces henceforth.

## 2 Paths and Rakes

Before describing our main algorithm, we treat some special cases that are problematic for it. These same cases, as subtrees of our input tree, also play a key role in our main algorithm itself.

A *path* is a tree in which all nodes have degree at most two. Clearly, the optimal angular resolution for a drawing of a path with convex faces is $\pi$, achieved by a drawing in which all vertices lie on a common line.

We define a *rake* to be a tree in which all nodes have degree at most three, and in which some path connects all degree-three vertices. Let $T$ be a rake, and

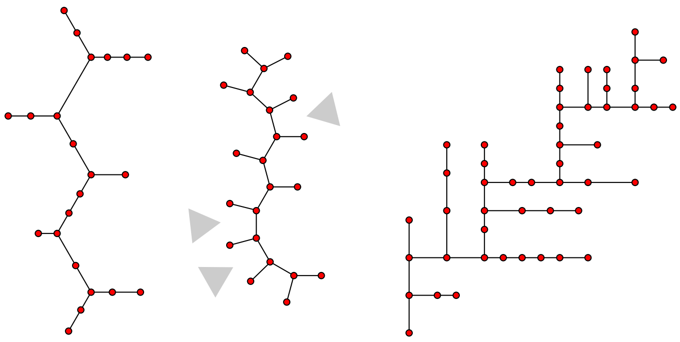


**Fig. 5.** Optimal angular resolution drawings of rakes. Left: a rake with no double turns, with angular resolution $2\pi/3$. Center: a rake with three double turns (marked by the gray triangles), requiring angular resolution $7\pi/12$. Right: a construction with angular resolution $\pi/2$ for any rake.

let $P$ be a minimal directed path connecting all degree three vertices of $T$. If $T$ is embedded in the plane, each degree-three vertex $v$ interior to $P$ has one incoming edge in $P$, one outgoing edge in $P$, and one unoriented edge not belonging to $P$. We say that $P$ makes a *left turn* at $v$ if the clockwise ordering of these three edges is the incoming edge, the outgoing edge, and the unoriented edge, and we say that $P$ makes a *right turn* at $v$ otherwise. We define a *double turn* to be a pair of consecutive turns in $P$ that are both left or both right.

**Lemma 1.** *An unembedded rake has optimal angular resolution $\frac{2\pi}{3}$. An embedded rake with $k$ double turns has optimal angular resolution $\pi(\frac{1}{2} + \frac{1}{6+2k})$.*

*Proof.* If a rake has no double turns, it may be drawn with its edges lying on the edges of a tiling of the plane by regular hexagons, as shown in Figure 5(left), and if the input is an unembedded rake then we may choose an embedding in which the turns alternate left and right and achieve this angular resolution. This is clearly optimal for any tree with degree three nodes.

For an embedded rake with double turns, such as the one in Figure 5(center), we consider the sequence of angles between consecutive leaves of the tree. These angles must be nonnegative and total $2\pi$. If the angular resolution is $\frac{\pi}{2} + \epsilon$, then the angle between the two paths incident to a degree three node that is not a turn is at least $\frac{\pi}{2} + \epsilon$. The angle between one of these paths and the path incident to the nearest turn is at least $2\epsilon$, because these two paths are connected via two angles of at least $\frac{\pi}{2} + \epsilon$. Similarly, the angle between the two paths in a double turn is at least $2\epsilon$. Remaining pairs of consecutive leaves may be parallel. Adding all these angles, we get $2(\frac{\pi}{2} + 3\epsilon)$ for the angles near the ends of the paths, and $2\epsilon$ for each double turn, for a total of $\pi + (6 + 2k)\epsilon$. Since this must equal at most $2\pi$, an upper bound on angular resolution of the stated form follows.

To achieve this bound, first assign angles to the leaves of the tree exactly matching the formula above: $\frac{\pi}{2} + \epsilon$ between the two paths incident to a degree

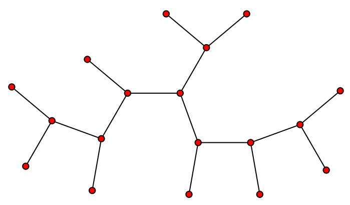


**Fig. 6.** A triple rake with one double turn and one short path. The optimal angular resolution for this tree as embedded is $5\pi/18$.

three node that is not a turn, etc. The path edges are then assigned angles of $\frac{\pi}{2} + \epsilon$ from the preceding leaf. With this angle assignment, all paths between consecutive leaves are seen to form convex arches, so, as we have already discussed, we may assign edge lengths arbitrarily resulting in a tree drawing with convex faces and the stated angular resolution. □

When $k$ is large, the angular resolution $\pi(1/2 + 1/(6 + 2k))$ closely approaches $\pi/2$. A very simple construction achieves angular resolution $\pi/2$ for any rake: simply draw the path connecting the degree three nodes by a monotonic path consisting of alternating and horizontal line segments, with the path proceeding horizontally after each right turn and vertically after each left turn, as shown in Figure 5(right). More generally, the same algorithm shows the following:

**Lemma 2.** *Let $T$ be a rake, and let two slopes $\theta_1$ and $\theta_2$ be given. Then $T$ may be drawn with all edges having slopes $\theta_1$ or $\theta_2$, so that all faces of $T$ except the outer face at its root are convex.*

When $|\theta_1 - \theta_2| \leq \pi/2$, the angular resolution of the drawing produced by Lemma 2 is $|\theta_1 - \theta_2|$. We will use this construction as part of our algorithm for drawing trees that are not rakes.

## 3 Triple Rakes

Given a tree $T'$, in which the maximum vertex degree is three, let $T'$ be the minimal spanning subtree of the degree three vertices in $T$. $T$ is a rake if and only if $T'$ is a path, but the next simplest case is when $T'$ contains a single degree three vertex $t$. In this case, if we root $T$ at $t$, the three subtrees descending from $t$ are rakes, so we call $T$ a *triple rake* (Figure 6).

If $T$ is embedded, then in each of the three paths of $T'$, oriented from $t$ to a leaf, we may define left turns, right turns, and double turns as we did in rakes. Additionally, we define a *short path* in $T'$ to be a path with no turns; that is, a single degree three vertex connected by paths to $t$ and to two leaves.

**Lemma 3.** *If $T$ is an unembedded triple rake with $s$ short paths, its optimal angular resolution is $\pi(\frac{1}{2} + \frac{1}{2(9-2s)})$. If $T$ is an embedded triple rake with $s$ short paths and $d$ double turns, its optimal angular resolution is $\pi(\frac{1}{2} + \frac{1}{2(9-2s+2d)})$.*

*Proof.* The unembedded case follows from the embedded case, by embedding the tree with no double turns. For the embedded case, we assume that the optimal angular resolution is $\pi/2 + \epsilon$, and count the number of times the angles increase by $\epsilon$ as we progress around the leaves of a drawing of the tree, as we did in Lemma 1. The angle between the two bottom leaves of a rake is at least $\pi/2 + \epsilon$, and in a rake that does not come from a short path the angle between one of the two bottom leaves and the next nearest path is another $2\epsilon$. Additionally, as in Lemma 1, each double turn leads to an angle increase of $2\epsilon$. The total angle increase as we go around the tree is $3\pi/2 + (9 - 2s + 2d)\epsilon = 2\pi$, from which the bound follows. This analysis fixes the angles of all leaves of the tree, from which it is straightforward to find a drawing achieving the stated bound. □

## 4 Plane Trees

We are ready to describe our general bound for the optimal angular resolution of a plane embedded tree. We assume our tree $T$ is not a path, rake, or triple rake, as those special cases were handled in previous sections. As the problem of tree drawing with convex faces does not depend on the root of the tree, we choose a new root $r$ as follows:

– If $T$ contains a vertex incident to four or more edges, we let $r$ be any such vertex.
– Otherwise, let $T'$ be the minimal subtree of $T$ containing all degree-three vertices in $T$; $T'$ can be formed by removing from $T$ all leaves of $T$, and all paths of degree-two vertices in $T$ that lead to a leaf. $T'$ cannot be a path, as we have assumed that $T$ is not a rake. Therefore, there exists a vertex in $T'$ with degree three. We let $r$ be any such vertex.

Once we have rooted $T$ at $r$, we consider for each node $v$ and each child $w$ of $v$ the subtree $T_w$ formed by $v$, $w$, and all descendants of $w$. It will be important for our algorithms to be able to determine whether $T_w$ is a path or rake.

**Lemma 4.** *For all $w$ we can determine whether $T_w$ is a path, a rake, or a tree that is not a path or rake, in total time $O(n)$.*

The algorithm for performing this determination is a simple bottom-up calculation on $T$; we omit the details.

We define a *fork at $v$* to be a subsequence of two or more children $w_i$ of $v$, contiguous in the ordering of the children given by the plane embedding of $T$ such that the trees $T_{w_i}$ for the first and last child in the subsequence are paths and all intermediate trees are rakes (Figure 7). We can also identify a fork with a subtree, formed by $v$, the subsequence of children in the fork, and all descendants of those children. When $v$ is not the root, the sequence of children of $v$ is a linear order, but for the root $v = r$ we consider this sequence as a cyclic order and allow any linear subsequence of this order. In particular, when $r$ has one child forming a path subtree and all its other children form rakes, we consider the sequence starting at the path, continuing through all the rakes, and ending at the path again to form a fork.

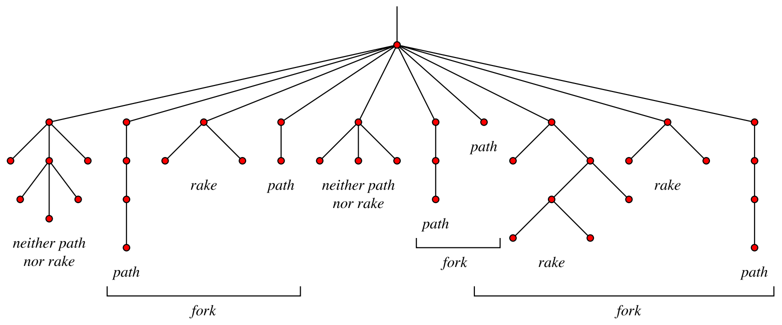


**Fig. 7.** A vertex with three forks. The subtrees descending from the top vertex of the figure contain additional forks.

**Lemma 5.** *If $T_w$ is a rake, it contains exactly one fork, at the bottommost vertex with two children. Otherwise there are at least two forks at vertices of $T_w$.*

*Proof.* We use induction on the height of $T_w$. If $w$ has one child $x$, and $T_w$ is not a rake, then $T_x$ is also not a rake, and the result follows. If $w$ has two children, and $T_w$ is not a rake, then either one child $x$ is not a rake, and again the result follows, or both children are rakes, and we have one fork in each. If $w$ has three or more children, one of which is neither a path nor a rake, then we have two forks in that child alone. If there are two or more rakes, then again we have one fork in each. If there is only one rake descending from $w$, then the other two subtrees descending from $w$ are paths, and we have one fork at $w$ itself and one in the rake. Finally, if all descendants of $w$ form paths, then there are at least two forks at $w$. □

**Lemma 6.** *Let $F$ be a fork at a vertex $v$, containing $r$ rakes, in a tree drawing with convex faces and angular resolution $\theta$. Then the angle between the first leaf and the last leaf in $F$ is at least $(r+1)\theta$.*

*Proof.* That angle is needed just for the $r+2$ edges connecting $v$ to its children in the fork. Adding the remaining edges in the fork cannot decrease the angle any further. □

**Lemma 7.** *Let $T$ be a tree containing $f$ forks, as rooted at $r$. Then any drawing of $T$ with convex faces has angular resolution at most $2\pi/f$.*

*Proof.* We prove more generally that, if $T$ has $f$ forks at some node $v$ or its descendants, in a drawing with angular resolution $\theta$, then the angle between the first and last leaves descending from $v$ is at least $f\theta$. The result comes from applying this bound to the trees descending from the children of the root.

To prove a lower bound of $f\theta$ on the angle between the first and last leaves descending from $v$, we use induction on the height of the subtree. The sequence of slopes of leaves increases monotonically as we proceed clockwise around the tree, increasing (by induction) by $\theta$ times the number of forks in each subtree that is not a path or rake. In a rake that is not part of a fork, the bottommost two paths must have an angle of at least $\theta$, matching the single fork (Lemma 5) that exists in the rake. Finally, each fork at $v$, containing $r$ rakes, leads to a total of $(r+1)$ forks when we include the fork within each rake, and by Lemma 6 the increase in angle in this case again matches the number of forks. □

**Lemma 8.** *In a tree that is not a path, rake, or triple rake, rooted at $r$ as described above, there are at least four forks in the tree.*

*Proof.* If two or more of the trees descending from children of $r$ are not paths or rakes, the result follows from Lemma 5. If $r$ has four or more children, exactly one of which is not a path or rake, then the other three children have two rakes, or form a fork with one rake, or form two forks; in all cases there are two forks from the non-rake child and two from the other three children. If $r$ has four or more children, all of which are paths or rakes, then each path has a fork clockwise of it and each rake has a form inside it, so again the total number of forks is at least four. Finally, if $r$ has degree three, then (by our choice of root) none of the subtrees descending from it is a path, and (since the tree is not a triple rake) at least one of these subtrees is not a rake, so we get two forks from this non-rake subtree and one each from the other two subtrees. □

By Lemmas 7 and 8, the angular resolution of a tree that is not a path, rake, or triple rake is at most $\pi/2$, so we may use the construction of Lemma 2.

**Lemma 9.** *Let $T$ be a tree containing $f$ forks, as rooted at $r$. Then $T$ has a drawing with convex faces and angular resolution $2\pi/f$.*

*Proof.* We assign slopes to the edges of $T$ in postorder. In a subtree containing $f'$ forks, the angle from the first leaf to the last leaf will be $2\pi f'/f$; thus, the total angle around the entire tree will be $2\pi$ as desired. When assigning slopes to the edges of the subtree rooted at $v$, we consider the children of $v$ in order.

Each subtree that is not a path or rake has its first leaf slope equal to that of the leaf immediately preceding the tree, and by induction can be drawn with the stated angle bound. We choose the slope of the edge leading from $v$ to the subtree in such a way that it bisects the angle formed by the subtree's first and last leaves; in this way, the angle between two consecutive edges incident to $v$ is half the angle spanned by the two subtrees, and thus at most $\pi$. In addition, this choice of slope is guaranteed to be at least $2\pi/f$ greater than that of the first leaf in the subtree, and at least $2\pi/f$ less than that of the last leaf in the subtree, so each edge incident to $v$ will form an angle of at least $2\pi/f$ with the preceding and succeeding edges. Finally, we can show that (together with our choice of slopes for other types of subtree) this choice of root slope will always be within the range of slopes of the edges at the child vertex of the subtree.

Each path that is not the second path of a fork is given a slope equal to that of the previous leaf. Each rake that is part of a contiguous sequence of rakes following a path is drawn using Lemma 2 in a way that increases the slope by $2\pi/f$, matching the bound for a subtree with a single fork; we align the root edge of the rake with the slope of its last leaf. Each other rake is drawn by Lemma 2 with its root edge aligned with the slope of its first leaf. Finally, the second path of any fork is assigned a slope greater than the preceding leaf by $2\pi/f$. □

Putting these lemmas together, we have the following result:

**Theorem 1.** *Let $T$ be an unrooted plane tree. Then in time $O(n)$ we can find a drawing of $T$ with convex faces and optimal angular resolution.*

*Proof.* We first test whether $T$ is a path, rake, or triple rake. If it is a path, we embed it trivially with angular resolution $\pi$. If it is a rake, we apply Lemma 1, and if it is a triple rake, we apply Lemma 3. Otherwise, we root $T$ as described at the start of this section, determine which subtrees are paths and rakes, count forks at each node, and apply the drawing method described in Lemma 9. The optimality of the angular resolution of this method follows from Lemma 7. □

## 5 Unembedded Trees

We are now ready to handle trees that do not have a plane embedding already fixed. Our choice of root at the beginning of the previous section does not depend on the embedding, so we may use it without change. However, the definition of a fork depends strongly on the embedding. Our goal in finding an embedding maximizing the angular resolution is to minimize the number of forks. To do so, define the *excess* of a node $v$ that is not the root of the tree to be $E_v = \max(0, P_v - N_v - 1)$, where $P_v$ is the number of trees descending from $v$ that are paths and $N_v$ is the number of trees descending from $v$ that are neither paths nor rakes. For the root $r$, define $P_r$ and $N_r$ similarly, and let $E_r = \max(0, P_v - N_v)$. The *total excess* $E(T)$ is the sum of the excesses at each node.

**Lemma 10.** *Every plane embedding of tree $T$ has at least $E(T)$ forks. There exists an embedding of $T$ with exactly $E(T)$ forks.*

*Proof.* In any embedding, at node $v$ there are $P_v$ paths, forming $P_v - 1$ potentially-consecutive pairs of paths ($P_v$ pairs in the cyclic order at the root). At most $N_v$ of these pairs can be separated by a subtree that is not a path or a rake, and the remaining pairs form forks, so there are at least $E_v$ forks at $v$ and $E(T)$ overall. This bound may be achieved by, at each node, placing the paths and the trees that are not paths or rakes in alternating order for as many alternations as possible. The placement of rakes in the ordering at each vertex and the ordering of the descendants within each rake does not affect the number of forks in $T$. □

**Theorem 2.** *Let $T$ be an unrooted unembedded tree. An embedding of $T$ and its drawing with convex faces and optimal angular resolution can be found in $O(n)$ time. If $T$ is not a path, rake, or triple rake, the optimal angular resolution among all drawings of $T$ with convex faces is $2\pi/E(T)$.*

*Proof.* We first test whether $T$ is a path, rake, or triple rake. If it is a path, we embed it trivially with angular resolution $\pi$. If it is a rake, we apply Lemma 1,

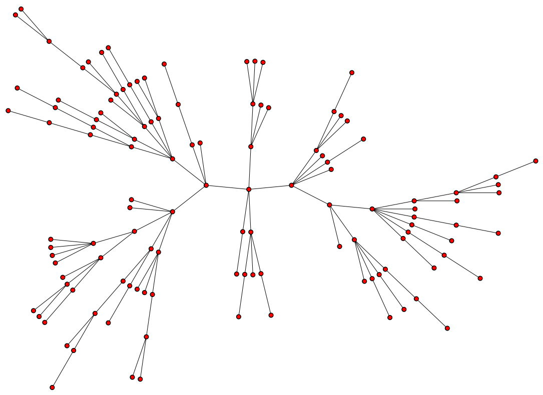


**Fig. 8.** The same tree as Figure 3, drawn with all edge lengths equal

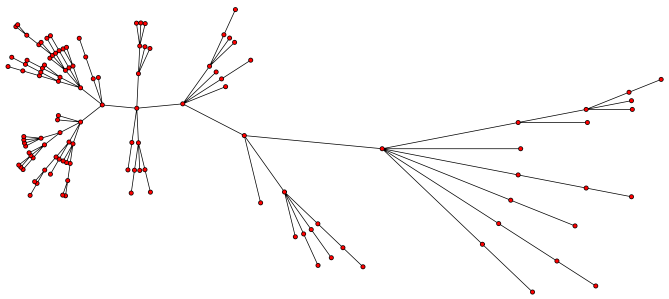


**Fig. 9.** The same tree as Figure 3, drawn with edge length inversely proportional to the distance from the root

and if it is a triple rake, we apply Lemma 3. Otherwise, we root $T$ as described at the start of the previous section, embed it with the minimum number of forks via Lemma 10, and apply Theorem 1 to the resulting embedded tree using the same root. The bound on the angular resolution follows from Lemmas 7, 9 and 10. □

## 6 Setting the Lengths

Although our focus has been on the edge slopes for tree drawings, we can not produce an actual drawing without setting the edge lengths. Our slopes have been chosen specifically to allow any possible setting of edge length, without introducing a crossing, so we may choose these lengths arbitrarily, to advance some aesthetic criterion for the drawing or convey some additional information about the tree. We discuss three possible choices of edge length.

**Uniform Edge Lengths:**
Assigning all edges the same length can produce a pleasantly uniform vertex spacing. Since all vertices are treated equivalently, this type of drawing

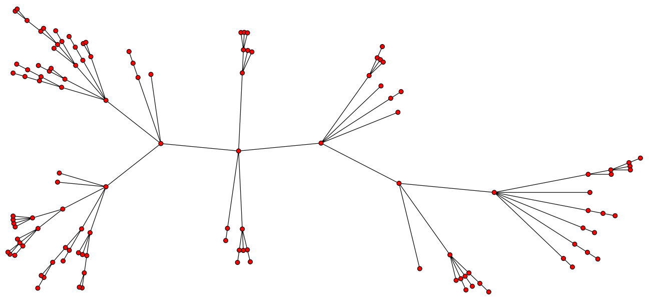


**Fig. 10.** The same tree as Figure 3, drawn with edge length proportional to the square root of the number of descendants of the edge's top vertex

is appropriate for unrooted trees. Trees with uniform edge lengths are shown in Figures 4, 5(center), 6, and 8.

**Radial Drawing:**

We may place the root of a rooted tree at the center of a system of concentric circles, and place each vertex on the circle corresponding to its distance from the root. To do so, we set the lengths of the edges in preorder. When setting the length of an edge $(u, v)$, the placement of $u$ on its circle will already have been fixed. Since the circle for $u$ is inside the circle for $v$, a ray from $u$ in the direction of the slope for edge $(u, v)$ will intersect the circle for $v$ in a unique point, at which we place $v$. We then set the edge length to the distance between the placements of its endpoints. A drawing in this style is shown in Figure 3. This style of tree can be effective in making visually apparent the root of the tree and the distance of each node from the root, especially if the drawing is displayed with the concentric circles visible, as they are in the figure. The root used for this placement algorithm need not be the same as the root $r$ that was chosen in our angle-setting algorithm.

**Position in Tree:**

Similarly to radial drawing, we may choose edge lengths that are functions of the position of the edge in the tree. Two drawings of this type are shown in Figures 9 and 10, with lengths that are functions of the distance from the root and the size of the subtree rooted at the top vertex of the edge respectively. It may also be of interest to vary edge lengths of this type continuously, to morph between multiple viewpoints in the same tree.

**Strength of Connection:**

Since the lengths of edges may be arbitrary, we may use them to convey additional information about the tree being drawn. For instance, if some edges form stronger connections between their vertices than others, we may display this by placing more strongly connected vertices closer together, and more weakly connected vertices farther apart.

## 7 Conclusions and Future Work

We have shown how to draw trees with convex faces and optimal angular resolution, among drawings that allow the edges to be drawn with any slope. However, it may be of interest to require that no edge from a parent to child is directed upwards (*downward drawing*) or that no edge is directed either upwards or to the right (necessary but not sufficient for *dominance drawing*). Similar techniques to the ones here, using a linear rather than circular ordering at the root of the tree, can find drawings of these types, with convex faces (except, in the case of dominance drawing, for the face above and to the right of the root) and optimal angular resolution $\theta/f$ where $\theta$ is the range of allowed edge slopes and $f$ is the number of forks in the tree. We omit the details due to lack of space.

It is also natural to extend our drawing methods to graphs other than trees. It seems likely that similar methods can optimize the angles of *pseudotrees* (connected undirected graphs with a single cycle) drawn so the cycle forms a bounded convex face and all other faces are convex and unbounded. Similarly, a reviewer suggested *Halin graphs*, are formed from a tree with no degree-two vertices by connecting its leaves into a cycle; we can form convex drawings of such graphs by choosing edge lengths for the tree so that the leaves are in convex position, but it is not clear how to optimize the angular resolution of such drawings.

Another reviewer suggested that it may be preferable to draw paths of degree-two vertices with small bends between the edges rather than having all edges lie on the same straight line, in order to make the vertex positions more apparent. Such a preference could be quantified by a modified definition of angular resolution, for instance one that measured the angle between any two consecutive edges as the smallest of the two complementary angles formed by the edges' lines; we expect that techniques similar to ours could be used to optimize this modified angular resolution, although the details would differ and the faces of the resulting drawings would likely be nonconvex.

We may also seek stronger optimality conditions for our drawings. In particular, consider the vector of angles between consecutive edges in a tree drawing, sorted from smallest to largest. Our algorithm finds a vector with first coordinate as large as possible, but can we find the maximum possible vector in the lexicographic ordering of all possible vectors? For plane trees, it seems possible to solve this problem in polynomial time by setting up a sequence of linear programs to optimize each successive coordinate of the vector of angles, but this is neither efficient nor satisfactorily combinatorial. For unembedded trees, a solution to the lexicographic optimization problem seems even more difficult, and we do not know whether it is likely to be polynomial or NP-hard.

## References

1. G. Liotta and H. Meijer. Voronoi drawings of trees. *Computational Geometry: Theory and Applications*, 24(3):147–178, 2003.
2. S. Malitz. On the angular resolution of planar graphs. In *Proc. 24th ACM Symp. Theory of Computing*, pages 527–538, 1992.

# Three-Dimensional Drawings of Bounded Degree Trees[⋆]

Fabrizio Frati and Giuseppe Di Battista

Università di Roma Tre
{frati,gdb}@dia.uniroma3.it

**Abstract.** We show an algorithm for constructing $3D$ straight-line drawings of balanced constant degree trees. The drawings have linear volume and optimal aspect ratio. As a side effect, we also give an algorithm for constructing $2D$ drawings of balanced constant degree trees in linear area, with optimal aspect ratio and with better angular resolution with respect to the one of [8]. Further, we present an algorithm for constructing $3D$ poly-line drawings of trees whose degree is bounded by $n^{1/3}$ in linear volume and with optimal aspect ratio.

## 1 Introduction

The problem of constructing $3D$ drawings of trees with limited volume is interesting both in practice and in theory and it has attracted the attention of several researchers. Since a $2D$ drawing is also a $3D$ drawing then the results known for two-dimensional drawings of trees are still valid in $3D$. However, embedding a $2D$ drawing in three dimensions fills the space only in one of its planes, while one would prefer a drawing uniformally distributed in the embedding space. A widely used measure for expressing this is given by the *aspect ratio* of a drawing, that is the ratio between the maximum and the minimum edge of its bounding box. Clearly, considering a $2D$ drawing of an $n$-nodes tree as a $3D$ drawing yields a bad ($O(n^{1/2})$) aspect ratio.

The state of the art in $2D$ can be summarized as follows. No algorithm is known for drawing an $n$-nodes tree in $O(n)$ area and such a bound is achieved only in special cases. For example, if the degree of the nodes is bounded by $n^{1/2}$, then the algorithm of Garg and Rusu [7] constructs $O(n)$ area straight-line drawings. As another example, complete trees can be drawn straight-line in linear area with the algorithm of Trevisan [8]. Concerning algorithms that work in three dimensions, Felsner et al. [5] have shown how to draw in $3D$ any outerplanar graph and so any tree using linear volume. The drawings constructed by such an algorithm have bad ($O(n)$) aspect ratio. In fact, they lie on the surface of a $O(n)$ length triangular prism. However, the problem of finding linear volume $3D$ drawings of trees with good aspect ratio is still open.

In this paper we contribute to the above problems: (1) In Section 3 we show how to adapt the algorithm in [3] for constructing a linear volume $3D$ drawing of a balanced tree with degree bounded by a constant. The aspect ratio is $O(1)$. (2) As a side effect of our technique we give an algorithm for drawing in $2D$ a balanced tree whose degree is bounded by a constant in linear area, with constant aspect ratio and $\Omega(1/\sqrt{n})$ angular resolution (Section 4). This improves the results of Trevisan that in [8] showed an

⋆ Work partially supported by "Project ALGO-NEXT: Algorithms for the Next Generation Internet and Web: Methodologies, Design, and Experiments", MIUR PRIN.

algorithm for constructing drawings with the same area and aspect ratio, but with only $O(1/n)$ angular resolution. (3) In Section 5, we show how to construct a poly-line $3D$ drawing of a tree with degree bounded by $n^{1/3}$ in $O(n)$ volume and $O(1)$ aspect ratio.

## 2 Preliminaries

We assume familiarity with trees and their drawings [4] and assume that trees are *rooted*. The *degree of a node* is the number of its children. The *degree of a tree* is the maximum degree of one of its nodes. The *height of a tree* is the maximum length (number of nodes) of a path from the root to a leaf. In the following we call $T_h$ a complete tree with height $h$. We call $r_h$ its root and, if the degree of $T_h$ is $k$, $T_{1,h-1}, T_{2,h-1}, \dots T_{k,h-1}$ the subtrees of $T_h$ rooted at the children of $r_h$. We call such children $r_{1,h-1}, r_{2,h-1}, \dots r_{k,h-1}$. For complete trees the number of nodes is a function of $h$ and $k$. Namely, $n = 1 + k + k^2 + \dots + k^{h-1} = \frac{k^h-1}{k-1}$. Hence $k^h = n(k-1)+1$ and so $h = \log_k [n(k-1)+1]$. A *balanced* tree is such that its height is logarithmic in the number of its nodes.

*Grid* drawings, *straight-line* drawings, and *poly-line* drawings are defined as usually ([4]). The *bounding box* $B(\Gamma)$ of a drawing $\Gamma$ is the smallest rectangle ($2D$) or parallelepiped ($3D$) with edges parallel to the coordinate axes, that covers $\Gamma$ completely. We denote by $left(B(\Gamma))$, $right(B(\Gamma))$, $back(B(\Gamma))$, $front(B(\Gamma))$, $bot(B(\Gamma))$ and $top(B(\Gamma))$ the sides of $B(\Gamma)$. In the $2D$ case $x$ grows from *left* to *right* and $y$ from *bottom* to *top*. In the $3D$ case $x$ grows from *left* to *right*, $y$ from *back* to *front* and $z$ from *bottom* to *top*. The *aspect ratio* of $\Gamma$ is the ratio between the maximum and the minimum edge of $B(\Gamma)$. $\Gamma$ is (*strictly*) *upward* in one coordinate direction if, for each node, such coordinate is (less than) not greater than the same coordinate of its children. The *angular resolution* of $\Gamma$ is the minimum angle between two segments incident to the same node. $\Gamma$ satisfies the *subtree separation* property ([1]) if, for any two node-disjoint subtrees of $T$, the bounding boxes of their partial drawings don't intersect. $\Gamma$ satisfies the *tip-over* property ([8]) if, for any node, its children are drawn on a line parallel to one coordinate axis. In the following we call $x$-line, $y$-line or $z$-line a line parallel to the $x$-axis, $y$-axis or $z$-axis, respectively. Analogously, we call $xy$-plane, $xz$-plane or $yz$-plane a plane parallel to the coordinate planes $xy$, $xz$ and $yz$, respectively.

## 3 Three-Dimensional Straight-Line Drawings of Balanced Constant Degree Trees

In the following we show an algorithm to draw a balanced constant degree tree $T$ in three dimensions. First, add extra nodes to $T$ until it is complete. This can be done without altering the height $h$ and the degree $k$ of $T$. Now we have to construct a drawing $\Gamma_h$ of a complete tree $T_h$. This can be done recursively as follows. If $h = 1$, then place $r_1$ in $(0,0,0)$. If $h > 1$, suppose you have drawn $\Gamma_{1,h-1}, \Gamma_{2,h-1}, \dots, \Gamma_{k,h-1}$. We distinguish three cases: (i) if $h \bmod 3 \equiv 2$, then place $\Gamma_{1,h-1}, \Gamma_{2,h-1}, \dots, \Gamma_{k,h-1}$ so that $left(\Gamma_{1,h-1}), \dots, left(\Gamma_{k,h-1})$ are on the same $yz$-plane, so that $back(\Gamma_{1,h-1}), \dots, back(\Gamma_{k,h-1})$ are on the same $xz$-plane and so that $top(\Gamma_{i,h-1})$ is one unit below

$bot(\Gamma_{i+1,h-1}), \forall i$ such that $1 \leq i < k$. Place $r_h$ one unit to the left and on the same $x$ line of $r_{1,h-1}$ (see Fig. 1 (a)); (ii) if $h \bmod 3 \equiv 0$, then place $\Gamma_{1,h-1}, \Gamma_{2,h-1}, \ldots, \Gamma_{k,h-1}$ so that $bot(\Gamma_{1,h-1}), \ldots, bot(\Gamma_{k,h-1})$ are on the same $xy$ plane, so that $back(\Gamma_{1,h-1})$, $\ldots, back(\Gamma_{k,h-1})$ are on the same $xz$-plane and so that $right(\Gamma_{i,h-1})$ is one unit to the left of $left(\Gamma_{i+1,h-1}), \forall i$ such that $1 \leq i < k$. Place $r_h$ one unit behind and on the same $y$ line of $r_{1,h-1}$ (see Fig. 1 (b)); (iii) if $h \bmod 3 \equiv 1$, then place $\Gamma_{1,h-1}, \Gamma_{2,h-1}, \ldots, \Gamma_{k,h-1}$ so that $bot(\Gamma_{1,h-1}), \ldots, bot(\Gamma_{k,h-1})$ are on the same $xy$-plane, so that $left(\Gamma_{1,h-1}), \ldots, left(\Gamma_{k,h-1})$ are on the same $yz$-plane and so that $front(\Gamma_{i,h-1})$ is one unit behind $back(\Gamma_{i+1,h-1}), \forall i$ such that $1 \leq i < k$. Place $r_h$ one unit below and on the same $z$ line of $r_{1,h-1}$ (see Fig. 1 (c)). Finally, remove from $T_h$ the extra nodes and their incident edges to obtain a drawing $\Gamma$ of $T$. The algorithm we have just described is the main ingredient in the proof of the following theorem.

**Theorem 1.** *Given an $n$-nodes balanced tree $T$ with height $h$ and constant degree $k$, there exists an $O(n)$ time algorithm that constructs a $3D$ crossing free straight-line grid drawing $\Gamma$ of $G$ such that: the volume is $O(n)$, the aspect ratio is $O(1)$, $\Gamma$ satisfies the subtree separation property, $\Gamma$ satisfies the tip-over property, and $\Gamma$ is (strictly) upward in each of the three coordinate directions.*

**Proof (sketch):** We construct a straight-line drawing $\Gamma$ of $T$ by applying the algorithm described in this section. By inductive arguments it's easy to show that $\Gamma$ is crossing-free and satisfies the subtree separation property and the tip-over property. Further, by an easy inductive analysis, it is possible to prove that $\Gamma_h$ (and so $\Gamma$) is contained in a bounding box $B(\Gamma_h)$ of dimension $[O(\sqrt[3]{n}) \times O(\sqrt[3]{n}) \times O(\sqrt[3]{n})]$, $[O(\sqrt[3]{n/k}) \times O(\sqrt[3]{n/k}) \times O(\sqrt[3]{nk^2})]$, or $[O(\sqrt[3]{nk}) \times O(\sqrt[3]{n/k^2}) \times O(\sqrt[3]{nk})]$ if $h \bmod 3 \equiv 1$, if $h \bmod 3 \equiv 2$, or if $h \bmod 3 \equiv 0$, respectively. Since $k = O(1)$ the bounds on the volume and on the aspect ratio of $\Gamma$ follow. It's easy to see that $\Gamma$ is *upward* in each of the three coordinate directions. A slight modification of the algorithm permits also to produce *strictly upward* drawings: for this purpose, it is sufficient to translate, in the inductive construction of the algorithm, the drawings of the subtrees $T_{1,h-1}, T_{2,h-1}, \ldots,$ $T_{k,h-1}$ by vectors $(1, 0, 1)$, $(1, 1, 0)$ and $(0, 1, 1)$, for the case in which $h \bmod 3 \equiv 0$, $h \bmod 3 \equiv 1$ and $h \bmod 3 \equiv 2$, respectively. Such a modification doesn't alter the asymptotic bounds on the volume and on the aspect ratio of $\Gamma$. Finally, the algorithm can be easily implemented to run in linear time. $\square$

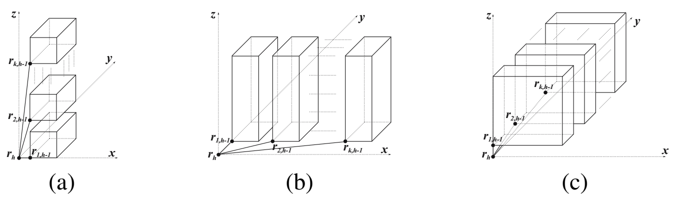
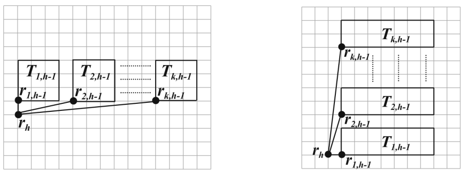


**Fig. 1.** Inductive construction of $\Gamma_h$: (a) $h \bmod 3 \equiv 2$. (b) $h \bmod 3 \equiv 0$. (c) $h \bmod 3 \equiv 1$.

## 4 Two-Dimensional Drawings of Constant Degree Balanced Trees

We now apply a variation of the algorithm in Section 3 to draw a balanced constant degree tree $T$ in two dimensions. First, add extra nodes to $T$ until it is complete. Again, this can be done without altering the height $h$ and the degree $k$ of $T$. Now we have to construct a drawing $\Gamma_h$ of a complete tree $T_h$. This can be done recursively as follows. If $h = 1$, then place $r_1$ in $(0, 0)$. If $h > 1$, suppose you have drawn $\Gamma_{1,h-1}, \Gamma_{2,h-1}, \ldots, \Gamma_{k,h-1}$. We distinguish two cases: (i) if $h$ is *even*, then place $\Gamma_{1,h-1}, \Gamma_{2,h-1}, \ldots, \Gamma_{k,h-1}$ so that $bot(\Gamma_{1,h-1}), \ldots, bot(\Gamma_{k,h-1})$ are on the same $x$-line and so that $left(\Gamma_{i+1,h-1})$ is one unit to the right of $right(\Gamma_{i,h-1}), \forall i$ such that $1 \leq i < k$. Place $r_h$ one unit below and on the same $y$-line of $r_{1,h-1}$ (see Fig. 2 (a)); (ii) if $h$ is *odd*, then place $\Gamma_{1,h-1}, \Gamma_{2,h-1}, \ldots, \Gamma_{k,h-1}$ so that $left(\Gamma_{1,h-1}), \ldots, left$ $(\Gamma_{k,h-1})$ are on the same $y$-line and so that $bot(\Gamma_{i+1,h-1})$ is one unit above $top(\Gamma_{i,h-1})$, $\forall i$ such that $1 \leq i < k$. Place $r_h$ one unit to the left and on the same $x$-line of $r_{1,h-1}$ (see Fig. 2 (b)). Finally, remove from $T_h$ the extra nodes and their incident edges to obtain a drawing $\Gamma$ of $T$. We have the following theorem:



(a) (b)

**Fig. 2.** Inductive construction of $\Gamma_h$: (a) $h$ even. (b) $h$ odd.

**Theorem 2.** *Given an $n$-nodes balanced tree $T$ with height $h$ and constant degree $k$, there exists an $O(n)$ time algorithm that constructs a $2D$ planar straight-line grid drawing $\Gamma$ of $T$ such that: the area is $O(n)$, the aspect ratio is $O(1)$, the angular resolution is $\Omega\,(1/\sqrt{n})$, $\Gamma$ satisfies the tip-over property, $\Gamma$ satisfies the subtree separation property, and $\Gamma$ is (strictly) upward in each of the two coordinate directions.*

**Proof (sketch):** We construct a straight-line drawing $\Gamma$ of $T$ by applying the algorithm described in this section. By inductive arguments it's easy to show that $\Gamma$ is planar and satisfies the subtree separation property and the tip-over property. Further, by an easy inductive analysis , it is possible to prove that $\Gamma_h$ (and so $\Gamma$) is contained in a bounding box $B(\Gamma_h)$ of dimension $[O(\sqrt{n}) \times O(\sqrt{n})]$, or $[O(\sqrt{nk}) \times O(\sqrt{n/k})]$, if $h$ is odd, or if $h$ is even, respectively. Since $k = O(1)$ the bounds on the area and on the aspect ratio of $\Gamma$ follow. It's easy to see that $\Gamma$ is *upward* in each of the three coordinate directions. A slight modification of the algorithm similar to that described in Section 3 permits also to produce *strictly upward* drawings without altering the asymptotic bounds on the area and on the aspect ratio of $\Gamma$. We now analyze the angular resolution of $\Gamma$. It is possible to show by induction that the angle between segments $\overline{r_{k-1,h-1}r_{1,h}}$ and $\overline{r_{k,h-1}r_{1,h}}$, say $\phi$, is the smallest angle in $\Gamma_h$. We call $l$ the length of the longest edge of $B(\Gamma_h)$. So $l$ is the number of grid points on the longest edge of $B(\Gamma_h)$ minus one, and so

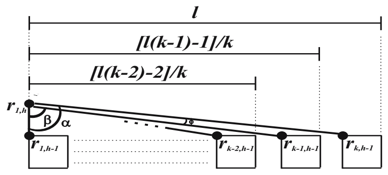


**Fig. 3.** Angular resolution of $\Gamma_h$

$l = O(\sqrt{nk})$. We now derive the value of $\sin(\phi)$ by applying the trigonometric formula $\sin(\phi) = \sin(\alpha)\cos(\beta) - \sin(\beta)\cos(\alpha)$ to the angles $\alpha, \beta$, and $\phi$ shown in Fig. 3 and by applying the Pythagorean Theorem to the two rectangular triangles between vertices $r_{1,h}, r_{1,h-1}, r_{k-1,h-1}$, and $r_{k,h-1}$:

$$\sin(\phi) = \frac{\left(\frac{k-1}{k}l - \frac{1}{k} + 1\right) - \left(\frac{k-2}{k}l - \frac{2}{k} + 1\right)}{\sqrt{(\frac{k-1}{k}l - \frac{1}{k} + 1)^2 + 1}\sqrt{(\frac{k-2}{k}l - \frac{2}{k} + 1)^2 + 1}} > \frac{\frac{l+1}{k}}{(l+1)^2 + 1} >$$
$$> \frac{l}{k(l^2 + 2l + 2)} = \Omega\left(\frac{1}{kl}\right) = \Omega\left(\frac{1}{k^{\frac{3}{2}}\sqrt{n}}\right) = \Omega\left(\frac{1}{\sqrt{n}}\right).$$

Finally, the algorithm can be easily implemented to run in linear time. □

The table below compares some asymptotic properties of the algorithm shown in this section with those of the algorithm of Trevisan ([8]).

| algorithm | area | aspect ratio | angular resolution | subtree separation |
|---|---|---|---|---|
| *Our Algorithm* | $O(n)$ | $O(1)$ | $\Omega\left(1/\sqrt{n}\right)$ | YES |
| *Algorithm* [8] | $O(n)$ | $O(1)$ | $O\left(1/n\right)$ | NO |

## 5 Three-Dimensional Poly-line Drawings of Bounded Degree Trees

This section is devoted to the proof of the following theorem:

**Theorem 3.** *Given a $n$-nodes tree $T$ with degree $k = O(n^\delta)$, where $\delta$ is a constant less than $\frac{1}{3}$, there exists a three-dimensional poly-line crossing-free drawing $\Gamma$ with $O(n)$ volume and $O(1)$ aspect ratio.*

The proof of the above theorem strongly exploits the techniques introduced in [6] by Garg et al. They showed that given two constants $\delta$ and $\alpha$, with $0 < \delta < \alpha < 1$, for every $n$-nodes tree $T$ with degree $k = O(n^\delta)$ it is possible to construct a two-dimensional upward planar poly-line grid drawing $\Gamma'$ with $O(n)$ area, height $H = O(n^{1-\alpha})$ and width $W = O(n^\alpha)$. This is done as follows: (1) $T$ is augmented with dummy nodes to an homeomorphic tree $T'$; (2) each node $v$ of $T'$ is associated with a layer $\gamma(v)$, so that for each edge $(u, v)$ of $T'$ $|\gamma(u) - \gamma(v)| \leq 1$; (3) it is constructed a planar straight-line drawing of $T'$ with the property that $y(v) = \gamma(v)$ for each vertex $v$; (4) each dummy node is replaced by a bend, obtaining the poly-line drawing $\Gamma'$ of $T$.

To obtain a three-dimensional drawing $\Gamma$ of $T$ with the properties claimed in Theorem 3, we suppose to apply the algorithm in [6]. Now we perform a "roll up" of $\Gamma'$, in a way very similar to that used in [2] to transform two-dimensional orthogonal drawings in three-dimensional drawings. This is done as follows. First, subdivide $\Gamma'$ in $O(H^{1/2})$ drawings $\Gamma'_0, \Gamma'_1, \ldots, \Gamma'_k$, so that $\Gamma'_i$ contains the part of $\Gamma'$ between layers $i \cdot \lfloor H^{1/2} \rfloor$ and $(i+1) \cdot \lfloor H^{1/2} \rfloor - 1$ (see Fig. 4 (b)). So the height of each $\Gamma'_i$ is $O(H^{1/2})$. Then we move each $\Gamma'_i$ to the plane $z = i$ and we reflect each $\Gamma'_i$ such that $i$ is odd with respect to $xy$-plane (see Fig. 4 (c)). More precisely, the transformation of $\Gamma'$ in $\Gamma$ consists in assigning the three coordinates to each vertex and to each bend so that: (1) the $x$-coordinate of each vertex (bend) $v$ of $T$ is equal to the $x$-coordinate of $v$ in $\Gamma'$; (2) denoting by $y^*(v)$ the $y$-coordinate of $v$ in $\Gamma'$, the $y$-coordinate of each vertex (bend) $v$ of $T$ that belongs to $\Gamma'_i$, with $i$ even (odd), is set equal to $y^*(v) - i \cdot \lfloor H^{1/2} \rfloor$ (resp. equal to $(i+1) \cdot \lfloor H^{1/2} \rfloor - y^*(v) - 1$); (3) the $z$-coordinate of each vertex (bend) $v$ of $T$ that belongs to $\Gamma'_i$ is equal to $i$. From [6], we know that by setting $\alpha$ to $1/3$, $\Gamma'$ has height $H = O(n^{2/3})$ and width $W = O(n^{1/3})$. Further, by our construction, the $y$-extension of $\Gamma$ is $H^{1/2} = O(n^{1/3})$ and the $z$-extension of $\Gamma$ is equal to the number of drawings $\Gamma'_i$, i.e. $O(n^{1/3})$. So the volume and aspect ratio bounds claimed in Theorem 3 follow. From the planarity of $\Gamma'$ and from the property that each segment of such drawing belongs to one layer or is between two consecutive layers it is easy to derive that $\Gamma$ is crossing-free.

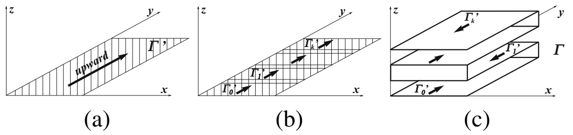


**Fig. 4.** (a) A planar poly-line upward grid drawing $\Gamma'$ of $T$. (b) Subdivision of $\Gamma'$ in partial drawings $\Gamma'_0, \Gamma'_1, \ldots, \Gamma'_k$,. (c) Roll up of $\Gamma'$ in a three-dimensional drawing $\Gamma$.

## References

1. T. M. Chan, M. T. Goodrich, S. Rao Kosaraju, and R. Tamassia. Optimizing area and aspect ratio in straight-line orthogonal tree drawings. *Comput. Geom.*, 23(2):153–162, 2002.
2. R. F. Cohen, P. Eades, T. Lin, and F. Ruskey. Three-dimensional graph drawing. *Algorithmica*, 17(2):199–208, 1997.
3. P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Comput. Geom.*, 2:187–200, 1992.
4. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.
5. S. Felsner, G. Liotta, and S. K. Wismath. Straight-line drawings on restricted integer grids in two and three dimensions. *J. Graph Algorithms Appl.*, 7(4):363–398, 2003.
6. A. Garg, M. T. Goodrich, and R. Tamassia. Planar upward tree drawings with optimal area. *Int. J. Comput. Geometry Appl.*, 6(3):333–356, 1996.
7. A. Garg and A. Rusu. Straight-line drawings of general trees with linear area and arbitrary aspect ratio. In *ICCSA (3)*, pages 876–885, 2003.
8. L. Trevisan. A note on minimum-area upward drawing of complete and Fibonacci trees. *Information Processing Letters*, 57(5):231–236, 1996.

# Simultaneous Graph Embedding with Bends and Circular Arcs*

Justin Cappos, Alejandro Estrella-Balderrama**, J. Joseph Fowler, and Stephen G. Kobourov

Department of Computer Science, University of Arizona
{justin,aestrell,jfowler,kobourov}@cs.arizona.edu

**Abstract.** We consider the problem of simultaneous embedding of planar graphs. We demonstrate how to simultaneously embed a path and an $n$-level planar graph and how to use radial embeddings for curvilinear simultaneous embeddings of a path and an outerplanar graph. We also show how to use star-shaped levels to find 2-bends per path edge simultaneous embeddings of a path and an outerplanar graph. All embedding algorithms run in $O(n)$ time.

## 1 Introduction

Embedding trees and other classes of planar graphs on predetermined point-sets, small integer grids, and levels is motivated by graph layout algorithms and applications in visualizing hierarchical information. Level and radial embeddings can also be used for simultaneous embedding of graphs which are in turn useful in dynamic graph visualization. Simultaneous embedding of planar graphs is also motivated by its relationship with problems of graph and geometric thickness.

A *geometric simultaneous embedding* of two vertex-labeled planar graphs on $n$ vertices in the $xy$-plane is possible if there exists a labeled point set of size $n$ such that each of the graphs can be realized on that point set (using the vertex-point mapping defined by the labels) with straight-line edges without crossings. For example, any two paths can be simultaneously embedded, while there exist pairs of outerplanar graphs that do not have a simultaneous embedding [3]. Geometric simultaneous embeddings are quite restrictive: pairs of trees and triples of paths may not have such embeddings. Less restrictive versions allow for larger classes of graphs to be embedded without crossings, using few bends per edge [7].

Suppose an $n$-vertex path $P$ is labeled 1 to $n$ from one endpoint to the other. In this paper, we show how to simultaneously embed $P$ with an $n$-vertex planar graph $G$ (also labeled from 1 to $n$) that remains planar when the $y$-coordinate of each vertex of $G$ equals its label. We can restrict each vertex of $G$ to lie on the distinct horizontal line, or *level*, $\ell_j = \{(x,j) \mid x \in \mathbb{R}\}$ given by its label $j \in \{1, 2, \ldots, n\}$. Such graphs are called *level planar graphs* with respect to the labeling of $G$. The ability to simultaneously embed $P$ and $G$ in this way depends

* This work is supported in part by NSF grants CCF-0545743 and ACR-0222920.
** Partially supported by CONACYT-México.

on the particular labeling of $G$. If $G$ is not level planar for the given labeling, then we give alternative simultaneous embedding techniques provided that $G$ is outerplanar: we retain the straight-line edges for $G$, but relax the edges of $P$ to be either composed of one circular arc each or to have 2 bends per path edge.

## 1.1 Related Work

Brass *et al.* [3] describe linear time algorithms for geometric simultaneous embeddings of pairs of paths, cycles, and caterpillars on an $O(n) \times O(n)$ grid. Geyer *et al.* [10] show that tree-tree pairs do not always have a geometric simultaneous embedding, and the status of tree-path pairs is open. If bends on the edges are allowed, Erten and Kobourov [7] show that tree-path pairs can be embedded simultaneously using at most one bend per tree edge and no bends of path edges on an $O(n) \times O(n^2)$ grid. Moreover, pairs of planar graphs can be embedded simultaneously using at most 3 bends per edge using an $O(n^2) \times O(n^2)$ grid.

A related problem is that of level planarity, where the goal is to display graphs according to a given hierarchical ordering of the vertices. Such graphs are called level planar graphs [5]. In particular, Jünger and Leipert [11] present a linear time planarity embedding algorithm for level planar graphs using PQ-trees, where the resulting embedding is a set of linear orderings of vertices on each level. Once a graph is determined level planar, Eades *et al.* [6] can produce a straight-line drawing in $O(|V|)$ time, though it may require exponential area. In [8], a characterization of trees that are level planar for any possible labeling of the vertices is given. These trees are called *unlabeled level planar* (ULP).

## 1.2 Our Contributions

We present results about simultaneous embeddings of pairs of graphs $(P, G)$ on $n$ vertices *without crossings*, where $P$ is always a path using curvilinear edges or piecewise-linear edges and $G$ is one of several different classes of planar graphs. Our results illustrate the following trade-offs between the class of graphs to which $G$ can belong and the type of edges used for $P$:

1. If $G$ is level planar, then we show how both $G$ and $P$ can be simultaneously embedded with straight-line edges in $O(n)$ time.
2. If $G$ is outerplanar, we show how to find a plane drawing for $G$ simultaneously with a drawing for $P$ that uses one circular arc per edge in $O(n)$ time.
3. If $G$ is outerplanar and piecewise-linear edges are desirable, we show how to obtain a plane drawing for $G$ simultaneously with a drawing for $P$ that uses two bends per edge in $O(n)$ time.

Table 1 summarizes our current results regarding simultaneous embedding of pairs of planar graphs and relevant previous results. Full details are given in [4].

**Table 1.** Summary of results. ULP here stands for unlabeled level planar, as defined above.

| Pair ($P$, $G$) | Edges in $P$ | Edges in $G$ | Condition | Reference |
|---|---|---|---|---|
| (Path, Path) | 0 bends | 0 bends | none | [3] |
| (Path, Tree) | 0 bends | 1 bend | none | [7] |
| (Path, Planar) | 0 bends | 0 bends | $G$ is ULP | Theorem 1 |
| (Path, Planar) | 0 bends | 0 bends | $G$ level planar w.r.t. $P$ | Theorem 2 |
| (Path, Outerplanar) | circular arcs | 0 bends | none | Theorem 3 |
| (Path, Outerplanar) | 2 bends | 0 bends | none | Theorem 8 |
| (Path, Tree) | 0 bends | 0 bends | none | Open |

## 2 Geometric Simultaneous Embedding

We try to use standard notation when discussing level graphs while focusing on the aspects of level graphs that give us simultaneous embeddings using straight-line segments. In doing so, we omit from our definitions certain properties of level graphs that are not directly relevant to our problem domain.

Let $G(V,E)$ be an $n$-vertex undirected graph with a labeling $\mathcal{L} : V \xrightarrow[\text{onto}]{1:1} \{1, 2, \ldots, n\}$, which induces a *level graph* $G(V,E,\phi)$ with a bijective *level assignment* $\phi = \mathcal{L}$ onto $n$ levels. A vertex $v$ in $V$ is a $j$-level vertex if $\phi(v) = j$. A *level drawing* is *level planar* if it admits a plane drawing such that each $j$-level vertex $v$ can be embedded onto the horizontal line $\ell_j = \big\{(x,j)\,|\,x \in \mathbb{R}\big\}$. A level graph is level planar if it has a level drawing.

While any planar graph $G$ admits some labeling for which it is level planar, only some planar graphs are level planar regardless of the labeling used. Such graphs are called *unlabeled level planar* (ULP) [8] and they can be characterized in terms of a pair of forbidden subtrees; see Fig. 1. Moreover, the linear-time recognition and embedding algorithms for ULP trees yield a straightforward way to simultaneously embed an $n$-vertex path and an $n$-vertex ULP tree as illustrated in the following theorem.

**Theorem 1.** *A geometric simultaneous embedding of an $n$-vertex graph $G$ and an $n$-vertex path $P$ can be computed in $O(n)$ time, provided $G$ is ULP.*

*Proof.* Label the vertices of $P$ sequentially 1 to $n$, and label the vertices of $G$ so that $\mathcal{L} = \phi$ for any $n$-level bijective assignment $\phi$ of the vertices in $G$ and $P$. If $G$ happens to be a ULP tree, it is either a caterpillar, radius-2 star, or degree-3 spider, each of which has an $O(n)$ time algorithm [8] to produce a compact straight-line level planar drawing of $G$. If $G$ is not a tree, the $O(n)$ time algorithm of Eades *et al.* [6] can provide a planar straight-line drawing of $G$ with level assignment $\phi = \mathcal{L}$. Regardless, the $y$-coordinate of each $j$-vertex of $G$ matches its label $j$ in a level drawing of $G$. Then we draw the path $P$ in a $y$-monotone fashion zig-zagging upward from one level to the next in $O(n)$ time. This completes our geometric $O(n)$ time simultaneous embedding of $P$ and $G$ since no path edges of $P$ can cross given its $y$-monotone nature. □

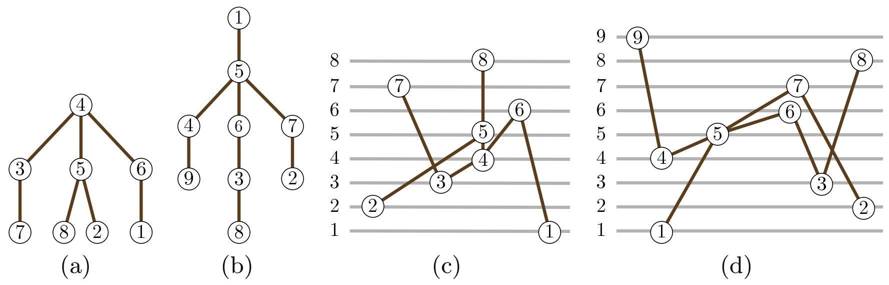


**Fig. 1.** Two forbidden trees $T_1$ in (a) and $T_2$ in (b) that fully characterize ULP trees, and their embeddings with crossings in (c) and (d), respectively, for the given labelings.

The requirement for $G$ to be ULP is overly restrictive. One can use the same approach to simultaneously embed a planar graph $G$ and a path $P$, provided that $G$ is level planar with respect to the labeling induced by $P$. This gives the following theorem.

**Theorem 2.** *A geometric simultaneous embedding of an $n$-vertex graph $G$ and an $n$-vertex path $P$ can be computed in $O(n)$ time, provided $G$ is level planar with respect to the labeling given by $P$.*

The disadvantage of this approach to simultaneously embed a path and a planar graph is that most planar graphs, including most trees, are not level planar for some labeling. This is not surprising since it is a strong restriction to have a predetermined order of the $y$-coordinates of the vertices. What is surprising, however, is that introducing curvature to the levels, by using circles in lieu of horizontal lines, is enough to allow us to embed all trees and outerplanar graphs with circular arcs for path edges. We show this in the next section.

## 3 Simultaneous Embedding with Curves

This section combines straight-line embeddings of outerplanar graphs with paths consisting of circular arcs to produce curvilinear simultaneous embeddings. First, we describe how to obtain a plane drawing of an $n$-vertex outerplanar graph $G$ on a set of concentric circles such that each vertex lies on a distinct circle, determined by the labeling of $G$. We then use this straight-line crossings-free drawing of $G$ to simultaneously embed $G$ with an $n$-vertex path $P$, such that each path edge consists of a circular arc that lies between adjacent concentric circles. This will give the primary theorem of this section.

**Theorem 3.** *An $n$-vertex outerplanar graph $G$ can be simultaneously embedded with an $n$-vertex path $P$ in $O(n)$ time such that $G$ forms a plane drawing and $P$ is drawn without crossings using one circular arc per edge.*

### 3.1 Embedding an Outerplanar Graph on Concentric Circles

In this section, we describe how to embed a radial plane drawing of an labeled $n$-vertex outerplanar graph $G$ on a set of distinct $n$ concentric circles using the labeling of $G$. This is similar to a radial level planar embedding on $n$ radial levels, i.e., circles, except that we are using straight lines instead of radially monotone polylines for edges. However, as straight-line edges are not necessarily radially monotone, the radial level planarity test and embeddings algorithms of Bachmaier *et al.* [1] cannot be directly applied here. Rather, we use the following result from Bose [2].

**Theorem 4.** *(**Theorem 6.2** of [2]) If the input point set $\mathcal{P}$ is in convex position then $O(n)$ time and space is sufficient to straight-line embed $G$ into $\mathcal{P}$.*

Using this theorem we can obtain our linear time radial straight-line embedding of $G$ onto $n$ concentric circles given by the next lemma.

**Lemma 5.** *Given a set $\mathcal{C}$ of $n$ concentric circles $\{C_1, C_2, \ldots, C_n\}$ centered at the origin $o$ with monotonically increasing radii $r_1, r_2, \ldots, r_n$, it is a sufficient condition that $\frac{r_1}{r_n} > \cos\frac{2\pi}{n}$ in order to obtain in $O(n)$ time a radial plane drawing of an $n$-vertex outerplanar graph $G$ with vertices labeled $1$ to $n$ such that each vertex with label $j$ is embedded on circle $C_j$.*

*Proof Sketch:* Our strategy is to first embed $G$ using straight-line edges onto a circle $C$ with radius $r$ centered at the origin $o$ without crossings in $O(n)$ time via Theorem 4. We want the vertices of $G$ evenly distributed along the circle, i.e., on a point set $\mathcal{P}$ such that there is a point $p_i$ in $\mathcal{P}$ at a distance $r$ from the origin $o$, i.e., $|\overline{op_i}| = r$, at every radian angle $\theta_k = (k-1)\frac{2\pi}{n}$, where $\theta_k$ is the angle $\angle p_1 o p_k$ for $k = 1, 2, \ldots, n$. Clearly, $\mathcal{P}$ forms a convex set; see Fig. 2(b).

Then we perturb the vertices in $O(n)$ time in a radial direction so that each one lies on its own circle according to the labeling of $G$, i.e., a vertex $v$ labeled $j$ is placed on $C_j$ where $C_n = C$. Finally, we determine that if our perturbation is sufficiently small, i.e., $\frac{r_1}{r_n} > \cos\frac{2\pi}{n}$, then the radial drawing remains free of crossings. We do this by perturbing the point set $\mathcal{P}$ to match the new locations of the vertices of $G$. We call this perturbed point set $\mathcal{P}'$; see Fig. 2(c).

We note that while Theorem 4 works for any convex point set $\mathcal{P}$, which vertex is embedded at which point of $\mathcal{P}$ is determined by the algorithm. We can show that when rerunning the algorithm on point set $\mathcal{P}'$ instead of $\mathcal{P}$, it makes the same choices. In order to do that it suffices to show that when we perturb the point set $\mathcal{P}$ to $\mathcal{P}'$ the following two conditions hold:

– all the points of $\mathcal{P}'$ remain in convex position *and*
– the order in which each point $p'$ of $\mathcal{P}'$ sees all the other points $\mathcal{P}' - p'$ using a radial line sweep centered at $p'$ remains the same.

Since the points $\mathcal{P}$ are uniformly distributed over all radial angles, retaining the convexity of $\mathcal{P}'$ also achieves the condition of retaining relative positioning provided that the points of $\mathcal{P}'$ are only perturbed in a radial direction. Next, we

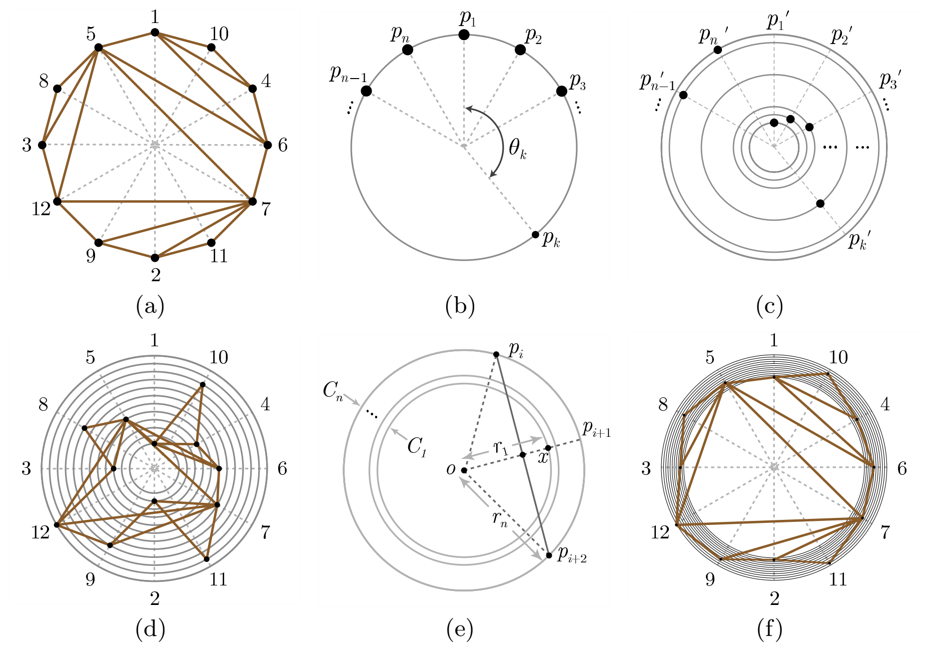


**Fig. 2.** The vertices of a 12-vertex outerplanar graph $G$ are embedded on a circle in (a). This embedding follows the general point set $\mathcal{P}$ given in (b). The points are then perturbed radially inward so that each vertex with label $j$ lies on $C_j$ for $j = 1, 2, \ldots, 12$, where $C_{12}$ is the outermost circle, yielding point set $\mathcal{P}'$ in (c). Drawing $G$ on $\mathcal{P}'$ gives (d). Crossings are avoided by restricting the ratio between the radii of $C_1$ to $C_n$ in (e) giving (f).

show that $\frac{r_1}{r_n} > \cos\frac{2\pi}{n}$ is a sufficient condition in maintaining the convexity and same convex hull of $\mathcal{P}$ when perturbing the points to $\mathcal{P}'$.

Perturbing the point $p_{i+1}$ of $\mathcal{P}$ to lie on the innermost circle $C_1$, while letting its neighboring points $p_i$ and $p_{i+2}$ along the convex hull of $\mathcal{P}$, remain on $C = C_n$, gives a worst case in terms of affecting the convexity of $\mathcal{P}$. Fig. 2(e) illustrates this. Let $x$ denote the midpoint of $\overline{p_i p_{i+2}}$. In order for $p_{i+1}$ to remain on the convex hull of $\mathcal{P}$, it is sufficient that the distance from $o$ to $x$ is less than $r_1$, the radius of the innermost circle $C_1$. Since the angle $\angle xop_i$ is $\frac{2\pi}{n}$, if the ratio of $\frac{r_1}{r_n} > \frac{|\overline{ox}|}{r_n} = cos\frac{2\pi}{n}$, then $p_{i+1}$ will lie in the outer half-plane formed by the line passing through $p_i$ and $p_{i+2}$, i.e., the half-plane not containing the origin $o$. □

## 3.2 Embedding a Path of Circular Arcs Between Concentric Circles

From Lemma 5, we have that given $n$ distinct concentric circles $C_1, C_2, \ldots, C_n$ of monotonically increasing radii $r_1, r_2, \ldots, r_n$, we can create a plane drawing of any $n$-vertex outerplanar graph $G(V, E)$ with labeling $\mathcal{L} : V \xrightarrow[\text{onto}]{1:1} \{1, 2, \ldots, n\}$

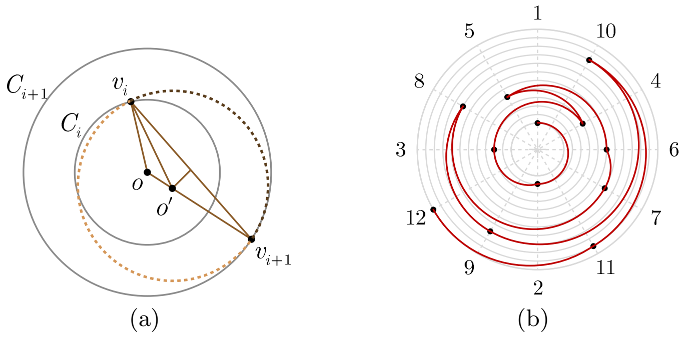


**Fig. 3.** Routing one circular arc per edge so that fits inside two consecutive concentric circles is in (a). The concentric circles are centered at $o$ where $o'$ is the center of a circle that gives a curve connecting $v_i$ and $v_{i+1}$ that stays within the annulus defined by $C_i$ and $C_{i+1}$. An example of this is given in (b) for the outerplanar graph from Fig. 2(a).

such that $v \in C_{\mathcal{L}(v)}$ for all $v \in V$ provided $\frac{r_1}{r_n} > \cos\frac{2\pi}{n}$. Here we assume the $n$-vertex path $P$ is labeled sequentially 1 to $n$. We show how to route the edges of the path using exactly one circular arc per path edge so that the arcs of $P$ form a radially monotone polyline, which implies that no two circular arcs intersect.

**Lemma 6.** *A radially monotonically increasing crossings-free drawing of an $n$-vertex path $P(V, E)$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\}$ can be realized on $n$ concentric circles $C_1, C_2, \ldots, C_n$, where $v_i \in C_i$ for $1, 2, \ldots, n$ with one circular arc per edge.*

*Proof.* It suffices to show that one circular arc always can be used to connect two consecutive vertices on the path, $v_i$ and $v_{i+1}$, such that the arc lies strictly outside circle $C_i$ and inside circle $C_{i+1}$ except for the end points of the arc at $v_i$ and $v_{i+1}$; see Fig. 3(a). Let $o$ be center of the circles. We compute $o'$, the center of the circle that forms the desired circular arc connecting $v_i$ and $v_{i+1}$ as follows. The center $o'$ is the intersection of the perpendicular bisector of $\overline{v_i v_{i+1}}$ and the line segment $\overline{o v_{i+1}}$. The radius of the circle centered at $o'$ is given by the distance from $o'$ to $v_i$. The shorter circular arc between $v_i$ and $v_{i+1}$ connects the two vertices and is located in the annulus between circles $C_i$ and $C_{i+1}$. Furthermore, the distance from $c$ to any point along the arc from $v_i$ to $v_{i+1}$ is monotonically increasing. Therefore, the entire path $P$ can be realized as a radially monotone polyline, implying no edge crossings, using one circular arc per edge; see Fig. 3(b). $\square$

Lemma 5 together with Lemma 6 gives us Theorem 3. However, as noted, this only works when we can restrict the radii of the concentric circles to be in a small range. One might wonder whether it is possible to use radially uniform concentric circles instead. The next subsection shows this is not the case.

### 3.3 Trees on Radially Uniform Concentric Circles

In this section we give an example of a 406-vertex tree with a labeling from 1 to 406 that cannot be straight-line embedded on a set of 406 radially uniform concentric circles such that each vertex lies on its respective circle.

**Lemma 7.** *There exists an 406-vertex tree $T(V,E)$ with labeling $\mathcal{L} : V \xrightarrow[onto]{1:1} \{1,2,\ldots 406\}$ that cannot be straight-line embedded on a set $\mathcal{C}$ of 406 radially uniform concentric circles $\{C_1, C_2, \ldots, C_n\}$ centered at $o$ with radii $r_1, r_2, \ldots, r_{406}$ such that $r_i = (i-1)\Delta$ for $i = 1,2,\ldots,406$ for any $\Delta > 0$, where each vertex with label $j$ is embedded on circle $C_j$.*

*Proof.* Here we use the ULP forbidden tree $T_1$ with 8 vertices from Fig. 1(a) to construct a 406 vertex tree $T$ with root $x$ which has 45 subtrees of 9 vertices each; see Fig. 4(a). Each of the 45 neighbors of $x$ is a degree-2 vertex connected to a copy of $T_1$. We start by labeling $x$ with 1 placing it on $C_1$, which has radius 0, so $x$ must be embedded at the center $o$.

Then we label each of its 45 neighbors with $362, 363, \ldots, 406$ so that at least one subtree, which is a copy of $T_1$, call it $T_1'$, must lie within the radian angle $\frac{2\pi}{45}$. W.l.o.g we assume that this sector is centered along a vertical line passing through the center $o$ since we can rotate the drawing of $T$ as needed.

Within this narrow sector, we observe that the tangents to the circles do no intersect any other circle; see Fig. 4(b). This is because the radius of $r_{i-1}$ is strictly less than $r_i \cos\frac{2\pi}{2\cdot 45} = r_i \cos\frac{\pi}{45}$ for $i = 1,2,\ldots,405$. In particular, $r_{405} = 404 < r_{406}\cos\frac{\pi}{45} = 405 \cdot 0.997564 = 404.013$.

Then we label the $k^{\text{th}}$ copy of $T_1$ with the labels from Fig. 1(c) adding the value of $(k-1)8+1$ to the labels for $k = 1,2,\ldots,45$. This preserves the $y$-ordering of the labels such that $T_1'$ (the copy of $T_1$ lying strictly within the radian angle

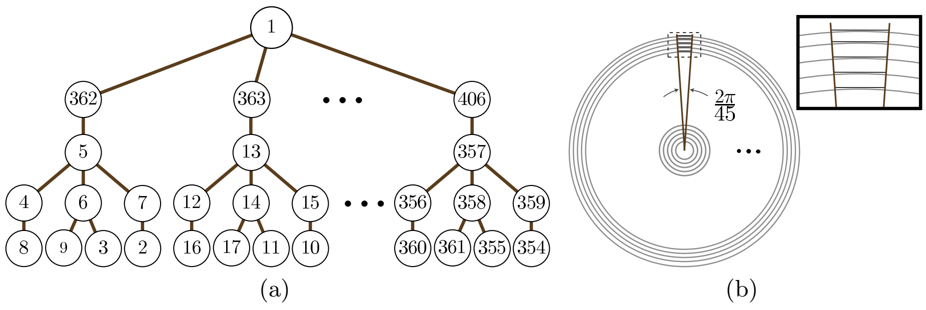


**Fig. 4.** The 406-vertex tree in (a) cannot be drawn on radially uniform concentric circles since there must exist one subtree that is a copy of $T_1$ from Fig. 1(a) that fully resides in a sector such that tangents of circles do not intersect any other circle. We can rotate this sector so that it lies directly above $o$ so that any vertices placed on the concentric circular arc in this sector must have strictly increasing $y$-coordinates as shown in (b).

$\frac{2\pi}{45}$ sector) must have strictly increasing $y$-coordinates. Hence, if $T_1'$ could be embedded on its circles, then it could be level planarly embedded, which is not the case for the given labeling of $T_1'$. The inability to level planarly embed $T_1'$ forbids a straight-line embedding of $T$. □

## 4 Simultaneous Embedding with Bends

This section combines straight-line embeddings of outerplanar graphs with paths whose edges are drawn with bends. We introduce the notion of *star-shaped levels* which allows us to obtain the main result of this section:

**Theorem 8.** *An $n$-vertex outerplanar graph $G$ can be simultaneously embedded with an $n$-vertex path $P$ in $O(n)$ time such that $G$ forms a plane drawing and $P$ is drawn without crossings with at most 2 bends per path edge.*

### 4.1 Projecting a 3D Outerplanar Graph onto the 2D $xy$-Plane

We start by using the following theorem to get a 3-dimensional embedding of an outerplanar graph onto a 3-side prism.

**Theorem 9.** *(Felsner, Liotta and Wismath [9]) Every outerplanar graph $G(V, E)$ with $n$ vertices admits a crossings-free straight-line grid drawing in three dimensions in optimal $O(n)$ volume that can be computed in $O(n)$ time and with the vertices of $G$ drawn on the grid points of a prism.*

The ability to do this projection can be used to give our next lemma.

**Lemma 10.** *There exists a projection of a 3-dimensional outerplanar graph $G$ on a 3-sided regular prism onto the $xy$-plane that preserves the number of straight-line edge crossings of $G$.*

*Proof Sketch:* The embedding of [9] uses the shortest-path distance from some arbitrary root vertex $r$ to every other vertex in the outerplanar graph $G$. Let $d_{\text{mod } 3}(v)$ denote the shortest-path distance from $r$ modulo 3. Fig. 5 gives an example of an outerplanar graph to be used in illustrating the embedding on star-shaped levels. The 3-dimensional regular prism used for this embedding can be visualized as standing vertically on a triangular base in the $xy$-plane in which the vertical edges are numbered 0, 1, and 2 in clockwise order looking down from the positive $z$-direction. Then $G$ is "wrapped" around the prism such that each vertex $v$ is embedded along the edge $d_{\text{mod } 3}(v)$ of the prism; see Fig. 6.

We will use the prism to construct the star-shaped levels. Assume that the base of the prism is an equilateral triangle with side length $\ell$ and that the height of the prism is $3\ell$. Additionally, let $r = \ell/2\sqrt{3}$ be the radius of the circle that is inscribed within the triangle; see Fig. 6(b). We need to shift all the vertices so that they all lie along a fairly narrow band above the distance $2r$ from the base. This is so that when we "unfold" each vertical side of the prism by laying it flat on the $xy$-plane we do not introduce a crossing; compare Fig. 6(a) to (b).

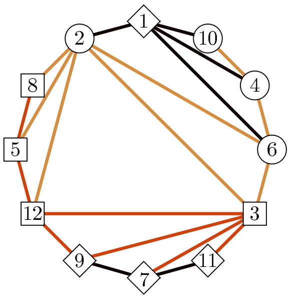


**Fig. 5.** Outerplanar graph $G$ used as an example for embedding on star-shaped levels. The vertices in this figure and Fig. 6 are in three different shapes (diamonds, circles, and squares) according to whether the vertex is at a shortest-path distance of 0, 1, or 2 modulo 3 from vertex 1, respectively. The edges are colored one of three colors in a similar fashion.

We pick the side of the rectangular face that is in a 90° counter-clockwise direction from the prism's base onto which to map the vertices along the prism edge. The positions of the vertices are then mapped directly to their corresponding positions along this projected edge in the $xy$-plane so that the edges on a prism face map directly to straight-line segments in the $xy$-plane.

There is the possibility that extra crossings will be introduced between edges incident to the same pair of prism edges in which one or both of the endpoints

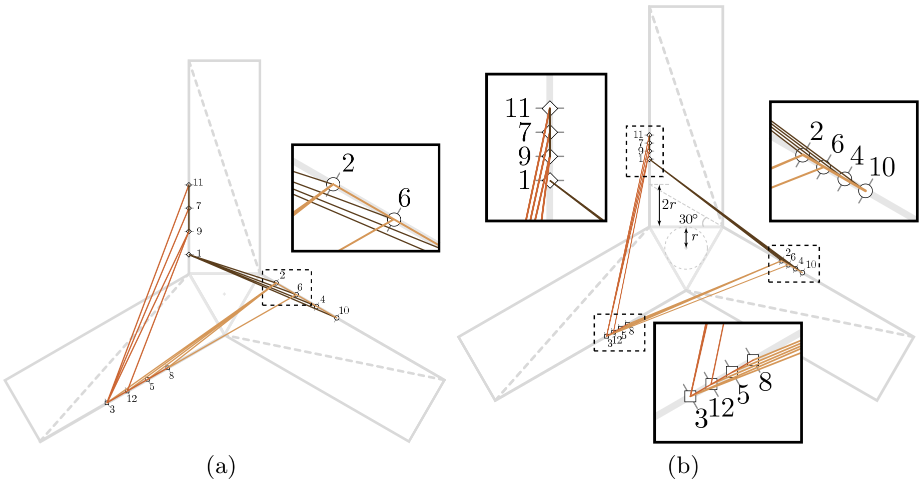


**Fig. 6.** Outerplanar graph $G$ from Fig. 5 projected onto the $xy$-plane from its 3-dimensional embedding. In (a) there are crossings since the vertices are not above the $2r$ threshold, where $r$ is radius of the circumscribed circle of the base of the prism. In (b) there are no crossings since all the vertices achieve this threshold.

lie within a distance of $2r$ from this base. Once all the endpoints lie above this threshold, which is the point at which an *extension* of an adjacent prism edge at an angle of 120° would intersect the edge in the $xy$-plane, then no crossings can occur. This is illustrated by projecting the outerplanar graph $G$ from Fig. 5 onto the $xy$-plane in which all the vertices are not above this threshold in Fig. 6(a), but achieve this $2r$ threshold in Fig. 6(b) eliminating all crossings. □

### 4.2 Simultaneous Embedding Using Star-Shaped Levels

In this section we show how to use the star-shaped levels generated by the outerplanar graph $G$ from the previous section to simultaneously embed a path $P$ with exactly 2 bends per edge giving our next lemma.

**Lemma 11.** *There exists a 2-bend per path edge crossings-free drawing of a path $P$ using star-shaped levels for any vertex labeling.*

*Proof Sketch:* Let $n_i$ be the number of vertices along the $i^{\text{th}}$ prism edge (for $i \in \{0, 1, 2\}$, the shortest-path distance of the vertices from $r$ modulo 3) where $n_{\max} = \max\{n_0, n_1, n_2\}$ is then the total number of star-shaped levels required for this simultaneous embedding. Next we need to perturb the $n_i$ vertices that lie along edge $i$ (after performing the above projection to the $xy$-plane) to lie along one of $n_{\max}$ closely adjacent nested star-shaped levels. How close these levels need to be is given by the subsequent lemma. Each star-shaped level has 6 sides and is a scaled-down version of the outermost level (w.r.t. the center of the circle circumscribed within the triangular prism base). This includes the 3 *prism edges* projected onto the $xy$-plane, and the 3 edges that each connect the top of one projected prism edge to the bottom of the next in a clockwise direction, termed *connecting edges*; see the dashed 3-pointed star in Fig. 7.

When perturbing the vertices, we are careful to move a vertex in a direction perpendicular to the prism edge (as well as all of the adjacent edges of the nested levels) on which it resided. The problem with perturbing the vertices too much is the introduction of crossings in $G$. For example, the outerplanar graph $G$ in Fig. 5 is shown in Fig. 7 on a set of star-shaped levels in which the levels are not spaced sufficiently close enough, resulting in several crossings.

The vertices are placed clockwise from outermost to innermost star-shaped level by the order given by the path labeling. However, in order to be able to route the path back and forth between vertices that alternate back and forth between adjacent sides of the prism, an extra connecting edge needs to be inserted to lie half-way between every pair of adjacent connecting edges. Since we need one for *each* level, an extra connecting edge also needs to be added to lie just interior to the ${n_{\max}}^{\text{th}}$ innermost connecting edge. In Fig. 7 these are shown as dashed line segments. We denote the levels along which the vertices lie as *regular levels*, which are depicted in Fig. 7 with solid gray edges, and the in-between levels as *half-levels* consisting of dashed edges.

The rule then for going from one vertex $u$ to the next vertex $v$ is that if going clockwise, then $v$'s connecting regular level's edge is used, otherwise, its

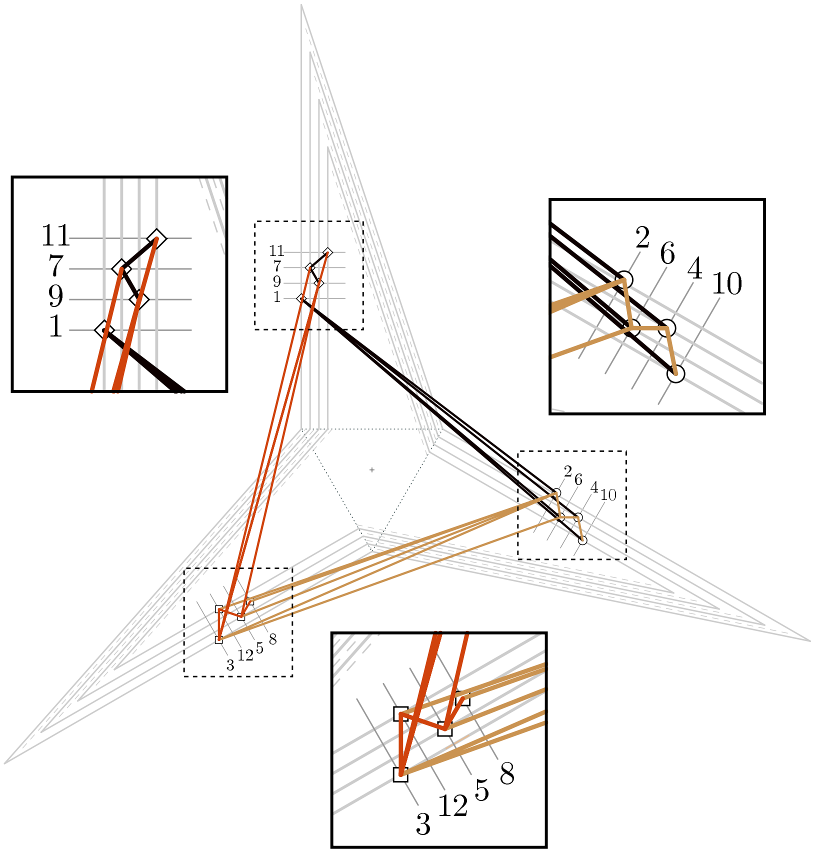


**Fig. 7.** Outerplanar graph $G$ from Fig. 5 embedded onto the star-shaped levels

connecting half-level's edge is used. Hence, if going clockwise, 2 bends are introduced at each endpoint of the connecting regular level's edge corresponding to the *destination* vertex. Otherwise, 2 bends are introduced at each endpoint of the connecting half-level's edge of the destination vertex. □

As was the case with circular arcs, making the star-shaped levels sufficiently close for a given outerplanar graph $G$, as well as an additional technical condition of placing the vertices of $G$ in close proximity along the middle portion of the prism edge, avoids crossings as given by our final lemma. The proof of this lemma (along with a detailed description of the geometry of the star-shaped levels) is included in the technical report [4].

**Lemma 12.** *Let $G$ be an $n$-vertex outerplanar graph. Let $\delta$ be the maximum separation between two vertices of $G$ along the same prism edge* before *perturbing vertices onto the other star-shaped levels, and let $\Delta$ be the maximum separation between the nested star-shaped levels. Then $G$ will be crossings-free when embedded onto the star-shaped levels provided $\Delta < \frac{\ell}{10n}$ and $\delta < \frac{\ell}{(n-1)^2}$.*

## 5 Conclusions and Open Problems

We presented results in simultaneous embeddings of path and outerplanar graphs with circular arc edges or a small number of bends. Other open problems include:

1. Do all tree-path pairs have geometric simultaneous embedding?
2. What is the complexity of determining whether two planar graphs admit a geometric simultaneous embedding?
3. What is the complexity of determining whether a pair of graphs can be simultaneously embedded?

## Acknowledgments

We would like to thank Jiří Fiala, Ferran Hurtado, and David Wood for the stimulating discussions about this problem.

## References

1. C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time. *J. Graph Algorithms Appl.*, 9(1):53–97, 2005.
2. P. Bose. On embedding an outer-planar graph in a point set. *CGTA: Computational Geometry: Theory and Applications*, 23(3):303–312, 2002.
3. P. Brass, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. B. Mitchell. On simultaneous graph embedding. In *8th Workshop on Algorithms and Data Structures*, pages 243–255, 2003.
4. J. Cappos, A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. Simultaneous graph embedding with bends and circular arcs. Technical Report TR06-02, University of Arizona, 2006.
5. G. Di Battista and E. Nardelli. Hierarchies and planarity theory. *IEEE Trans. Systems Man Cybernet.*, 18(6):1035–1046 (1989), 1988.
6. P. Eades, Q. Feng, X. Lin, and H. Nagamochi. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. *Algorithmica*, 44(1):1–32, 2006.
7. C. Erten and S. G. Kobourov. Simultaneous embedding of planar graphs with few bends. In *12th Symposium on Graph Drawing (GD)*, pages 195–205, 2004.
8. A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. Characterization of unlabeled level planar trees. Manuscript accepted by 14th Symposium on Graph Drawing, 2006.
9. S. Felsner, G. Liotta, and S. Wismath. Straight-line drawings on restricted integer grids in two and three dimensions. *Journal of Graph Algorithms and Applications*, 7(4):363–398, 2003.
10. M. Geyer, M. Kaufmann, and I. Vrto. Two trees which are self-intersecting when drawn simultaneously. In *13th Symposium on Graph Drawing*, pages 201–210, 2005.
11. M. Jünger and S. Leipert. Level planar embedding in linear time. *J. Graph Algorithms Appl.*, 6(1):67–113, 2002.

# Embedding Graphs Simultaneously with Fixed Edges*

Fabrizio Frati

Dipartimento di Informatica e Automazione – Università di Roma Tre
frati@dia.uniroma3.it

**Abstract.** We show that a planar graph and a tree can always be simultaneously embedded with fixed edges and that two outerplanar graphs generally cannot.

## 1 Introduction

A *simultaneous embedding* of two planar graphs $G_1$ and $G_2$ is a pair of drawings of $G_1$ and $G_2$ such that each drawing is planar and each vertex common to $G_1$ and $G_2$ is represented by the same point in both drawings. Unfortunately, if one wishes to visualize the edges of $G_1$ and $G_2$ as rectilinear segments (the so called *geometric simultaneous embedding*), not all pairs of graphs can be embedded simultaneously. Erten and Kobourov ([4]), Brass et al. ([1]), and Geyer et al. ([7]) have shown that it is not always possible to embed simultaneously with straight-line edges a planar graph and a path, three paths, and two trees, respectively. On the other hand, if one permits that each edge of a graph is displayed as a different Jordan curve (the so called *simultaneous embedding*), then by the results of Pach and Wenger ([9]) any number of planar graphs can be embedded simultaneously. Restricting the last constraints, one could permit that each edge is represented by a Jordan curve, but could force edges common to more graphs to have the same representation in the drawing of each graph (the so called *simultaneous embedding with fixed edges*). Di Giacomo and Liotta ([3]) showed that an outerplanar graph and a cycle can always be simultaneously embedded with fixed edges, improving the results in [4], where it is shown how to embed simultaneously with fixed ("consistent") edges a tree and a path. In [4] and [3] the problem of finding simultaneous embeddings with fixed edges of pairs of trees and of pairs of planar graphs is explicitly mentioned.

In this paper we improve the results on simultaneous embedding with fixed edges of graphs, by showing that there exist two outerplanar graphs that cannot be simultaneously embedded with fixed edges (Section 3) and that a planar graph and a tree can always be simultaneously embedded with fixed edges (Section 4). Then in Section 5 we give conclusions and suggest some open problems.

## 2 Preliminaries

We assume familiarity with graphs and their drawings (see e.g. [2]).

A *drawing* of a graph is a mapping of each vertex to a distinct point in the plane and of each edge to a Jordan curve between the endpoints of the edge. A *planar drawing* is

* Work partially supported by EC - Fet Project DELIS - Contract no 001907, by "Project ALGO-NEXT: Algorithms for the Next Generation Internet and Web: Methodologies, Design, and Experiments", and by MIUR Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale.

such that no two edges intersect. A *planar graph* is a graph that admits a planar drawing. An *embedding* of a graph $G$ is a circular ordering of the edges incident on each vertex of $G$. An embedding of a graph specifies what are its faces in any drawing respecting such embedding and so it specifies the *dual graph* of $G$ that is the graph with one vertex for each face of $G$ and with one edge between two vertices if the corresponding faces share an edge in $G$. A *poly-line drawing* is such that the edges are sequences of rectilinear segments. A *straight-line drawing* is such that all edges are rectilinear segments. It has been shown in [5] that every planar graph admits a planar straight-line drawing.

A *simultaneous embedding with fixed edges* of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with a bijective mapping $\gamma : V_1 \rightarrow V_2$ between their vertices is a pair of drawings $\Gamma_1$ and $\Gamma_2$ of $G_1$ and of $G_2$, respectively, such that: (i) each of $\Gamma_1$ and $\Gamma_2$ is a planar drawing, (ii) each vertex $v_2 = \gamma(v_1)$, with $v_1 \in V_1$ and $v_2 \in V_2$, is mapped in $\Gamma_2$ to the same point where $v_1$ is mapped in $\Gamma_1$, and (iii) an edge belonging to both $E_1$ and $E_2$ is represented by the same simple Jordan curve in $\Gamma_1$ and $\Gamma_2$. In this paper we only deal with simultaneous embedding with fixed edges, so in the following, unless otherwise specified, *simultaneous embedding* will always stand for *simultaneous embedding with fixed edges*.

## 3 Simultaneous Embeddings of Outerplanar Graphs

In this section we show that there exist two outerplanar graphs that cannot be simultaneously embedded. This result was already obtained in [1] for geometric simultaneous embedding. Although we believe that the pair of outerplanar graphs presented in [1] can not be simultaneously embedded even in the *fixed edges* setting, this was never pointed up and also the proof in [1] exploits the fact that the edges are drawn as segments.

**Theorem 1.** *There exist two outerplanar graphs that can not be simultaneously embedded with fixed edges.*

To prove Theorem 1 we first show the topologies of two outerplanar graphs $G_1$ and $G_2$ and a bijective mapping $\gamma$ between their vertices. We use the same name $v_i$ for a vertex $u$ of $G_1$ and a vertex $v$ of $G_2$ to mean that $v = \gamma(u)$.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two outerplanar graphs with six vertices each and with $E_1 = \{(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_1, v_4), (v_2, v_4), (v_2, v_5), (v_4, v_5), (v_2, v_6), (v_3, v_6)\}$, and $E_2 = \{(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_1, v_6), (v_2, v_6), (v_2, v_5), (v_6, v_5), (v_2, v_4), (v_3, v_4)\}$ (see Fig. 1). To show that $G_1$ and $G_2$ admit no simultaneous

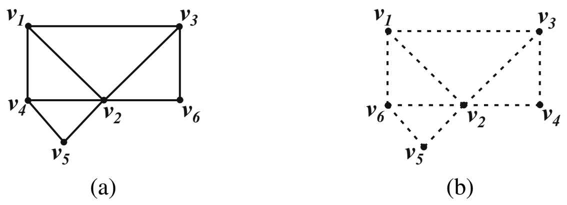


**Fig. 1.** Outerplanar graphs (a) $G_1$ and (b) $G_2$

embedding with mapping $\gamma$ between their vertices, we try to construct embeddings $E_1$ of $G_1$ and $E_2$ of $G_2$, proving that at least one between $E_1$ and $E_2$ must be non planar.

First we embed vertices $v_1$, $v_2$, and $v_3$. These vertices form a cycle $C_1$, that is common to $E_1$ and $E_2$, and that divides the plane in two parts, one inside and one outside $C_1$. Then, after having drawn $v_4$ and its incident edges, the plane is subdivided in four regions $F_1$, $F_2$, $F_3$, and $F_4$, delimited by cycles between vertices $(v_1, v_2, v_3)$, $(v_1, v_2, v_4)$, $(v_1, v_3, v_4)$, and $(v_2, v_3, v_4)$, respectively, each one with the fourth vertex outside. This is because drawing $v_4$ inside or outside $C_1$ and changing the clockwise order of the edges incident in $v_4$ only permit to choose the external face of the simultaneous embedding, as shown in Fig. 2 (a) – (d). Since only the edges $(v_1, v_4)$ and $(v_3, v_4)$ can intersect, regions $F_2$, $F_3$ and $F_4$ can overlap, while region $F_1$ can not intersect any other region, as shown in Fig. 2 (e).

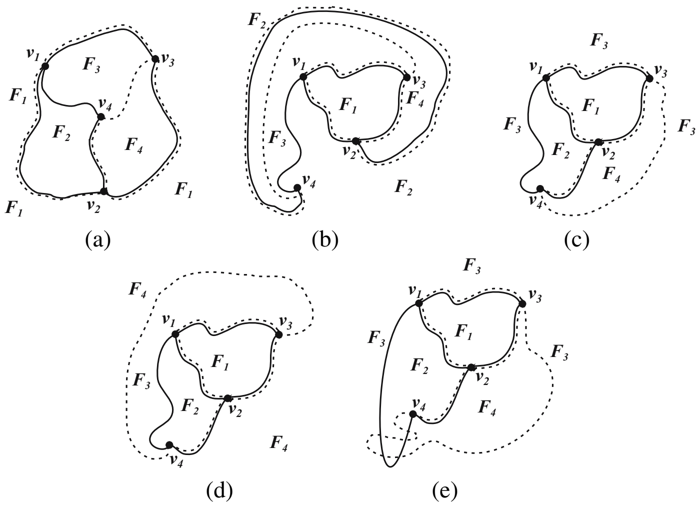


**Fig. 2.** Embedding vertices $v_1$, $v_2$, $v_3$, and $v_4$ with different external faces. (a) $F_1$. (b) $F_2$. (c) $F_3$. (d) $F_4$. (e) Intersection between faces of $E_1$ and $E_2$.

Now we embed $v_6$ and its incident edges. It is easy to observe that $v_6$ must be placed inside region $F_1$. In fact if $v_6$ is placed inside $F_2$, then $(v_3, v_6)$ intersects the cycle $(v_1, v_2, v_4)$ in $E_1$; if $v_6$ is placed inside $F_4$, then $(v_1, v_6)$ intersects the cycle $(v_2, v_3, v_4)$ in $E_2$; if $v_6$ is placed inside $F_3$, then $(v_2, v_6)$ either intersects an edge between $(v_1, v_3)$ and $(v_1, v_4)$ in $E_1$ or intersects an edge between $(v_1, v_3)$ and $(v_3, v_4)$ in $E_2$. We have shown that $v_4$ and $v_6$ must be placed one inside and the other outside $C_1$. Note that $v_4$ is adjacent to $v_5$ in $G_1$ and $v_6$ is adjacent to $v_5$ in $G_2$. Hence, embedding $v_5$ anywhere in the plane creates an edge $e$ from inside to outside $C_1$. Since $e$ intersects $C_1$, that is common to $G_1$ and $G_2$, there is not a placement for $v_5$ preserving the planarity of both $E_1$ and $E_2$ and this concludes the proof.

## 4 Simultaneous Embedding of a Planar Graph and a Tree

In this section we prove the following theorem:

**Theorem 2.** *A planar graph and a tree can be simultaneously embedded with fixed edges with any established mapping between their vertices.*

To prove Theorem 2 we show how to construct a simultaneous drawing of a planar graph $G$ and of a tree $T$ with any mapping $\gamma$ between their vertices.

Augment $G$ to a triangular graph $G'$, by adding edges that will be removed later and then, by using one of the well-known methods to draw with straight-line edges a triangular graph ([5,6,10]), construct a straight-line drawing of $G'$ with dual graph $D$. Note that the edges of $T$ that are common to edges of $G'$ have been already drawn. Let $E_T$ denote the set of edges of $T$ that have still to be drawn. From now on, we call $\Gamma$ the current simultaneous drawing of $G'$ and $T$. Let also $S_N$ denote the set of already drawn segments representing straight-line edges of $T$ or representing parts of poly-line edges of $T$. Now, until all the edges in $E_T$ have been drawn, choose one of them, say $(u, v)$, and draw it by using the following procedure that we call *Add Edge*.

1. Select a simple path $p = (f_1, f_2, \ldots, f_k)$ in $D$ with the following properties: (i) $f_1$ is dual to a face of $G'$ that contains $u$, (ii) $f_k$ is dual to a face of $G'$ that contains $v$, and (iii) each edge $(f_i, f_{i+1})$, $1 \le i < k$, is not dual to an edge of $G'$ represented by a segment in $S_N$. Then, for every face $F_i$ of $G'$ that is dual to a vertex $f_i$ of $p$, $1 \le i \le k$, choose a point $p_i$ in the interior of the region representing $F_i$ in $\Gamma$. For every edge $(w_1, w_2)$ of $G'$ dual to an edge $(f_i, f_{i+1})$ of $p$, $1 \le i < k$, choose a point $e_i$ in the interior of the segment representing $(w_1, w_2)$ in $\Gamma$ (see Fig. 3 (a)).

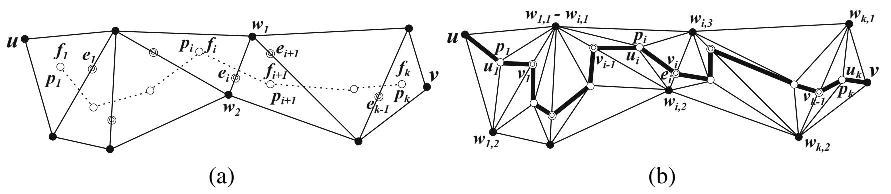


**Fig. 3.** (a) A path $p$ in the dual graph $D$ of $G'$, as in Step 1 of *Add Edge*. The vertices of $G'$ (of $D$) are the black ones (the white ones). The $p_i$'s are choosen coincident with the vertices of $D$. The double circles show the $e_i$'s. (b) $G'$ augmented and drawn as in Steps 2 and 3 of *Add Edge*. Both the thin and the thick edges belong to $G'$. The thick segments show the polygonal line $e_{pol}$, representing the edge $(u, v)$ of $T$ in $\Gamma$.

2. Augment $G'$ by adding to it a vertex $u_i$ for each vertex $f_i$ of $p$, $1 \le i \le k$, and a vertex $v_i$ for each edge $(f_i, f_{i+1})$ of $p$, $1 \le i < k$. Add also to $G'$ the following edges: for each face $F_i$ dual to $f_i \in p$, $1 < i < k$, let $(w_{i,1}, w_{i,2})$ and $(w_{i,2}, w_{i,3})$ be the edges of $F_i$ dual to $(f_{i-1}, f_i)$ and $(f_i, f_{i+1})$, respectively. Add to $G'$ the edges $(w_{i,1}, u_i)$, $(w_{i,2}, u_i)$, $(w_{i,3}, u_i)$, $(v_{i-1}, u_i)$, and $(v_i, u_i)$. Split $(w_{i,1}, w_{i,2})$ in two edges $(w_{i,1}, v_{i-1})$ and $(w_{i,2}, v_{i-1})$ and split $(w_{i,2}, w_{i,3})$ in two edges $(w_{i,2}, v_i)$ and

$(w_{i,3}, v_i)$. For the face $F_1$ dual to $f_1 \in p$ let $a$ and $b$ be the vertices of $F_1$ distinct from $u$, and let $(w_{1,1}, w_{1,2})$ be the edge of $F_1$ dual to $(f_1, f_2)$. Add to $G'$ edges $(u, u_1)$, $(a, u_1)$, $(b, u_1)$, and $(v_1, u_1)$. Split $(w_{1,1}, w_{1,2})$ in two edges $(w_{1,1}, v_1)$ and $(w_{1,2}, v_1)$. For the face $F_k$ dual to $f_k \in p$ let $c$ and $d$ be the vertices of $F_k$ distinct from $v$ and let $(w_{k,1}, w_{k,2})$ be the edge of $F_k$ dual to $(f_{k-1}, f_k)$. Add to $G'$ edges $(v, u_k)$, $(c, u_k)$, $(d, u_k)$, and $(v_{k-1}, u_k)$. Split $(w_{k,1}, w_{k,2})$ in two edges $(w_{k,1}, v_{k-1})$ and $(w_{k,2}, v_{k-1})$.

3. As shown in Fig. 3 (b), map each $u_i$ to $p_i$, $1 \leq i \leq k$ and each $v_i$ to $e_i$, $1 \leq i < k$. Draw the edges added to $G'$ as straight lines. Draw the edge $(u, v)$ of $T$ in $\Gamma$ as a polygonal line $e_{pol}$ passing through points $u, p_1, e_1, \ldots, p_i, e_i, \ldots, p_{k-1}, e_{k-1}, p_k, v$.
4. Remove $(u, v)$ from $E_T$ and add to $S_N$ every segment of $e_{pol}$.

When $E_T = \varnothing$, the final drawing of $G$ is obtained by deleting from the current $G'$ all the edges not belonging to $G$. Note that it is possible that some edges of $G$ are now represented by a polygonal line obtained by repeatedly splitting the starting straight-line edge. The final drawing of $T$ is formed by all the segments in $S_N$.

We now show that the above described algorithm constructs a simultaneous embedding of $G$ and $T$. To check this, we show that: (i) the construction of the simultaneous drawing of $G$ and $T$ starts from a partial simultaneous planar drawing, (ii) the planarity of the drawing of $G$ is preserved after each application of *Add Edge*, (iii) the planarity of the drawing of $T$ is preserved after each application of *Add Edge*, (iv) one can always apply *Add Edge* until all the edges in $E_T$ have been drawn, and (v) each edge common to $G$ and $T$ has the same representation in both the drawings of $G$ and of $T$.

(i) The planarity of the starting simultaneous drawing $\Gamma$ is a consequence of the planarity of the straight-line drawings obtained by applications of the algorithms in [5,6,10].

(ii) Each time one applies *Add Edge* to draw an edge of $E_T$, $G'$ is augmented by adding new vertices and new edges to it. This is done in such a way that both the triangulation of $G'$ and the planarity of its current drawing are preserved, as can be easily checked (see Figure 3).

(iii) After each execution of *Add Edge* the subgraph of $T$ that has been already drawn is a subgraph of $G'$. By construction, the last assertion is true after triangulating $G$ and after straight-line drawing $G'$ and it remains true also after each application of *Add Edge*. To prove this, observe that at step 2 of *Add Edge* one augments $G'$ by adding some new vertices and edges to it, then at step 3 one draws these new vertices and edges. Then an edge of $T$ is drawn as a polygonal line whose bends coincide with the new vertices of $G'$ and whose edges coincide with some of the new edges of $G'$. So the planarity of the drawing of $T$ is a consequence of the planarity of the drawing of $G'$.

(iv) Suppose that when we are starting a new execution of *Add Edge* the dual graph $D$ of $G'$ is not connected after the removal of the edges that are dual to edges of $T$ represented by segments in $S_N$. This is equivalent of saying that the removed edges form a cutset for $D$. From [8] we know that:

**Lemma 1.** *Let $G$ be a planar graph and $D$ be a geometric dual of $G$, then a set of edges in $G$ forms a cycle (or cutset) in $G$ if and only if the corresponding set of edges of $D$ forms a cutset (res. cycle) in $D$.*

So the set of edges of $T$ already drawn forms a cycle and this gives us a contradiction, since $T$ is a tree. So $D$ is connected even after removing from it the edges that are dual to edges of $T$ already drawn and this permits us to select a path $p$ in $D$ with the properties described at step 1 of *Add Edge* and to apply *Add Edge*.

(v) After triangulating $G$ and after drawing the resulting triangular graph $G'$, the edges that are common to both graphs are drawn as straight-line segments between the same end-points and while applying *Add Edge* they are splitted in the same way. So the edges that are common to $G$ and $T$ have the same drawing in $\Gamma$.

## 5 Conclusions

In this paper we have shown that two outerplanar graphs can not always be simultaneously embedded with fixed edges, while a planar graph and a tree can. Observe that after the last execution of *Add Edge* (see Section 4) the dual graph $D$ of the augmented triangular graph $G'$ is still connected even after the removal from $D$ of the edges dual to edges of the tree. Hence an other execution of *Add Edge* is still possible and so the algorithm proposed in Section 4 works more generally when the first graph is planar and the second is a tree augmented by an edge. This straightforwardly implies that the algorithm works also for a planar graph and a cycle. A drawback of the algorithm in Section 4 is that of using a large number of bends, and so it remains an open problem to find an algorithm for drawing a planar graph and a tree with fixed edges with only a small number of bends and within a small area. As far as we know, it is also still open the geometric simultaneous embedding of a tree and a path and the simultaneous embedding without mapping ([1]) of two planar graphs.

## References

1. P. Brass, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. B. Mitchell. On simultaneous planar graph embeddings. In *WADS*, pages 243–255, 2003.
2. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.
3. E. Di Giacomo and G. Liotta. A note on simultaneous embedding of planar graphs (abstract). In *Proc. EWCG 2005*, pages 207–210, 2005.
4. C. Erten and S. G. Kobourov. Simultaneous embedding of planar graphs with few bends. In *Graph Drawing*, pages 195–205, 2004.
5. I. Fáry. On straight line representation of planar graphs. *Ac. Sci. Math. Sz.*, 11:229–233, 1948.
6. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.
7. M. Geyer, M. Kaufmann, and I. Vrto. Two trees which are self-intersecting when drawn simultaneously. In *Graph Drawing*, pages 201–210, 2005.
8. T. Nishizeki and N. Chiba. *Planar Graphs: Theory and Algorithms*. North-Holland, Amsterdam, 1988.
9. J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. In *Graph Drawing*, pages 263–274, 1998.
10. W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Sympos. Discr. Alg.*, pages 138–148, 1990.

# Drawing Cubic Graphs with at Most Five Slopes

B. Keszegh[1], J. Pach[2,4], D. Pálvölgyi[3], and G. Tóth[4]

[1] Central European University, Budapest
[2] Courant Institute, NYU, New York
[3] Eötvös University, Budapest
[4] A. Rényi Institute of Mathematics, Budapest

**Abstract.** We show that every graph $G$ with maximum degree three has a straight-line drawing in the plane using edges of at most five different slopes. Moreover, if $G$ is connected and has at least one vertex of degree less than three, then four directions suffice.

## 1 Introduction

A planar layout of a graph $G$ is called a *straight-line drawing* if the vertices of $G$ are represented by distinct points in the plane and every edge is represented by a straight-line segment connecting the corresponding pair of points and not passing through any other point representing a vertex. If it leads to no confusion, in notation and terminology we make no distinction between a vertex and the corresponding point and between an edge and the corresponding segment. The *slope* of an edge of the layout is the slope of the segment representing it. Layouts with few slopes and few bends have been extensively studied in "graph drawing" [2]. In particular, Ungar proved that every three-connected *cubic* planar graph (i.e., every vertex has degree three) can be drawn using only vertical and horizontal straight-line edges and altogether at most three bends on the outer-face [8].

Wade and Chu [9] introduced the following graph parameter: The *slope number* of a graph $G$ is the smallest number $s$ with the property that $G$ has a straight-line drawing with edges of at most $s$ distinct slopes and with no bends. Obviously, if $G$ has a vertex of degree $d$, then its slope number is at least $\lceil d/2 \rceil$, because, according to the above definitions, in a proper drawing two edges are not allowed to partially overlap. The question arises whether the slope number can be bounded from above by any function of the maximum degree $d$ (see [3]). Barát, Matoušek, and Wood [1] and, independently, Pach and Pálvölgyi [7] proved that the answer is no for $d \geq 5$. Trivially, every graph of maximum degree two has slope number at most three. What happens if $d = 3$ or 4?

The aim of this note is to establish the following theorem.

**Theorem 1.** *Every graph of maximum degree at most three has slope number at most five.*

Our terminology is somewhat unorthodox: by the *slope* of a line $\ell$, we mean the angle $\alpha$ modulo $\pi$ such that a counterclockwise rotation through $\alpha$ takes the

$x$-axis to a position parallel to $\ell$. The slope of an edge (segment) is the slope of the line containing it. In particular, the slopes of the lines $y = x$ and $y = -x$ are $\pi/4$ and $-\pi/4$, and they are called *Northeast* (or Southwest) and *Northwest* (or Southeast) lines, respectively.

For any two points $p_1 = (x_1, y_1), p_2 = (x_2, y_2) \in \mathbf{R}^2$, we say that $p_2$ is *to the North* (or *to the South* of $p_1$ if $x_2 = x_1$ and $y_2 > y_1$ (or $y_2 < y_1$). Analogously, we say that $p_2$ is to the Northeast (to the Northwest) of $p_1$ if $y_2 > y_1$ and $p_1p_2$ is a Northeast (Northwest) line. Directions are often abbreviated by their first letters: N, NE, E, SE, etc. These four directions are referred to as *basic*. That is, a line $\ell$ is said to be of one of the four basic directions if $\ell$ is parallel to one of the axes or to one of the NE and NW lines $y = x$ and $y = -x$.

The main tool of our proof is the following result of independent interest.

**Theorem 2.** *Let $G$ be a connected graph that is not a cycle and whose every vertex has degree at most three. Suppose that $G$ has at least one vertex of degree less than three, and denote by $v_1, ..., v_m$ the vertices of degree at most two ($m \geq 1$).*

*Then, for any sequence $x_1, x_2, \ldots, x_m$ of real numbers, linearly independent over the rationals, $G$ has a straight-line drawing with the following properties:*
(1) *Vertex $v_i$ is mapped into a point with $x$-coordinate $x(v_i) = x_i$ ($1 \leq i \leq m$);*
(2) *The slope of every edge is $0, \pi/2, \pi/4$, or $-\pi/4$.*
(3) *No vertex is to the North of any vertex of degree two.*
(4) *No vertex is to the North or to the Northwest of any vertex of degree one.*

It was shown by Dujmović at al. [3] that every planar graph with maximum degree three has a drawing with noncrossing straight-line edges of at most three different slopes, except that three edges of the outer-face may have a bend.

Eppstein [6], Duncan et al. [4], and Barát et al. [1] studied another parameter, the *geometric thickness* of a graph, which is closely related to the slope number.

Max Engelstein [5], a student from Stuyvesant High School, New York has shown that every graph of maximum degree *three* that has a Hamiltonian cycle can be drawn with edges of at most *five* different slopes.

## 2 Embedding Cycles

Let $C$ be a straight-line drawing of a cycle in the plane. A vertex $v$ of $C$ is said to be a *turning point* if the slopes of the two edges meeting at $v$ are not the same.

We start with two simple auxiliary statements.

**Lemma 2.1.** *Let $C$ be a straight-line drawing of a cycle such that the slope of every edge is $0$, $\pi/4$, or $-\pi/4$. Then the $x$-coordinates of the vertices of $C$ are not independent over the rational numbers.*

*Moreover, there is a vanishing linear combination of the $x$-coordinates of the vertices, with as many nonzero (rational) coefficients as many turning points $C$ has.*

**Proof.** Let $v_1, v_2, \dots, v_n$ denote the vertices of $C$ in cyclic order ($v_{n+1} = v_1$). Let $x(v_i)$ and $y(v_i)$ be the coordinates of $v_i$. For any $i$ ($1 \le i \le n$), we have $y(v_{i+1}) - y(v_i) = \lambda_i \left(x(v_{i+1}) - x(v_i)\right)$, where $\lambda_i = 0, 1$, or $-1$, depending on the slope of the edge $v_i v_{i+1}$. Adding up these equations for all $i$, the left-hand sides add up to zero, while the sum of the right-hand sides is a linear combination of the numbers $x(v_1), x(v_2), \dots, x(v_n)$ with integer coefficients of absolute value at most *two.*

Thus, we are done with the first statement of the lemma, unless all of these coefficients are zero. Obviously, this could happen if and only if $\lambda_1 = \lambda_2 = \dots = \lambda_n$, which is impossible, because then all points of $C$ would be collinear, contradicting our assumption that in a proper *straight-line drawing* no edge is allowed to pass through any vertex other than its endpoints.

To prove the second statement, it is sufficient to notice that the coefficient of $x(v_i)$ vanishes if and only if $v_i$ is not a turning point. □

Lemma 2.1 shows that Theorem 2 does not hold if $G$ is a cycle. Nevertheless, according to the next claim, cycles satisfy a very similar condition. Observe, that the main difference is that here we have an exceptional vertex, denoted by $v_0$.

**Lemma 2.2.** *Let $C$ be a cycle with vertices $v_0, v_1, \dots, v_m$, in this cyclic order.*

*Then, for any real numbers $x_1, x_2, \dots, x_m$, linearly independent over the rationals, $C$ has a straight-line drawing with the following properties:*
(1) *Vertex $v_i$ is mapped into a point with $x$-coordinate $x(v_i) = x_i$ ($1 \le i \le m$);*
(2) *The slope of every edge is $0, \pi/4$, or $-\pi/4$.*
(3) *No vertex is to the North of any other vertex.*
(4) *No vertex has a larger $y$-coordinate than $y(v_0)$.*

**Proof.** We can assume without loss of generality that $x_2 > x_1$. Place $v_1$ at any point $(x_1, 0)$ of the $x$-axis. Assume that for some $i < m$, we have already determined the positions of $v_1, v_2, \dots v_i$, satisfying conditions (1)–(3). If $x_{i+1} > x_i$, then place $v_{i+1}$ at the (unique) point Southeast of $v_i$, whose $x$-coordinate is $x_{i+1}$. If $x_{i+1} < x_i$, then put $v_{i+1}$ at the point West of $x_i$, whose $x$-coordinate is $x_{i+1}$. Clearly, this placement of $v_{i+1}$ satisfies (1)–(3), and the segment $v_i v_{i+1}$ does not pass through any point $v_j$ with $j < i$.

After $m$ steps, we obtain a noncrossing straight-line drawing of the path $v_1 v_2 \dots v_m$, satisfying conditions (1)–(3). We still have to find a right location for $v_0$. Let $R_W$ and $R_{SE}$ denote the rays (half-lines) starting at $v_1$ and pointing to the West and to the Southeast. Further, let $R$ be the ray starting at $v_m$ and pointing to the Northeast. It follows from the construction that all points $v_2, \dots, v_m$ lie in the convex cone below the $x$-axis, enclosed by the rays $R_W$ and $R_{SE}$.

Place $v_0$ at the intersection point of $R$ and the $x$-axis. Obviously, the segment $v_m v_0$ does not pass through any other vertex $v_j$ ($0 < j < m$). Otherwise, we could find a drawing of the cycle $v_j v_{j+1} \dots v_m$ with slopes $0, \pi/4$, and $-\pi/4$. By Lemma 2.1, this would imply that the numbers $x_j, x_{j+1}, \dots, x_m$ are *not* independent over the rationals, contradicting our assumption. It is also clear that the horizontal segment $v_0 v_1$ does not pass through any vertex different from its endpoints because all other vertices are below the horizontal line determined by $v_0v_1$. Hence, we obtain a proper straight-line drawing of $C$ satisfying conditions (1),(2), and (4).

It remains to verify (3). The only thing we have to check is that $x(v_0)$ does not coincide with any other $x(v_i)$. Suppose it does, that is, $x(v_0) = x(v_i) = x_i$ for some $i > 0$. By the second statement of Lemma 2.1, there is a vanishing linear combination

$$\lambda_0 x(v_0) + \lambda_1 x_1 + \lambda_2 x_2 + \ldots + \lambda_m x_m = 0$$

with rational coefficients $\lambda_i$, where the number of nonzero coefficients is at least the number of turning points, which cannot be smaller than *three*. Therefore, if in this linear combination we replace $x(v_0)$ by $x_i$, we still obtain a nontrivial rational combination of the numbers $x_1, x_2, \ldots, x_m$. This contradicts our assumption that these numbers are independent over the rationals. □

## 3 The Embedding Procedure: Proof of Theorem 2

First we settle Theorem 2 in a special case.

**Lemma 3.1** *Let $m, k \geq 2$ and let $G$ be a graph consisting of two disjoint cycles, $C = \{v_0, v_1, \ldots, v_m\}$ and $C' = \{v'_0, v'_1, \ldots, v'_m\}$, connected by a single edge $v_0v'_0$.*

*Then, for any sequence $x_1, x_2, \ldots, x_m, x'_1, x'_2, \ldots, x'_k$ of real numbers, linearly independent over the rationals, $G$ has a straight-line drawing satisfying the following conditions:*
(1) *The vertices $v_i$ and $v'_j$ are mapped into points with $x$-coordinates $x(v_i) = x_i$ $(1 \leq i \leq m)$ and $x(v_j) = x'_j$ $(1 \leq j \leq k)$.*
(2) *The slope of every edge is $0, \pi/2, \pi/4$, or $-\pi/4$.*
(3) *No vertex is to the North of any vertex of degree two.*

**Proof of Lemma 3.1.** Apply Lemma 2.2 to cycle $C$ with vertices $v_0, v_1, \ldots, v_m$, with assigned $x$-coordinates $x_1, x_2, \ldots, x_m$, and analogously, to the cycle $C'$, with vertices $v'_0, v'_1, \ldots, v'_k$ and assigned $x$-coordinates $x'_1, x'_2, \ldots, x'_k$. For simplicity, the resulting drawings are also denoted by $C$ and $C'$.

Let $x_0$ and $x'_0$ denote the $x$-coordinates of $v_0 \in C$ and $v'_0 \in C'$. It follows from Lemma 2.1 that $x_0$ is a linear combination of $x_1, x_2, \ldots, x_m$, and $x'_0$ is a linear combination of $x'_1, x'_2, \ldots, x'_k)$ with rational coefficients. Therefore, if $x_0 = x'_0$, then there is a nontrivial linear combination of $x_1, x_2, \ldots, x_m, x'_1, x'_2, \ldots, x'_k$ that gives 0, contradicting the assumption that these numbers are independent over the rationals. Thus, we can conclude that $x_0 \neq x'_0$. Assume without loss of generality that $x_0 < x'_0$. Reflect $C'$ about the $x$-axis, and shift it in the vertical direction so that $v'_0$ ends up to the Northeast from $v_0$. Clearly, we can add the missing edge $v_0v'_0$. Let $D$ denote the resulting drawing of $G$. We claim that $D$ meets all the requirements of the Theorem. Conditions (1), (2), and (3) are obviously satisfied, we only have to check that no vertex lies in the interior of an edge. It follows from Lemma 2.2 that the $y$-coordinates of $v_1, \ldots, v_m$ are all smaller than or equal to the $y$-coordinate of $v_0$ and the $y$-coordinates of

$v'_1, \ldots, v'_k$ are all greater than or equal to the $y$-coordinate of $v'_0$. We also have $y(v_0) < y(v'_0)$. Therefore, there is no vertex in the interior of $v_0v'_0$. Moreover, no edge of $C$ (resp. $C'$) can contain any vertex of $v'_0, v'_1, \ldots, v'_k$ (resp. $v_0, v_1, \ldots, v_m$) in its interior. □

The rest of the proof is by induction on the number of vertices of $G$. The statement is trivial if the number of vertices is at most *two*. Suppose that we have already established Theorem 2 for all graphs with fewer than $n$ vertices.

Suppose that $G$ has $n$ vertices, it is not a cycle and not the union of two cycles connected by one edge. Let $v_1, v_2, \ldots, v_m$ be the vertices of $G$ with degree less than *three*, and let the $x$-coordinates assigned to them be $x_1, x_2, \ldots, x_m$.

We distinguish several cases.

**Case 1:** *G has a vertex of degree one.*

Assume, without loss of generality, that $v_1$ is such a vertex. If $G$ has no vertex of degree *three*, then it consists of a simple path $P = v_1v_2 \ldots v_m$, say. Place $v_m$ at the point $(x_m, 0)$. In general, assuming that $v_{i+1}$ has already been embedded for some $i < m$, and $x_i < x_{i+1}$, place $v_i$ at the point West of $v_{i+1}$, whose $x$-coordinate is $x_i$. If $x_i > x_{i+1}$, then put $v_i$ at the point Northeast of $v_{i+1}$, whose $x$-coordinate is $x_i$. The resulting drawing of $G = P$ meets all the requirements of the theorem. To see this, it is sufficient to notice that if $v_j$ would be Northwest of $v_m$ for some $j < m$, then we could apply Lemma 2.1 to the cycle $v_jv_{j+1} \ldots v_m$, and conclude that the numbers $x_j, x_{j+1}, \ldots, x_m$ are dependent over the rationals. This contradicts our assumption.

Assume next that $v_1$ is of degree *one*, and that $G$ has at least one vertex of degree *three*. Suppose without loss of generality that $v_1v_2 \ldots v_kw$ is a path in $G$, whose internal vertices are of degree *two*, but the degree of $w$ is *three*. Let $G'$ denote the graph obtained from $G$ by removing the vertices $v_1, v_2, \ldots, v_k$. Obviously, $G'$ is a connected graph, in which the degree of $w$ is *two*.

If $G'$ is a *cycle*, then apply Lemma 2.2 to $C = G'$ with $w$ playing the role of the vertex $v_0$ which has no preassigned $x$-coordinate. We obtain an embedding of $G'$ with edges of slopes $0, \pi/4$, and $-\pi/4$ such that $x(v_i) = x_i$ for all $i > k$ and there is no vertex to the North, to the Northeast, or to the Northwest of $w$. By Lemma 2.1, the numbers $x(w), x_{k+1}, \ldots, x_m$ are not independent over the rationals. Therefore, $x(w) \neq x_k$, so we can place $v_k$ at the point to the Northwest or to the Northeast of $w$, whose $x$-coordinate is $x_k$, depending on whether $x(w) > x_k$ or $x(w) < x_k$. After this, embed $v_{k-1}, \ldots, v_1$, in this order, so that $v_i$ is either to the Northeast or to the West of $v_{i+1}$ and $x(v_i) = x_i$. According to property (4) in Lemma 2.1, the path $v_1v_2 \ldots v_k$ lies entirely above $G'$, so that no point of $G$ can lie to the North or to the Northwest of $v_1$.

If $G'$ is *not a cycle*, then use the induction hypothesis to find an embedding of $G'$ that satisfies all conditions of Theorem 2, with $x(w) = x_k$ and $x(v_i) = x_i$ for every $i > k$. Now place $v_k$ very far from $w$, to the North of it, and draw $v_{k-1}, \ldots, v_1$, in this order, in precisely the same way as in the previous case. Now if $v_k$ is far enough, then none of the points $v_k, v_{k-1}, \ldots, v_1$ is to the Northwest or to the Northeast of any vertex of $G'$. It remains to check that condition (4) is true for $v_1$, but this follows from the fact that there is no point of $G$ whose $y$-coordinate is larger than that of $v_1$.

From now on, we can and will assume that $G$ has *no vertex of degree one.*

A graph with *four* vertices and *five* edges between them is said to be a *$\Theta$-graph.*

**Case 2:** *$G$ contains a $\Theta$-subgraph.*

Suppose that $G$ has a $\Theta$-subgraph with vertices $a, b, c, d$, and edges $ab$, $bc$, $ac$, $ad$, $bd$. If neither $c$ nor $d$ has a third neighbor, then $G$ is identical to this graph, which can easily be drawn in the plane with all conditions of the theorem satisfied.

If $c$ and $d$ are connected by an edge, then all four points of the $\Theta$-subgraph have degree *three*, so that $G$ has no other vertices. So $G$ is a complete graph of four vertices, and it has a drawing that meets the requirements.

Suppose that $c$ and $d$ have a common neighbor $e \neq a, b$. If $e$ has no further neighbor, then $a, b, c, d, e$ are the only vertices of $G$, and again we can easily find a proper drawing. Thus, we can assume that $e$ has a third neighbor $f$. By the induction hypothesis, $G' = G \setminus \{a, b, c, d, e\}$ has a drawing satisfying the conditions of Theorem 2. In particular, no vertex of $G'$ is to the North of $f$ (and to the Northwest of $f$, provided that the degree of $f$ in $G'$ is *one*). Further, consider a drawing $H$ of the subgraph of $G$ induced by the vertices $a, b, c, d, e$, which satisfies the requirements. We distinguish two subcases.

If the degree of $f$ in $G'$ is *one*, then take a very small *homothetic* copy of $H$ (i.e., similar copy in parallel position), and rotate it about $e$ in the clockwise direction through $3\pi/4$. There is no point of this drawing, denoted by $H'$, to the Southeast of $e$, so that we can translate it into a position in which $e$ is to the Northwest of $f \in V(G')$ and very close to it. Connecting now $e$ to $f$, we obtain a drawing of $G$ satisfying the conditions. Note that it was important to make $H'$ very small and to place it very close to $f$, to make sure that none of its vertices is to the North of any vertex of $G'$ whose degree is at most *two*, or to the Northwest of any vertex of degree *one* (other than $f$).

If the degree of $f$ in $G'$ is *two*, then we follow the same procedure, except that now $H'$ is a small copy of $H$, rotated by $\pi$. We translate $H'$ into a position in which $e$ is to the North of $f$, and connect $e$ to $f$ by a vertical segment. It is again clear that the resulting drawing of $G$ meets the requirements in Theorem 2. Thus, we are done if $c$ and $d$ have a common neighbor $e$.

Suppose now that only one of $c$ and $d$ has a third neighbor, different from $a$ and $b$. Suppose, without loss of generality, that this vertex is $c$, so that the degree of $d$ is *two.* Then in $G' = G \setminus \{a, b, d\}$, the degree of $c$ is *one.* Apply the induction hypothesis to $G'$ so that the $x$-coordinate originally assigned to $d$ is now assigned to $c$ (which had no preassigned $x$-coordinate in $G$). In the resulting drawing, we can easily reinsert the remaining vertices, $a, b, d$, by adding a very small square whose lowest vertex is at $c$ and whose diagonals are parallel to the coordinate axes. The highest vertex of this square will represent $d$, and the other two vertices will represent $a$ and $b$.

We are left with the case when both $c$ and $d$ have a third neighbor, other than $a$ and $b$, but these neighbors are different. Denote them by $c'$ and $d'$, respectively. Create a new graph $G'$ from $G$, by removing $a, b, c, d$ and adding a new vertex

$v$, which is connected to $c'$ and $d'$. Draw $G'$ using the induction hypothesis, and reinsert $a, b, c, d$ in a small neighborhood of $v$ so that they form the vertex set of a very small square with diagonal $ab$. (See Figure 1.) As before, we have to choose this square sufficiently small to make sure that $a, b, c, d$ are not to the North of any vertex $w \neq c', d', v$ of $G'$, whose degree is at most *two*, or to the Northwest of any vertex of degree *one*. Thus, we are done if $G$ has a $\Theta$-subgraph.

So, from now on we assume that $G$ has *no $\Theta$-subgraph.*

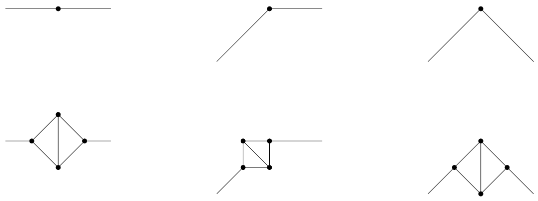


**Fig. 1.** Replacing $v$ by $\Theta$

**Case 3:** *$G$ has no cycle that passes through a vertex of degree* two.

Since $G$ is not three-regular, it contains at least one vertex of degree *two*. Consider a decomposition of $G$ into two-connected blocks and edges. If a block contains a vertex of degree *two*, then it consists of a single edge. The block decomposition has a treelike structure, so that there is a vertex $w$ of degree *two*, such that $G$ can be obtained as the union of two graphs, $G_1$ and $G_2$, having only the vertex $w$ in common, and there is no vertex of degree *two* in $G_1$.

By the induction hypothesis, for any assignment of rationally independent $x$-coordinates to all vertices of degree less than *three*, $G_1$ and $G_2$ have proper straight-line embeddings (drawings) satisfying conditions (1)–(4) of the theorem. The only vertex of $G_1$ with a preassigned $x$-coordinate is $w$. Applying a vertical translation, if necessary, we can achieve that in both drawings $w$ is mapped into the same point. Using the induction hypothesis, we obtain that in the union of these two drawings, there is no vertex in $G_1$ or $G_2$ to the North or to the Northwest of $w$, because the degree of $w$ in $G_1$ and $G_2$ is *one* (property (4)). This is stronger than what we need: indeed, in $G$ the degree of $w$ is *two*, so that we require only that there is no point of $G$ to the North of $w$ (property (3)).

The superposition of the drawings of $G_1$ and $G_2$ satisfies all conditions of the theorem. Only two problems may occur:

1. A vertex of $G_1$ may end up at a point to the North of a vertex of $G_2$ with degree *two*.
2. The (unique) edges in $G_1$ and $G_2$, incident to $w$, may partially overlap.

Notice that both of these events can be avoided by enlarging the drawing of $G_1$, if necessary, from the point $w$, and rotating it about $w$ by $\pi/4$ in the clockwise direction. The latter operation is needed only if problem 2 occurs. This completes the induction step in the case when $G$ has no cycle passing through a vertex of degree *two*.

It remains to analyze the last case.

**Case 4:** *$G$ has a cycle passing through a vertex of degree* two.

By assumption, $G$ itself is not a cycle. Therefore, we can also find a *shortest* cycle $C$ whose vertices are denoted by $v, u_1, \dots, u_k$, in this order, where the degree of $v$ is *two* and the degree of $u_1$ is *three*. The length of $C$ is $k+1$.

It follows from the minimality of $C$ that $u_i$ and $u_j$ are not connected by an edge of $G$, for any $|i-j|>1$. Moreover, if $|i-j|>2$, then $u_i$ and $u_j$ do not even have a common neighbor ($1 \leq i \neq j \leq k$). This implies that any vertex $v \in V(G \setminus C)$ has at most *three* neighbors on $C$, and these neighbors must be consecutive on $C$. However, *three* consecutive vertices of $C$, together with their common neighbor, would form a $\Theta$-subgraph in $G$ (see Case 2). Hence, we can assume that every vertex belonging to $G \setminus C$ is joined to at most *two* vertices on $C$.

Let $B_i$ denote the set of all vertices of $G \setminus C$ that have precisely $i$ neighbors on $C$ ($i = 0, 1, 2$). Thus, we have $V(G \setminus C) = B_0 \cup B_1 \cup B_2$. Further, $B_1 = B_1^2 \cup B_1^3$, where an element of $B_1$ belongs to $B_1^2$ or $B_1^3$, according to whether its degree in $G$ is *two* or *three*.

Consider the list $v_1, v_2, \dots, v_m$ of all vertices of $G$ with degree *two*. (Recall that we have already settled the case when $G$ has a vertex of degree *one*.) Assume without loss of generality that $v_1 = v$ and that $v_i$ belongs to $C$ if and only if $1 \leq i \leq j$ for some $j \leq m$.

Let $\mathbf{x}$ denote the *assignment* of $x$-coordinates to the vertices of $G$ with degree *two*, that is, $\mathbf{x} = (x(v_1), x(v_2), \dots, x(v_m)) = (x_1, x_2, \dots, x_m)$. Given $G$, $C$, $\mathbf{x}$, and a real parameter $L$, we define the following so-called Embedding Procedure$(G, C, \mathbf{x}, L)$ to construct a drawing of $G$ that meets all requirements of the theorem, and satisfies the additional condition that the $y$-coordinate of every vertex of $C$ is at least $L$ higher than the $y$-coordinates of all other vertices of $G$.

Step 1: If $G' := G \setminus C$ is *not* a cycle, then construct recursively a drawing of $G' := G \setminus C$ satisfying the conditions of Theorem 2 with the assignment $\mathbf{x}'$ of $x$-coordinates $x(v_i) = x_i$ for $j < i \leq m$, and $x(u_1') = x_1$, where $u_1'$ is the unique vertex in $G \setminus C$, connected by an edge to $u_1 \in V(C)$.

If $G' = G \setminus C$ is a cycle, then, by assumption, there are at least two edges between $C$ and $G'$. One of them connects $u_1$ to $u_1'$. Let $u_\alpha u_\alpha'$ be another such edge, where $u_\alpha \in C$ and $u_\alpha' \in G'$. Since the maximum degree is three, $u_1' \neq u_\alpha'$. Now construct recursively a drawing of $G' := G \setminus C$ satisfying the conditions of Lemma 2.2, with the assignment $\mathbf{x}'$ of $x$-coordinates $x(v_i) = x_i$ for $j < i \leq m$, $x(u_1') = x_1$, and with exceptional vertex $u_\alpha'$.

Step 2: For each element of $B_1^2 \cup B_2$, take two rays starting at this vertex, pointing to the Northwest and to the North. Further, take a vertical ray pointing to the North from each element of $B_1^3$ and each element of the set $B_{\mathbf{x}} :=$

$\{(x_2, 0), (x_3, 0), \ldots, (x_j, 0)\}$. Let $\mathcal{R}$ denote the set of all of these rays. Choose the $x$-axis above all points of $G'$ and all intersection points between the rays in $\mathcal{R}$.

For any $u_h$ $(1 \leq h \leq k)$ whose degree in $G$ is *three*, define $N(u_h)$ as the unique neighbor of $u_h$ in $G \setminus C$. If $u_h$ has degree *two* in $G$, then $u_h = v_i$ for some $1 \leq i \leq j$, and let $N(u_h)$ be the point $(x_i, 0)$.

Step 3: Recursively place $u_1, u_2, \ldots u_k$ on the rays belonging to $\mathcal{R}$, as follows. Place $u_1$ on the vertical ray starting at $N(u_1) = u_1'$ such that $y(u_1) = L$. Suppose that for some $i < k$ we have already placed $u_1, u_2, \ldots u_i$, so that $L \leq y(u_1) \leq y(u_2) \leq \ldots \leq y(u_i)$ and there is no vertex to the West of $u_i$. Next we determine the place of $u_{i+1}$.

If $N(u_{i+1}) \in B_1^2$, then let $r \in \mathcal{R}$ be the ray starting at $N(u_{i+1})$ and pointing to the Northwest. If $N(u_{i+1}) \in B_1^3 \cup B_{\mathbf{x}}$, let $r \in \mathcal{R}$ be the ray starting at $N(u_{i+1})$ and pointing to the North. In both cases, place $u_{i+1}$ on $r$: if $u_i$ lies on the left-hand side of $r$, then put $u_{i+1}$ to the Northeast of $u_i$; otherwise, put $u_{i+1}$ to the West of $u_i$.

If $N(u_{i+1}) \in B_2$, then let $r \in \mathcal{R}$ be the ray starting at $N(u_{i+1})$ and pointing to the North, or, if we have already placed a point on this ray, let $r$ be the other ray from $N(u_{i+1})$, pointing to the Northwest, and proceed as before.

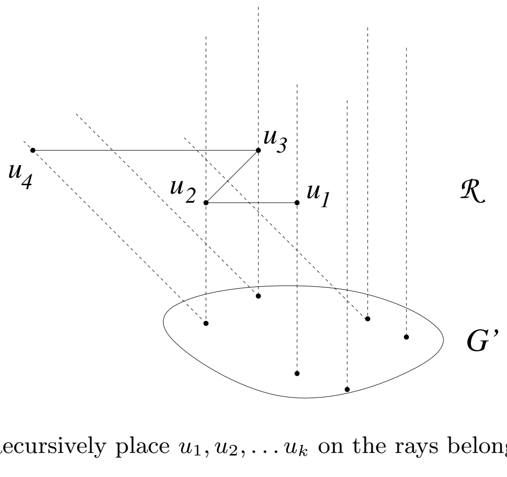


**Fig. 2.** Recursively place $u_1, u_2, \ldots u_k$ on the rays belonging to $\mathcal{R}$

Step 4: Suppose we have already placed $u_k$. It remains to find the right position for $u_0 := v$, which has only two neighbors, $u_1$ and $u_k$. Let $r$ be the ray at $u_1$, pointing to the North. If $u_k$ lies on the left-hand side of $r$, then put $u_0$ on $r$ to the Northeast of $u_k$; otherwise, put $u_0$ on $r$, to the West of $u_k$.

During the whole procedure, we have never placed a vertex on any edge, and all other conditions of Theorem 2 are satisfied □.

Remark that the $y$-coordinates of the vertices $u_0 = v, u_1, \ldots, u_k$ are at least $L$ higher than the $y$-coordinates of all vertices in $G \setminus C$. If we fix $G, C$, and $\mathbf{x}$,

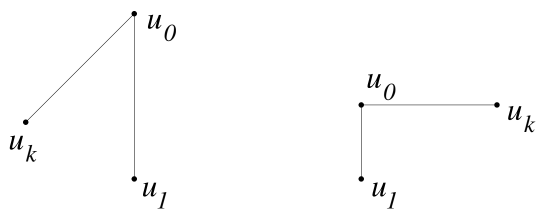


**Fig. 3.** Find the right position for $u_0$

and let $L$ tend to infinity, the coordinates of the vertices given by the above Embedding Procedure$(G, C, \mathbf{x}, L)$ change continuously.

## 4 Proof of Theorem 1

We are going to show that any graph $G$ with maximum degree three permits a straight-line drawing using only the four basic directions (of slopes $0, \pi/2, \pi/4$, and $-\pi/4$), and perhaps one further direction, which is almost vertical and is used for at most one edge in each connected component of $G$.

Denote the connected components of $G$ by $G_1, G_2, \ldots, G_t$. If a component $G_s$ is not three-regular, or if it is a complete graph with *four* vertices, then, by Theorem 2, it can be drawn using only the four basic directions. If $G_s$ has a $\Theta$-subgraph, one can argue in the same way as in Case 2 of the proof of Theorem 2: Embed recursively the rest of the graph, and attach to it a small copy of this subgraph such that all edges of the $\Theta$-subgraph, as well as the edges used for the attachment, are parallel to one of the four basic directions. Actually, in this case, $G_s$ itself can be drawn using the four basic directions, so the fifth direction is not needed.

Thus, in the rest of the proof we can assume that $G_s$ is three-regular, it has more than *four* vertices, and it contains no $\Theta$-subgraph. For simplicity, we drop the subscript and we write $G$ instead of $G_s$. Choose a shortest cycle $C = u_0u_1 \ldots u_k$ in $G$. Each vertex of $C$ has precisely one neighbor in $G \setminus C$. On the other hand, as in the proof of the last case of Theorem 2, all vertices in $G \setminus C$ have at most two neighbors in $C$.

We distinguish two cases.

**Case 1.** $G \setminus C$ is a cycle. Since $G$ is three-regular, $C$ and $G \setminus C$ are of the same size and the remaining edges of $G$ form a matching between the vertices of $C$ and the vertices of $G \setminus C$. For any $i$, $0 \leq i \leq k$, let $u'_i$ denote the vertex of $G \setminus C$ which is connected to $u_i$. Denote the vertices of $G \setminus C$ by $v_0, v_1, \ldots, v_k$, in cyclic order, so that $v_1 = u'_1$. Then we have $v_i = u'_0$, for some $i > 1$. Apply Lemma 2.2 to $G \setminus C$ with a rationally independent assignment $\mathbf{x}$ of $x$-coordinates to the vertices $v_1, \ldots, v_k$, such that $x(v_1) = 1$, $x(v_i) = \sqrt{2}$, and the $x$-coordinates of the other vertices are all greater than $\sqrt{2}$. (Recall that $v_0$ is an exceptional vertex with no assigned $x$-coordinate.) It is not hard to see that if we follow the construction described in the proof of Lemma 2.2, we also have $x(v_0) > \sqrt{2}$.

**Case 2.** $G \setminus C$ is *not* a cycle. Let $u_0'$ denote the neighbor of $u_0$ in $G \setminus C$. Since $G$ has no $\Theta$-subgraph, $u_0'$ cannot be joined to both $u_1$ and $u_k$. Assume without loss of generality that $u_0'$ is not connected to $u_1$. Let $u_1'$ denote the neighbor of $u_1$ in $G \setminus C$.

Fix a rationally independent assignment $\mathbf{x}$ of $x$-coordinates to the vertices of degree at most *two* in $G \setminus C$, such that $x(u_0') = \sqrt{2}$, $x(u_1') = 1$, and the $x$-coordinates of the other vertices are all greater than $\sqrt{2}$. Consider a drawing of $G \setminus C$, meeting the requirements of Theorem 2.

Now in both cases, let $G'$ denote the graph obtained from $G$ after the removal of the edge $u_0u_0'$. Clearly $G \setminus C = G' \setminus C$, and for any $L$, Embedding Procedure$(G', C, \mathbf{x}, L)$ gives a drawing of $G'$. It follows from the construction, that $x(u_0) = x(u_1) = x(u_1') = 1$, $x(u_0') = \sqrt{2}$. Therefore, for any sufficiently small $\varepsilon > 0$ there is an $L > 0$ such that Embedding Procedure$(G', C, \mathbf{x}, L)$ gives a drawing of $G'$, in which the slope of the line connecting $u_0$ and $u_0'$ is $\frac{\pi}{2} + \varepsilon$.

We want to add the segment $u_0u_0'$ to this drawing. Since there is no vertex with $x$-coordinate between 1 and $\sqrt{2}$, the segment $u_0u_0'$ cannot pass through any vertex of $G$.

Summarizing: if $\varepsilon$ is sufficiently small (that is, if $L$ is sufficiently large), then each component of the graph has a proper drawing in which all edges are of one of the four basic directions, with the exception of at most one edge whose slope is $\frac{\pi}{2} + \varepsilon$. If we choose an $\varepsilon > 0$ that works for all components, then the whole graph can be drawn using only at most *five* directions. This concludes the proof of Theorem 1. □

## 5 Algorithm and Concluding Remarks

Based on the proof, it is not hard to design an algorithm to find a proper drawing, in quadratic time.

First, if our graph is a circle, we have no problem drawing it in $O(n)$ steps. If our graph has a vertex of degree *one* then the procedure of Case 1 of the proof of Theorem 2 requires at most $O(m)$ time when we reinsert $v_1, \ldots, v_m$.

We can check if our graph has any $\Theta$-subgraph in $O(n)$ time. If we find one, we can proceed by induction as in Case 2 of the proof of Theorem 2. We can reinsert the $\Theta$-subgraph as described in Case 2 in $O(1)$ time.

Now assume that we have a vertex $v$ of degree *two*. Execute a breadth first search from any vertex, and take a *minimal* vertex of degree two, that is, a vertex $v$ of degree two, all of whose descendants are of degree three. If there is an edge in the graph connecting a descendant of $v$ with a non-descendant, then there is a cycle through $v$; we can find a minimal one with a breadth first search from it and proceed as in Case 4. Otherwise, $v$ can play the role of $w$ in Case 3, and we can proceed recursively.

Finally, if the graph is 3-regular, then we draw each component separately, except the last step, when we have to pick an $\epsilon$ small enough simultaneously for all components, this takes $O(n)$ steps. We only have to find the greatest slope and pick an $\varepsilon$ such that $\frac{\pi}{2} + \varepsilon$ is even steeper.

We believe that this algorithm is far from being optimal. It may perform a breadth first search for each induction step, which is probably not necessary. One may be able to replace this step by repeatedly updating the results of the first search. We cannot even rule out that the problem can be solved in linear time.

## References

1. J. Barát, J. Matoušek, and D. Wood: Bounded-degree graphs have arbitrarily large geometric thickness, *Electronic J. Combinatorics* **13**/1 (2006), R3.
2. G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis: *Graph Drawing*, Prentice Hall, Upper Saddle River, N.J., 1999.
3. V. Dujmović, M. Suderman, and D.R. Wood: Really straight graph drawings, in: *Graph Drawing (GD'04), J. Pach, ed., Lecture Notes in Computer Science* **3383**, Springer-Verlag, Berlin, 2005, 122–132.
4. C. A. Duncan, D. Eppstein, and S.G. Kobourov: The geometric thickness of low degree graphs, in: *Proc. 20th ACM Symp. on Computational Geometry (SoCG'04)*, ACM Press, 2004, 340–346.
5. M. Engelstein: Drawing graphs with few slopes, Research paper submitted to the Intel Competition for high school students, New York, October 2005.
6. D. Eppstein: Separating thickness from geometric thickness, in: *Towards a Theory of Geometric Graphs (J. Pach, ed.), Contemporary Mathematics* **342**, Amer. Math. Soc, Providence, 2004, 75–86.
7. J. Pach and D. Pálvölgyi: Bounded-degree graphs can have arbitrarily large slope numbers, *Electronic J. Combinatorics* **13**/1 (2006), N1.
8. P. Ungar: On diagrams representing maps, *J. London Math. Soc.* **28** (1953), 336–342.
9. G. A. Wade and J. H. Chu: Drawability of complete graphs using a minimal slope set, *The Computer J.* **37** (1994), 139–142.

# Planarity Testing and Optimal Edge Insertion with Embedding Constraints

Carsten Gutwenger, Karsten Klein, and Petra Mutzel

University of Dortmund, Germany
{carsten.gutwenger,karsten.klein,petra.mutzel}@cs.uni-dortmund.de

**Abstract.** Many practical applications demand additional restrictions on an admissible planar embedding. In particular, constraints on the permitted (clockwise) order of the edges around a vertex, like so-called *side constraints*, abound. In this paper, we introduce a set of hierarchical embedding constraints that also comprises side constraints. We present linear time algorithms for testing if a graph is *ec-planar*, i.e., admits a planar embedding satisfying the given embedding constraints, as well as for computing such an embedding. Moreover, we characterize the set of all possible ec-planar embeddings and consider the problem of finding a planar combinatorial embedding of a planar graph such that an additional edge can be inserted with the minimum number of crossings; we show that this problem can still be solved in linear time under the additional restrictions of embedding constraints.

## 1 Introduction

Graphs are used in numerous application domains for visualizing information. Examples include software engineering, data bases, business process modeling, VLSI-design, and bioinformatics. In many cases, application specific layout rules have to be observed which impose restrictions on an admissible graph layout. Consequently, automatic layout systems have to respect these restrictions in addition to the aesthetic criteria they try to optimize. In database diagrams, for example, links between attributes should enter the tables only at the left or right side of the corresponding attributes, the placement of reactants in chemical reactions or biological pathways should reflect their role within the displayed reactions, and in UML class diagrams, generalization edges should leave a class object at the top and enter a base class object at the bottom. Many of these layout rules impose restrictions on the admissible embeddings for a drawing. Even more important is the possibility to use drawing restrictions in order to express the user's preferences and to guide the layout phase.

In this paper, we consider restrictions on the allowed order of incident edges around a vertex, e.g., to specify groups of edges that have to appear consecutively around the vertex or that have a fixed clockwise order. Such constraints occur, e.g., in form of *side constraints*, where incident edges are assigned to the four sides of a rectangular vertex, or *port constraints* where edges have prescribed attachment points. In particular, we introduce three types of constraints which may be arbitrarily nested: *grouping*, *oriented* (prescribed clockwise order), and *mirror* constraints (prescribed reversible order). We call a planar embedding that fulfills the given set of constraints an *ec-planar* embedding.

Even though constraint handling is an important issue because of its relevance in practical applications, e.g., in interactive graph drawing (see, e.g., [2, 15, 4, 3]), there is only few previous work concerning constraints on the admissible embeddings of a graph. Di Battista et al. [6] consider embedding constraints that appear in database schemas, and Dornheim [8] studies the problem of computing embeddings satisfying topological constraints, where prescribed edges have to be embedded inside or outside of a cycle, repsectively. On the other hand, linear time complexity for planarity testing and embedding has been shown in [14, 5].

Our contribution is a linear time algorithm for testing if a graph with a set of embedding constraints is ec-planar; see Sect. 5. The main challenge is to incorporate oriented constraints, where a given clockwise order needs to be satisfied. Furthermore, we characterize all possible ec-planar embeddings using BC- and SPQR-trees, which also yields a linear time algorithm for computing an ec-planar embedding.

An important optimization goal for laying out graphs is the minimization of crossings. The problem of minimizing the number of crossings in a drawing is NP-hard [9] and no practically efficient method exists so far. The *planarization* approach for crossing minimization first deletes a number of edges until the remaining graph is planar and then carefully reinserts these edges so that the number of crossings is minimized; see [12]. In [13], the problem of optimally inserting an additional edge between vertices $v$ and $w$ into a planar graph $G$ is considered and an algorithm to solve the problem in linear time is given. The algorithm first computes the SPQR-tree $\mathcal{T}$ of $G$ and a shortest path $\Psi$ between nodes in $\mathcal{T}$ whose skeletons contain $v$ and $w$, respectively. The optimal insertion path is constructed by simply concatenating locally optimal insertion paths of the tree nodes on $\Psi$. When embedding constraints have to be considered, locally optimal solutions need not lead to globally optimal solutions and the greedy approach cannot be applied anymore. In Sect. 6, we give a linear time algorithm to solve the optimal edge insertion problem under the presence of embedding constraints. Given an ec-planar graph $G$ with embedding constraints $C$ and an additional edge $e$, our algorithm computes an ec-planar embedding of $G$ with respect to $C$, together with a crossing minimal insertion path for $e$.

All the proofs omitted in this paper can be found in [10].

## 2 Preliminaries

A *combinatorial embedding* of a planar graph $G$ is defined as a clockwise ordering of the incident edges for each vertex with respect to a crossing-free drawing of $G$ in the plane. A *planar embedding* is a combinatorial embedding together with a fixed *external face.*

A *block* is a maximal 2-connected subgraph. The relationship between blocks and cut vertices is given by the *block-cutvertex tree*, or *BC-tree* for short. If $G$

is 2-connected, its *SPQR-tree* $\mathcal{T}$ represents the decomposition of $G$ into its 3-connected components comprising serial, parallel, and 3-connected structures; see [7] for a formal definition. The respective structure is given by a skeleton graph associated with each tree node, which is either a cycle (S-node), a bundle of parallel edges (P-node), or a 3-connected simple graph (R-node). We denote with $\mathit{skeleton}(\mu)$ the skeleton graph associated with node $\mu$. In addition, Q-nodes serve as representatives for the edges of $G$. For each vertex $v$ of $G$, the nodes in $\mathcal{T}$ whose skeletons contain $v$ are called the *allocation nodes* of $v$.

If $G$ is 2-connected and planar, its SPQR-tree $\mathcal{T}$ represents all combinatorial embeddings of $G$. In particular, a combinatorial embedding of $G$ uniquely defines a combinatorial embedding of each skeleton in $\mathcal{T}$, and fixing the combinatorial embedding of each skeleton uniquely defines a combinatorial embedding of $G$.

## 3 Embedding Constraints

Let $G = (V, E)$ be a graph. An embedding constraint specifies the admissible clockwise order of the incident edges of a vertex in a combinatorial embedding of $G$. In this paper, we consider the case where a vertex has at most one embedding constraint and either all or none of the edges incident to a vertex are subject to embedding constraints.

An *embedding constraint* at a vertex $v \in V$ is a rooted, ordered tree $T_v$ such that its leaves are exactly the edges incident to $v$. The inner nodes of $T_v$, also called *constraint-nodes* or *c-nodes* for short, are of three types: *oc-nodes* (oriented constraint-nodes), *mc-nodes* (mirror constraint-nodes), and *gc-nodes* (grouping constraint-nodes). Since $T_v$ is an ordered tree, it imposes an order on its leaves and thus on the incident edges of $v$. We consider this order as a cyclic order and represent all *admissible* cyclic, clockwise orders of the incident edges of $v$ by defining, how the order of the children of c-nodes in $T_v$ can be changed:

**gc-node:** The order of children may be arbitrarily permuted.
**mc-node:** The order of children may be reversed.
**oc-node:** The order of children is fixed.

Fig. 1 gives an example. A c-node with a single child is obviously redundant, therefore we demand that each c-node has at least two children.

Let $C$ be a set of embedding constraints at distinct vertices of $G$. A combinatorial embedding $\Gamma$ of $G$ *observes* the embedding constraints in $C$, if for each embedding constraint $T_v \in C$, the cyclic clockwise order of the edges around $v$ in $\Gamma$ is admissible with respect to $T_v$. A planar embedding observing the embedding constraints in $C$ is an *ec-planar embedding* with respect to $C$, and $(G, C)$ is *ec-planar*, if there exists an ec-planar embedding of $G$ with respect to $C$.

## 4 ec-Expansion

A basic building block of the ec-planarity test is a structural transformation applied to a given graph $G$ with embedding constraints $C$. For each embedding constraint $T_v$ at vertex $v$, this transformation expands $v$ according to the

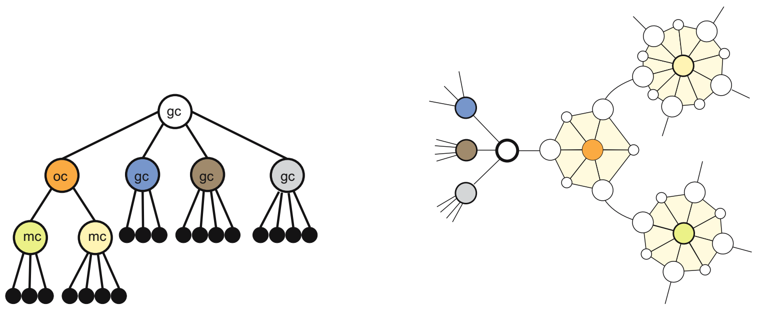


**Fig. 1.** Embedding constraint $T_v$ (left) and the corresponding expansion (right)

structure of $T_v$. We call the resulting graph the *ec-expansion* $E(G, C)$ of $G$ with respect to $C$. The details of this transformation are given below.

### 4.1 Construction of the ec-Expansion

The *ec-expansion* $E(G, C)$ of $G$ with respect to $C$ is constructed as follows. Let $T_v \in C$ be an embedding constraint and $T'_v$ the subgraph obtained from $T_v$ by omitting its leaves. Recall that the leaves of $T_v$ are exactly the edges incident to $v$. We replace $v$ in $G$ by the tree $T'_v$ and connect the edges incident to $v$ with the parents of the corresponding leaves. This transformation introduces a vertex in $G$ for every c-node in $T_v$. Each vertex $u$ corresponding to an oc- or mc-node is further replaced by a *wheel gadget* which is a wheel graph with $2d$ spokes, were $e_1, \ldots, e_d$ are the edges incident to $u$. Then, the respective wheel gadget consists of a cycle $x_1, y_1, \ldots, x_d, y_d$ of length $2d$ and a vertex, called *hub*, incident to every vertex on the cycle. The vertex $u$ is replaced by this wheel gadget, such that $e_i$ is connected to $x_i$ for $1 \leq i \leq d$. According to the type of the expanded c-node, we distinguish between *O-hubs* (oc-nodes) and *M-hubs* (mc-nodes). We refer to the edges introduced during the ec-expansion as *expansion edges*. Fig. 1 shows a constraint tree and the corresponding expansion of the vertex. $E(G, C)$ can be constructed in linear time and its size is also linear in the size of $G$.

The purpose of the wheel gadgets is to model the fixed order of the children of the corresponding c-node. Since a wheel gadget is a 3-connected graph, it admits only two combinatorial embeddings that are mirror images of each other. The order in which non-gadget edges are attached to the wheel cycle is either the order given by the corresponding c-node, or the reverse order. Every face incident to the hub is a triangle; we call these faces *inner wheel gadget faces*.

### 4.2 ec-Expansion and ec-Planar Embeddings

Though the ec-expansion serves as a tool for modeling the embedding constraints in $C$, a planar embedding of $E(G, C)$ needs to fulfill certain conditions in order to induce an ec-planar embedding of $G$ with respect to $C$. We call a planar embedding $\Gamma$ of $E(G, C)$ *ec-planar* if

1. the external face of $\Gamma$ does not contain a hub;
2. every face incident to a hub is a triangle consisting solely of edges of the corresponding wheel gadget; and
3. each O-hub $h$ is *oriented correctly*, i.e., the cyclic, clockwise order of the edges around $h$ in $\Gamma$ corresponds to the order specified by the corresponding oc-node.

Let $\Gamma$ be an ec-planar embedding of $E(G, C)$. Then, we obtain an ec-planar embedding of $(G, C)$ as follows. For each vertex $v \in G$, there is a connected subgraph $G_v$ in $E(G, C)$ resulting from expanding $v$. Let $\bar{G}_v \subset E(G, C)$ be the rest of the graph, i.e., the graph induced by the vertices not contained in $G_v$. The conditions above assure that the planar embedding $\Gamma_v$ of $G_v$ induced by $\Gamma$ is such that $\bar{G}_v$ lies in the external face of $\Gamma_v$. The edges that connect $G_v$ to $\bar{G}_v$ correspond to the edges incident to $v$ in $G$. Their cyclic clockwise order around $G_v$ is admissible with respect to $T_v$, since the wheel gadgets fix the order of the edges specified by oc- and mc-nodes, and O-hubs are oriented correctly. We shrink $G_v$ to a single vertex by contracting all edges in $G_v$ while preserving the embedding, thus resulting in an admissible order of the edges around $v$.

If we have an ec-planar embedding of $(G, C)$, then the edges around each vertex $v$ are ordered such that the constraints in $T_v$ are fulfilled. It is easy to see that we can replace each such vertex $v$ by the expansion graph corresponding to $T_v$ in such a way that we obtain an ec-planar embedding of $E(G, C)$. Thus, we get the following result:

**Lemma 1.** *Let $G$ be a graph with embedding constraints $C$. Then, $(G, C)$ is ec-planar if and only if $E(G, C)$ is ec-planar. Moreover, every ec-planar embedding of $E(G, C)$ induces an ec-planar embedding of $(G, C)$.*

## 5 ec-Planarity Testing

Though it is sufficient to test each block of a graph separately for planarity, this is not the case for ec-planarity. However, it is sufficient to test the blocks of the ec-expansion separately as the following lemma shows.

**Lemma 2.** *$E(G, C)$ is ec-planar iff every block of $E(G, C)$ is ec-planar.*

*Proof.* If $E(G, C)$ is ec-planar, then there is an ec-planar embedding of $E(G, C)$, and this embedding implies an ec-planar embedding for each block of $E(G, C)$.

Suppose now that each block of $E(G, C)$ is ec-planar. Since a wheel gadget $\mathscr{G}$ in $E(G, C)$ is 3-connected, $\mathscr{G}$ is completely contained in a single block $B$ of $E(G, C)$ and therefore also the hub of $\mathscr{G}$ is not a cut vertex of $E(G, C)$. For each edge $(u, v) \in \mathscr{G}$, the pair $\{u, v\}$ is not a separation pair in $B$ by construction, hence every inner wheel face of $\mathscr{G}$ is also a face in every planar embedding of $B$.

We construct an ec-planar embedding of $E(G, C)$ starting with an arbitrary block $B$ of $E(G, C)$. Let $\Pi$ be an ec-planar embedding of $B$. We add the remaining blocks successively to $\Pi$. Let $B'$ be another block of $E(G, C)$ that shares a vertex $c$ with $B$, and let $\Pi'$ be an ec-embedding of $B'$. Since $c$ cannot be an O- or M-hub, we can pick faces $f \in \Pi$ and $f' \in \Pi'$ that are incident to $c$ and not inner wheel gadget faces. We insert $\Pi'$ with $f'$ as external face into the face $f$ of $\Pi$. This results in an ec-planar embedding of $B \cup B'$. We add the remaining blocks in the same way, resulting in an ec-planar embedding of $E(G, C)$. □

If we can characterize all ec-planar embeddings of the blocks of $E(G, C)$, the construction in the proof of Lemma 2 also shows, how to enumerate all ec-planar embeddings of $E(G, C)$ by traversing its BC-tree. In the following, we devise such a characterization. Let $B$ be a block of $E(G, C)$ and $\mathcal{T}$ its SPQR-tree.

**Observation 1.** *Every wheel gadget $\mathcal{G}$ is completely contained within the skeleton of an R-node. In particular, the hub of $\mathcal{G}$ occurs only in the skeleton of a single R-node.*

If $B$ is planar, then the skeleton of an R-node is a 3-connected planar graph, thus having exactly two planar embeddings which are mirror images of each other. We call two O-hubs contained in the same skeleton $S$ *conflicting* if none of the two planar embeddings of $S$ orients both O-hubs correctly. The following theorem gives us an easy to check condition for ec-planarity and characterizes all possible ec-planar embeddings:

**Theorem 1.** *Let $G$ be a graph with embedding constraints $C$. Let $B$ be a block of $E(G, C)$ and $\mathcal{T}$ its SPQR-tree. Then, the following holds:*

1. *$B$ is ec-planar iff $B$ is planar and no skeleton of an R-node of $\mathcal{T}$ contains conflicting O-hubs.*
2. *If $B$ is ec-planar, then the embeddings of the skeletons of $\mathcal{T}$ induce an ec-planar embedding of $B$ iff each O-hub in the skeleton of an R-node is oriented correctly.*

*Proof.* If $B$ admits an ec-planar embedding, then this embedding induces embeddings of the skeletons of $\mathcal{T}$ such that every O-hub in the skeleton of an R-node is oriented correctly. In particular, no R-node skeleton contains conflicting O-hubs.

Suppose now that $B$ is planar and no R-node skeleton contains conflicting O-hubs. For each R-node skeleton containing at least one O-hub, we can choose planar embeddings such that all O-hubs are oriented correctly within the skeletons. We have to show that the embeddings of the skeletons induce an ec-planar embedding of $B$, even if we choose arbitrary embeddings for the remaining skeletons. This holds, since every such embedding $\Pi$ has the property that each O-hub is oriented correctly, because wheel gadgets are completely contained within R-node skeletons by Observation 1 and inner wheel gadget faces are preserved. We can pick any face of $\Pi$ as external face which is not an inner wheel face (such a face always exists) and obtain an ec-planar embedding of $B$. □

Function IsEcPlanar depicted in Alg. 1 applies Theorem 1 and devises a linear time ec-planarity test, which can easily be extended so that it computes an ec-planar embedding as well.

```
1: function IsEcPlanar(Graph $G$, Constraints $C$) : bool
2:     Construct ec-expansion $E$ of $(G, C)$.
3:     if $E$ is not planar then return false
4:     for each block $B$ of $E$ do
5:         Construct SPQR-tree $\mathcal{T}$ of $B$.
6:         for each R-node $\mu \in \mathcal{T}$ do
7:             if $skeleton(\mu)$ contains two conflicting O-hubs then
8:                 return false
9:             end if
10:        end for
11:    end for
12:    return true
13: end function
```

**Algorithm 1.** Ec-planarity testing

**Theorem 2.** *Let $G = (V, E)$ be a graph with embedding constraints $C$. Then, algorithm* IsEcPlanar *tests $(G, C)$ for ec-planarity in time $O(|V|+|E|)$. Moreover, if $(G, C)$ is ec-planar, an ec-planar embedding of $(G, C)$ can also be computed in time $O(|V| + |E|)$.*

*Proof.* By Lemma 1 and 2, it is sufficient to test every block of $E(G, C)$ for ec-planarity. Hence, the correctness of Alg. 1 follows from Theorem 1.

Constructing the ec-expansion and testing planarity [14] can be done in linear time. For each block $B$ of $E(G, C)$, we construct its SPQR-tree, which requires linear time in the size of $B$; see [11]. The check for conflicting O-hubs is easy to implement: For each R-node skeleton $S$, we compute a planar embedding of $S$. If this embedding contains both correctly as well as not correctly oriented O-hubs, then there is a conflict, otherwise not. Since the total size of skeleton graphs is linear in the size of $B$ and a planar embedding can be found in linear time (see, e.g., [5]), we need linear running time for each block. Hence, the total running time is linear in the size of $E(G, C)$, which is $O(|V| + |E|)$.

In order to find an ec-planar embedding of $G$, we just have to compute embeddings of the skeleton graphs for each block as described in Theorem 1 and combine the embeddings as described in the proof of Lemma 2. $\square$

## 6 ec-Edge Insertion

We first generalize the terms insertion path and traversing costs introduced in [13]. Intuitively, the edges in an insertion path are the edges we need to cross when inserting an edge $(x, y)$ into an embedding. Let $G + (x, y)$ be a graph with embedding constraints $C$. An *ec-edge insertion path* for $x, y$ in an ec-planar embedding $\Pi$ of $G$ is a sequence of edges $e_1, \dots, e_k$ of $G$ satisfying the following conditions:

1. There is a face $f_x \in \Pi$ with $x, e_1 \in f_x$, a face $f_y \in \Pi$ with $e_k, y \in f_y$, and faces $f_i \in \Pi$ with $e_i, e_{i+1} \in f_i$ for $1 \le i < k$.

2. The edge order around $x$ and $y$ is admissible with respect to $C$ if $(x, y)$ leaves $x$ via face $f_x$ and enters $y$ via face $f_y$.

Finding a shortest ec-insertion path in a fixed embedding $\Pi$ is easy: We only need to identify the set of faces $F_x$ incident to $x$ where the insertion path may start, and $F_y$ incident to $y$ where it may end, and then find a shortest path in the dual graph of $\Pi$ connecting a face in $F_x$ with a face in $F_y$.

We are interested in the shortest possible ec-insertion path among all ec-planar embeddings of $G$, which we also call an *optimal ec-insertion path* in $G$. In particular, we need to identify the required ec-planar embedding of $G$. In order to represent all ec-planar embeddings of $G$, we apply Lemma 1 and use its ec-expansion instead. More precisely, we use the subgraph $K = E(G + (x, y), C) \setminus e$, where $e = (v, w)$ is the edge of $E(G + (x, y), C)$ connecting the expansion of $x$ with the expansion of $y$. An ec-insertion path in an ec-planar embedding of $K$ is defined as before with the only difference that we replace the second condition with

2′. $e_1, \ldots, e_k$ contains no expansion edge of $K$.

It is easy to see that we can also use this definition for a subgraph $B$ of $K$ and two distinct vertices of $B$ that are not hubs.

We adapt the notion of traversing costs defined in [13] to ec-planarity. Let $e$ be a skeleton edge, and let $\Pi$ be an arbitrary ec-embedding of the graph *expansion*$^+(e)$ (which is the expansion graph of $e$ plus the edge $e$) with dual graph $\Pi^*$, in which all edges corresponding to gadget edges have length $\infty$ and the other edges have length 1. Let $f_1$ and $f_2$ be the two faces in $\Pi$ separated by $e$. We denote with $P(\Pi^*, e)$ the length of the shortest path in $\Pi^*$ that connects $f_1$ and $f_2$ and does not use the dual edge of $e$. Hence, we have $P(\Pi, e) \in \mathbb{N} \cup \{\infty\}$.

**Lemma 3.** *Let $\mu$ be a node in $\mathcal{T}$ and let $e$ be an edge in skeleton$(\mu)$. Then, the length of the path $P(\Pi^*, e)$ is independent of the ec-embedding $\Pi$ of expansion$^+$ $(e)$.*

*Proof.* Let $m$ be the number of edges in $G_e :=$ *expansion*$^+(e)$ and $G'_e$ be the graph obtained from $G_e$ by replacing each gadget edge with $m+1$ parallel edges. Then, each embedding $\Pi$ of $G_e$ corresponds to an embedding $\Pi'$ of $G'_e$, and the length of the path $P(\Pi, e)$ is $\infty$ if and only if the corresponding path in $\Pi'$ is longer than $m$. Applying Lemma 1 in [13] and observing that the ec-embeddings of $G_e$ are a non-empty subset of the embeddings of $G_e$ yields the lemma. $\square$

Thus, we define the *ec-traversing costs* $c(e)$ of a skeleton edge $e$ as the length of the path $P(\Pi^*, e)$ for an arbitrary ec-embedding $\Pi$ of *expansion*$^+(e)$.

The hard part of the algorithm is to find an ec-insertion path in a block $B$ of $K$. Our task is to compute an optimal ec-insertion path between two nodes $v, w$ of $B$. In particular, we are not allowed to cross expansion edges of $B$. The function OptimalEcBlockInserter shown in Alg. 2 and 3 solves this problem.

It starts by computing the SPQR-tree $\mathcal{T}$ of $B$ and embeds the skeletons such that they imply an ec-embedding of $B$, i.e, the R-node skeletons are embedded

```
procedure OptimalEcBlockInserter(Block $B$ of $K$, vertex $v$, vertex $w$)
    Construct SPQR-tree $\mathcal{T}$ of $B$ such that the embeddings of the skeletons imply
    a feasible embedding of $B$.

    Find the shortest path $\mu_1, \dots, \mu_k$ in $\mathcal{T}$ between an allocation node $\mu_1$ of $v$ and
    $\mu_k$ of $w$. Root $\mathcal{T}$ such that $\mu_k$ becomes the parent of $\mu_{k-1}$ (if $k > 1$).

    $\lambda_\ell := \lambda_r := 0$          ▷ length of shortest insertion path leaving to the left/ right

    for $i = 1, \dots, k$ do
        let $S_i = skeleton(\mu_i)$

        let $G_i$ be the graph obtained from $S_i$ by replacing each edge not repre-
        senting $v$ or $w$ with its expansion graph, and let $\Pi_i$ be the embedding of
        $G_i$ induced by the embeddings of the skeletons of $\mathcal{T}$.

        if $\mu_i$ is a P-node then
            $(\phi^i_\ell, \Delta^i_\ell) := (\ell, \epsilon)$; $(\phi^i_r, \Delta^i_r) := (r, \epsilon)$          ▷ no crossings required
        else                                                              ▷ S- or R-node
            if $i = 1$ then
                $L_v := R_v :=$ the set of incident faces of the copy of $v$ in $S_i$
            else
                let $e_v$ be the representative of $v$ in $S_i$
                $L_v := \{$ the left face of $e_v\}$
                $R_v := \{$ the right face of $e_v\}$
            end if
            if $i = k$ then
                $L_w := R_w :=$ the set of incident faces of the copy of $w$ in $S_i$
            else
                let $e_w$ be the representative of $w$ in $S_i$
                $L_w := \{$ the left face of $e_w\}$
                $R_w := \{$ the right face of $e_w\}$
            end if                                              ▷ continued on next page...
```

**Algorithm 2.** Computation of an optimal ec-insertion path (2-connected case)

correctly. Then, the shortest path $\Upsilon := \mu_1, \dots, \mu_k$ between an allocation node of $v$ and of $w$ is identified. In order to achieve a consistent orientation, we root $\mathcal{T}$ such that $\Upsilon$ is a descending path in the tree, i.e., $\mu_i$ is the parent of $\mu_{i-1}$ for $i = 2, \dots, k$. Note that the rooting of the SPQR-tree implies a direction of the skeleton edges: the edges in a skeleton with reference edge $e_r = (s, t)$ are directed such that the skeleton is a planar *st*-graph; see, e.g., [7]. This direction is necessary in order to identify the left and the right face of an edge.

The algorithm traverses the path $\Upsilon$ from $\mu_1$ to $\mu_{k-1}$ and iteratively computes the lengths of the shortest ec-insertion paths that start from $v$ and leave the pertinent graph $P_i$ of $\mu_i$ to the left or to the right, respectively, where all ec-embeddings of $P_i$ are considered. Here, left and right refer to the direction of the reference edge of $\mu_i$. These lengths are maintained in the variables $\lambda_\ell$ and $\lambda_r$. Finally, when node $\mu_k$ is considered, this information is used to determine a shortest insertion path ending at $w$.

```
            ▷ Compute shortest ec-insertion paths (from l/r to l/r) within $G_i$.
            ▷ Note: $p_{\ell r} = p_{\ell\ell}$ and $p_{rr} = p_{r\ell}$ if $i \in \{1, k\}$.
            $p_{\ell r} :=$ SHORTESTECINSPATH$(\Pi_i, L_v, L_w)$
            $p_{\ell\ell} :=$ SHORTESTECINSPATH$(\Pi_i, L_v, R_w)$
            $p_{rr} :=$ SHORTESTECINSPATH$(\Pi_i, R_v, L_w)$
            $p_{r\ell} :=$ SHORTESTECINSPATH$(\Pi_i, R_v, R_w)$
            ▷ Collect possible solutions.
            $\Lambda_\ell := \{ (\lambda_\ell + |p_{\ell\ell}|, \ell, p_{\ell\ell}),\ (\lambda_r + |p_{r\ell}|, r, p_{r\ell}) \}$
            $\Lambda_r := \{ (\lambda_\ell + |p_{\ell r}|, \ell, p_{\ell r}),\ (\lambda_r + |p_{rr}|, r, p_{rr}) \}$
            if $\mu_i$ is an R-node that can be mirrored then
                $\Lambda_\ell := \Lambda_\ell \cup \{ (\lambda_\ell + |p_{rr}|, \ell, p^*_{rr}),\ (\lambda_r + |p_{\ell r}|, r, p^*_{\ell r}) \}$
                $\Lambda_r := \Lambda_r \cup \{ (\lambda_\ell + |p_{r\ell}|, \ell, p^*_{r\ell}),\ (\lambda_r + |p_{\ell\ell}|, r, p^*_{\ell\ell}) \}$
            end if
            ▷ Pick best solution.
            $(\lambda_\ell, \phi^i_\ell, \Delta^i_\ell) := \min_{1,3} \Lambda_\ell$
            $(\lambda_r, \phi^i_r, \Delta^i_r) := \min_{1,3} \Lambda_r$
        end if
    end for
    ▷ Build final ec-insertion path. Note: $\lambda_\ell = \lambda_r$ always holds here!
    $s_k := \ell$                                        ▷ Start with empty path.
    for $i := k$ downto 1 do                             ▷ Collect path backward.
        $p_i := \Delta^i_{s_i}$;  $s_{i-1} := \phi^i_{s_i}$
    end for
    return $p_1 + \cdots + p_k$
end procedure
```

**Algorithm 3.** Procedure OPTIMALECBLOCKINSERTER (part 2)

For each node $\mu_i$, the following information is computed:

– $\phi^i_\ell$ (resp. $\phi^i_r$) indicates if the shortest ec-insertion path leaving $P_i$ to the left (right) uses the shortest ec-insertion path that leaves $P_{i-1}$ to the left (in this case the value is $\ell$) or to the right (the value is $r$).
– $\Delta^i_\ell$ (resp. $\Delta^i_r$) is the subpath that is appended to the path leaving $P_{i-1}$ when leaving $P_i$ to the left (right).

These values are solely used for the purpose of creating the optimal ec-insertion path at the end of the procedure. If $s \in \{\ell, r\}$ denotes a side, we denote with $\bar{s}$ the other side, i.e., $\bar{\ell} = r$ and vice versa.

The for-loop starts by expanding all edges of the skeleton $S_i$ of $\mu_i$ except for edges representing $v$ or $w$. The resulting graph is called $G_i$. If $1 < i < k$, then $G_i$ will contain two virtual edges $e_v$ (representing $v$) and $e_w$ (representing $w$). Note that we obtain $P_i$ (plus reference edge) by replacing $e_v$ with $P_{i-1}$.

If $\mu_i$ is a P-node, then the optimal ec-insertion path leaving $P_{i-1}$ to the left (right) is also an optimal ec-insertion path leaving $P_i$ to the left (right); we just need to permute the parallel edges in $S_i$ such that $e_v$ is the leftmost (rightmost) edge. Otherwise, we have four possibilities for extending an ec-insertion path leaving $P_i$. Such a path may start in a face left or right of $e_v$, and may end in a face left or right of $e_w$. In addition, we have to consider two special cases: if $i = 1$ then $G_i$ contains $v$ and the ec-insertion path may start in any face incident to $v$; if $i = k$ then $G_i$ contains $w$ and the ec-insertion path may end in any face incident to $w$. We compute the (at most) four possible shortest ec-insertion paths using the function ShortestEcInsPath($\Pi, F_s, F_t$). Here $\Pi$ is an ec-embedding of an ec-expansion, $F_s$ are the faces where the insertion path may start, and $F_t$ are the faces where it may end. The ec-insertion path is found using BFS in the dual graph of $\Pi$, where edges corresponding to gadget edges are removed (which means that it is forbidden to cross their primal counterparts). We call these shortest ec-insertion paths $p_{\ell\ell}, p_{\ell r}, p_{r\ell}, p_{rr}$, where $p_{\ell\ell}$ stands for the path starting in a face in $L_v$ and ending in a face in $R_w$ etc. We have two choices for a shortest ec-insertion path leaving $P_i$ to the left if we consider only the given embedding of the skeleton of $\mu_i$:

– We leave $P_{i-1}$ to the left (or start at $v$ if $i = 1$) and end in a face in $R_w$ (e.g., we enter $e_w$ from right). This path has length $\lambda_\ell + |p_{\ell\ell}|$.
– We leave $P_{i-1}$ to the right (or start at $v$ if $i = 1$) and end in a face in $R_w$ (e.g., we enter $e_w$ from left). This path has length $\lambda_r + |p_{r\ell}|$.

For the shortest ec-insertion path leaving $P_i$ to the right, we have two similar cases. Further choices are possible if $\mu_i$ is an R-node that can be mirrored. We could mirror the embedding of $S_i$, expand the skeleton edges as before such that we obtain an embedding $\tilde{\Pi}_i$, and compute the four paths in $\tilde{\Pi}_i$ again. Notice that $\tilde{\Pi}_i$ is not simply the mirror image of $\Pi_i$. However, this is not necessary. We observe that, e.g., the path $\tilde{p}_{\ell\ell}$ is obtained from $p_{rr}$ by reversing the subsequences of edges that have been created by expanding a common skeleton edge of $S_i$. We call this path $p^*_{rr}$. A similar argumentation holds for $\tilde{p}_{\ell r}, \tilde{p}_{r\ell}, \tilde{p}_{rr}$. It follows that we have at most four possible choices for leaving $P_i$ to the left and to the right, respectively. Among all possible choices, we pick the shortest one.

After processing all nodes $\mu_i$, it is easy to reconstruct the best ec-insertion path from $v$ to $w$ using $\phi^i_{\ell/r}$ and $\Delta^i_{\ell/r}$. Notice that $\lambda_\ell = \lambda_r$ holds at the end, since $L^k_w = R^k_w$.

**Theorem 3.** *Let $B = (V, E)$ be a block of $K$ and let $v$ and $w$ be two distinct vertices of $B$. Then, Function* OptimalEcBlockInserter *computes an optimal ec-insertion path for $v$ and $w$ in $B$ in time $O(|E|)$.*

The edge insertion algorithm can easily be generalized to connected graphs by using the same technique as in [13] for the unconstrained edge insertion.

## 7 Conclusion and Future Work

We introduced a flexible concept of embedding constraints which allows to model a wide range of constraints on the order of incident edges. We presented a linear time algorithm for testing ec-planarity, as well as a characterization of all possible ec-embeddings. The latter is in particular important for developing algorithms that optimize over the set of all ec-planar embeddings. We showed that optimal edge insertion can still be performed in linear time when embedding constraints have to be respected. In order to devise practically successful graph drawing algorithms, the following problems should be considered:

- Incorporate the concept of embedding constraints into the planarization approach [1, 12] so that also non-ec-planar graphs can be handled. In particular, algorithms for finding ec-planar subgraphs are required; this problem can, e.g., be solved in quadratic time using successive ec-planarity testing.
- Solve the so-called *orientation problem* for orthogonal graph drawing, e.g., allow to fix some edges to attach only at the top side of a rectangular vertex.
- In some applications, only a subset of the edges is subject to embedding constraints at a vertex $v$, i.e., some edges can attach at arbitrary positions. Hence, we wish to extend the concept of embedding constraints for so-called *free edges* that are not contained in the tree $T_v$.

## References

[1] C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity relationship diagrams. *Journal of Systems and Software*, 4:163–173, 1984.

[2] K.-F. Böhringer and F. N. Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *Proc. of CHI-90*, pages 43–51, 1990.

[3] U. Brandes, M. Eiglsperger, M. Kaufmann, and D. Wagner. Sketch-driven orthogonal graph drawing. In *Proc. GD 2002*, volume 2528 of *LNCS*, pages 1–11, 2002.

[4] Ulrik Brandes and Dorothea Wagner. A bayesian paradigm for dynamic graph layout. In *Proc. GD '97*, volume 1353 of *LNCS*, pages 236–247, 1997.

[5] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *J. Computer and System Sciences*, 30:54–76, 1985.

[6] G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Drawing database schemas. *Softw. Pract. Exper.*, 32(11):1065–1098, 2002.

[7] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25(5):956–997, 1996.

[8] C. Dornheim. Planar graphs with topological constraints. *J. Graph Algorithms Appl*, 6(1):27–66, 2002.

[9] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):312–316, 1983.

[10] C. Gutwenger, K. Klein, and P. Mutzel. Planarity testing and optimal edge insertion with embedding constraints. Technical Report TR06-1-005, Chair of Algorithm Engineering, University of Dortmund, 2006.

[11] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR trees. In J. Marks, editor, *Proc. GD 2000*, volume 1984 of *LNCS*, pages 77–90, 2001.

[12] C. Gutwenger and P. Mutzel. An experimental study of crossing minimization heuristics. In G. Liotta, editor, *Proc. GD 2003*, volume 2912 of *LNCS*, pages 13–24, 2004.

[13] C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. *Algorithmica*, 41(4):289–308, 2005.

[14] J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.

[15] Stephen C. North. Incremental layout in DynaDAG. In *Proc. GD '95*, volume 1027 of *LNCS*, pages 409–418, 1996.

# Open Rectangle-of-Influence Drawings of Inner Triangulated Plane Graphs

Kazuyuki Miura[1], Tetsuya Matsuno[2], and Takao Nishizeki[3]

[1]Faculty of Symbiotic Systems Science
Fukushima University, Fukushima 960-1296, Japan
[2,3]Graduate School of Information Sciences
Tohoku University, Sendai 980-8579, Japan
miura@sss.fukushima-u.ac.jp
matsuno@nishizeki.ecei.tohoku.ac.jp
nishi@ecei.tohoku.ac.jp

**Abstract.** A straight-line drawing of a plane graph is called an open rectangle-of-influence drawing if there is no vertex in the proper inside of the axis-parallel rectangle defined by the two ends of every edge. In an inner triangulated plane graph, every inner face is a triangle although the outer face is not always a triangle. In this paper, we first obtain a sufficient condition for an inner triangulated plane graph $G$ to have an open rectangle-of-influence drawing; the condition is expressed in terms of a labeling of angles of a subgraph of $G$. We then present an $O(n^{1.5}/\log n)$-time algorithm to examine whether $G$ satisfies the condition and, if so, construct an open rectangle-of-influence drawing of $G$ on an $(n-1) \times (n-1)$ integer grid, where $n$ is the number of vertices in $G$.

## 1 Introduction

Recently automatic aesthetic drawing of graphs has created intense interest due to their broad applications, and as a consequence, a number of drawing methods have come out [1,3,4,5,6,12,13,14,15]. The most typical drawing of a plane graph $G$ is a *straight-line drawing* in which all vertices of $G$ are drawn as points and all edges are drawn as straight-line segments without any edge-intersection. A straight-line drawing is called a *grid drawing* if all vertices are put on grid points of integer coordinates. Figure 1 depicts three grid drawings of the same graph.

In this paper, we deal with a type of a grid drawing under an additional constraint, known as a "*rectangle-of-influence drawing*" [11]. A *rectangle-of-influence* of an edge $e$ is an axis-parallel rectangle having $e$ as one of its diagonals. In each of Figs. 1(a)–(c) a rectangle-of-influence is shaded for an edge $e = (u, v)$ drawn by a thick line. We call a grid drawing a *rectangle-of-influence drawing* (or simply an *RI-drawing*) if there is no vertex in a rectangle-of-influence of any edge. Figures 1(a) and (b) depict RI-drawings, while Fig. 1(c) depicts a grid drawing which is not an RI-drawing. An RI-drawing often looks pretty, since vertices tend to be separated from edges.

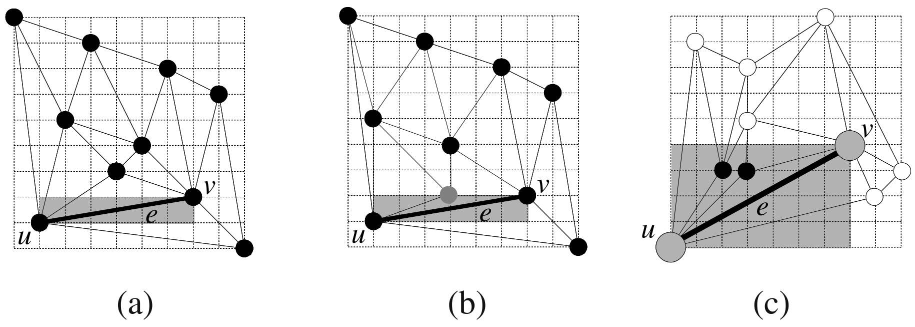


**Fig. 1.** (a) A closed RI-drawing, (b) an open RI-drawing, and (c) a non-RI-drawing of an inner triangulated plane graph without filled 3-cycles

A rectangle-of-influence of an edge $e$ is *closed* if it contains the boundary of a rectangle, and is *open* if it does not contain the boundary. In a *closed RI-drawing* every rectangle-of-influence is regarded as a closed one, while in an *open RI-drawing* every rectangle-of-influence is regarded as an open one. In a closed RI-drawing, there is no vertex except the ends not only in the proper inside of a rectangle-of-influence of each edge but also on the boundary, as illustrated in Fig. 1(a). In an open RI-drawing, there may be a vertex other than the ends on the boundary of a rectangle, as illustrated in Fig. 1(b). Thus a closed RI-drawing is an open RI-drawing, but an open RI-drawing is not always a closed RI-drawing.

Biedl *et al.* [1] showed that a plane graph $G$ has a closed RI-drawing if and only if $G$ has no filled 3-cycle, that is, a cycle of three vertices such that there is a vertex in the proper inside. They also presented a linear-time algorithm to find a closed RI-drawing of $G$ on an $(n-1) \times (n-1)$ grid if $G$ has no filled 3-cycle, where $n$ is the number of vertices in $G$. It is also known that every 4-connected plane graph with four or more vertices on the outer facial cycle has an open RI-drawing on a smaller grid, that is, a $W \times H$ grid with $W + H \leq n$, and such a drawing can be found in linear time [12], where $W$ and $H$ are the width and height of an integer grid, respectively. A plane graph $G$ may have an open RI-drawing even if $G$ has a filled 3-cycle. However, a necessary and sufficient condition for an open RI-drawing has not been known.

In a *triangulated* plane graph, all facial cycles are 3-cycles. In an *inner triangulated* plane graph, all inner facial cycles are 3-cycles although the outer facial cycle is not necessarily a 3-cycle, as illustrated in Fig. 2(a). Every plane graph can be augmented to an inner triangulated plane graph under some constraint [2].

In this paper we deal with open RI-drawings of triangulated plane graphs and inner triangulated plane graphs. We first show that one can decide in linear time whether a given triangulated plane graph $G$ has an open RI-drawing, and that if $G$ has such a drawing then it can be constructed in linear time on a $W \times H$

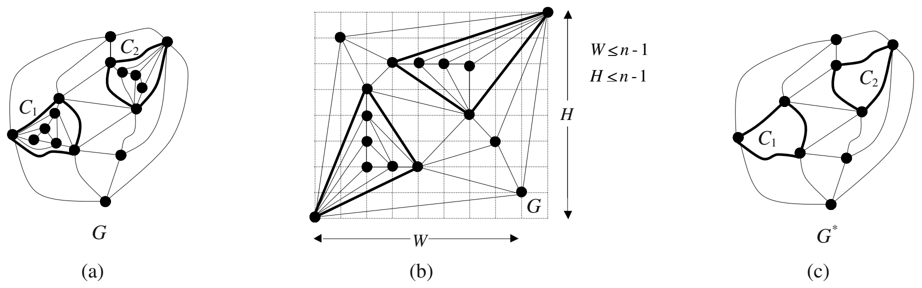


**Fig. 2.** (a) An inner triangulated plane graph $G$ with two maximal filled 3-cycles $C_1$ and $C_2$, (b) an open RI-drawing of $G$, and (c) a graph $G^*$ without filled 3-cycles

grid with $W + H = n$, where $n$ is the number of vertices in $G$. (See Fig. 3.) We then obtain a sufficient condition for an inner triangulated plane graph $G$ to have an open RI-drawing. (See Figs. 2(a) and (b).) Our condition is expressed in terms of a labeling of angles of a subgraph $G^*$ of $G$ with integers 0, 1, 2, 3 and 4, where $G^*$ is obtained from $G$ by removing all vertices and edges in the proper inside of every maximal filled 3-cycle of $G$. Figure 2(c) depicts $G^*$ for $G$ in Fig. 2(a). Note that $G^*$ is an inner triangulated plane graph. We also present an $O(n^{1.5}/\log n)$-time algorithm to examine whether $G$ satisfies the condition and, if so, construct an open RI-drawing of $G$ on an $(n-1) \times (n-1)$ grid. The complexity $O(n^{1.5}/\log n)$ is due to a step where the algorithm finds a perfect matching in a bipartite graph. It would be interesting to know if the complexity can be improved. In the case where $G$ has no filled 3-cycle, our algorithm provides a closed RI-drawing of $G$. It is an alternative algorithm to the algorithm of Biedl *et al.* [1] for the family of inner triangulated plane graphs with no filled 3-cycle.

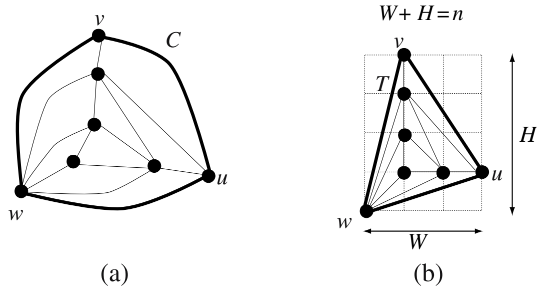


**Fig. 3.** (a) A triangulated plane graph $G$, and (b) an open RI-drawing $D$ of $G$

## 2 Drawing Triangulated Plane Graphs

Suppose that $G$ is a triangulated plane graph with four or more vertices as illustrated in Fig. 3(a), and that $G$ has an open RI-drawing $D$ as illustrated in

Fig. 3(b). The outer facial cycle $C = uvw$ of $G$ is a filled 3-cycle, and is drawn as a triangle $T$ in $D$. A straight-line segment is *oblique* if it is neither horizontal nor vertical. Two or three sides of $T$ are oblique; otherwise, $T$ has exactly one oblique side, and hence $T$ is a right-angled triangle having both a vertical side and a horizontal side; since the proper inside of such a triangle $T$ is covered by the open rectangle-of-influence of the oblique side, the inner vertices of $G$ could not be drawn. Thus there are the following three cases to consider.

(a) Two sides of $T$ are oblique and the other side is horizontal, as illustrated in Fig. 4(a);
(b) Two sides of $T$ are oblique and the other side is vertical, as illustrated in Fig. 4(b); and
(c) all the three sides of $T$ are oblique, as illustrated in Fig. 4(c).

Only the line segments in $T$ drawn by thick lines in Figs. 4(a)–(c) are not covered by the open rectangle-of-influences of three edges of $C$. Therefore, all inner vertices of $G$ must be located on the thick line segments in Figs. 4(a)–(c). Thus one can know that the graph $G$ and the drawing $D$ must have the structure illustrated in Fig. 4(f). More precisely, one of the three vertices $u, v$ and $w$ of $C$, say $w$, is adjacent to all the other vertices $z_1, z_2, \cdots, z_{n-1}$ in $G$. One may assume that $z_1 = v$, $z_{n-1} = u$, and $z_1, z_2, \cdots, z_{n-1}$ is a path in the triangulated plane graph $G$. Then, for some index $c$, $2 \le c \le n-2$, every edge of $G$, that is neither incident to $w$ nor on the path $z_1, z_2, \cdots, z_{n-1}$, joins vertices $z_i$ and $z_j$ with $1 \le i < c < j \le n-1$. The drawings in Figs. 4(d) and (e) are particular cases in which exactly two of the three outer vertices, say $v$ and $w$, are adjacent to all the other vertices in $G$ and hence $c = 2$. Note that $G = K_4$ if each of $u, v$ and $w$ is adjacent to all the other vertices in $G$.

Conversely, if $G$ has the structure above, illustrated in Fig. 4(f), then $G$ has an open RI-drawing on a $W \times H$ grid such that $W + H = n$. Note that $W = n - c$ and $H = c$ for the index $c$ above.

We thus have the following theorem.

**Theorem 1.** One can decide in linear time whether a given triangulated plane graph $G$ has an open RI-drawing or not. If $G$ has such a drawing, then it can be constructed in linear time on a $W \times H$ grid such that $W + H = n$.

## 3 Drawing Inner Triangulated Plane Graphs

In this section, we first present a sufficient condition for an inner triangulated plane graph $G$ to have an open RI-drawing, and then give an algorithm to examine whether $G$ satisfies the condition and, if so, construct an open RI-drawing of $G$. We may assume that $G$ is 2-connected.

### 3.1 Sufficient Condition

If $G$ has no filled 3-cycle, then $G$ has a closed RI-drawing [1], which is an open RI-drawing. Therefore, we may assume without loss of generality that $G$ has

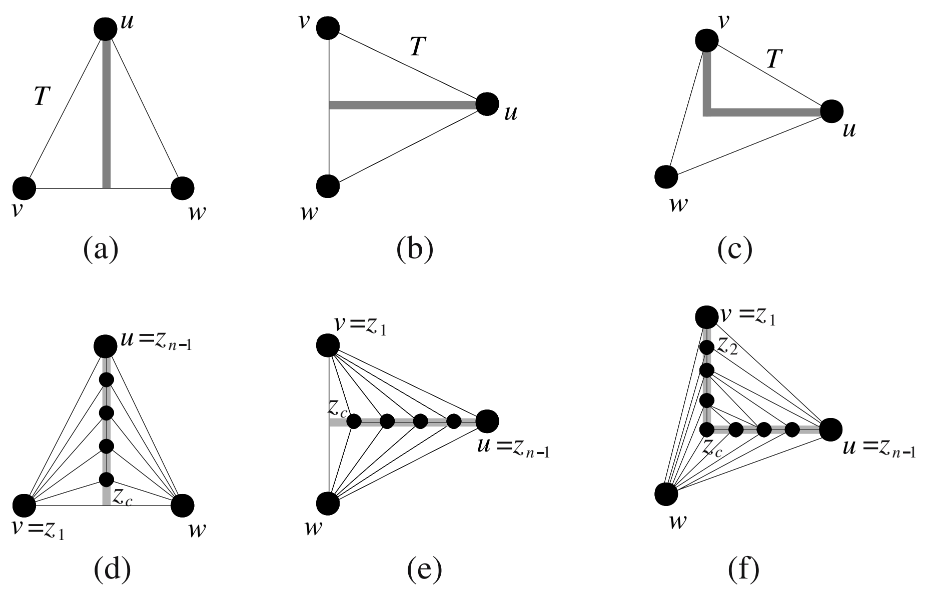


**Fig. 4.** (a)–(c) Three shapes of triangle $T$, and (d)–(f) graphs $G$ and drawings $D$

filled 3-cycles. Let $C_1, C_2, \cdots, C_k$, $k \geq 1$, be the maximal filled 3-cycles of $G$. The plane graph $G$ in Fig. 2(a) has two maximal filled 3-cycles $C_1$ and $C_2$ drawn by thick lines, and hence $k = 2$. We denote by $G(C_i)$ the *inside graph* induced by the vertices of $C_i$ and the vertices inside $C_i$. $G(C_i)$ is a triangulated plane graph. (Figure 3(a) depicts $G(C_1)$ for the graph $G$ and a maximal filled 3-cycle $C_1$ in Fig. 2(a).) One may assume without loss of generality that the inside graph $G(C_i)$ for every maximal filled 3-cycle $C_i$ has an open RI-drawing; otherwise, $G$ has no open RI-drawing.

One can transform an arbitrary open RI-drawing of $G$ in a way that every edge of $G^*$ is oblique. (The proof is omitted in this extended abstract.) Thus, one may assume without loss of generality that, in an open RI-drawing $D$ of $G$, every edge of $G^*$ is oblique, as illustrated in Fig. 2(b). A vertex on the outer facial cycle of $G^*$ is called an *outer vertex*, while a vertex not on the outer facial cycle is called an *inner vertex*. An angle of (a polygonal drawing of) a face of $G^*$ is called an *angle* of $G^*$. (See Fig. 7.) An angle of an inner face is called an *inner angle*, while an angle of the outer face is called an *outer angle*. At each vertex $v$ in $G^*$, draw two lines, one with slope 0 and one with slope $\infty$, as illustrated in Fig. 5. These two lines define four half-lines at $v$. We say that an angle at $v$ contains a number $i$ of the four half-lines, $0 \leq i \leq 4$, if the region of the plane defined by that angle contains $i$ half-lines at $v$. Thus, in Fig. 5, angles $\alpha_0, \alpha_1, \alpha_2$ and $\alpha_3$ contain 0,1,2 and 1 half-lines, respectively. In Fig. 6, the outer angles of outer vertices $v_i$, $0 \leq i \leq 4$, contains $i$ half-lines.

Our condition is expressed in terms of a labeling of $G^*$. A *labeling* $L^*$ of $G^*$ is an assignment of label 0,1,2,3 or 4 to each angle of $G^*$, as illustrated in Fig. 7(a).

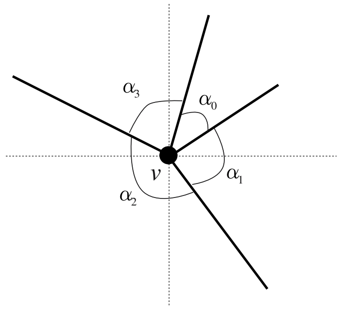


**Fig. 5.** Angles $\alpha_0, \alpha_1, \alpha_2$ and $\alpha_3$

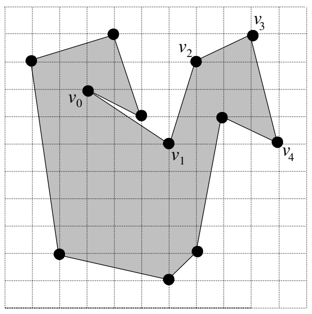


**Fig. 6.** Non-convex outer polygon

Label $i$, $0 \leq i \leq 4$, means that the angle with label $i$ contains $i$ half-lines. We say that a grid drawing $D^*$ of $G^*$ *realizes the labeling* $L^*$ if every angle labeled $i$ by $L^*$ contains $i$ half-lines in $D^*$ for each $i$, $0 \leq i \leq 4$. For a grid drawing $D^*$ of $G^*$, we denote by $L(D^*)$ the labeling of $G^*$ induced by $D^*$.

Let $D^*$ be a drawing of $G^*$ in an open RI-drawing $D$ of $G$, and let $L(D^*)$ be a labeling of $G^*$ induced by $D^*$. Clearly $L(D^*)$ satisfies the following condition:

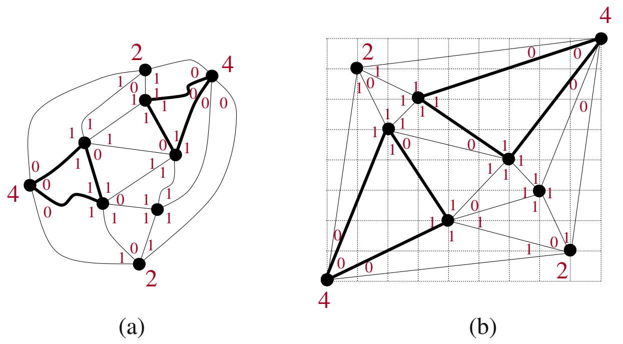


**Fig. 7.** (a) A good labeling $L^*$ of $G^*$, and (b) a good open RI-drawing $D^*$ of $G^*$ realizing $L^*$

(a) For each vertex $v$ of $G^*$, the labels around $v$ total to 4.

We now claim that $L(D^*)$ satisfies the following condition:

(b) Every inner facial 3-cycle $C$ of $G^*$ has labels $0, 1$ and $1$. If $C$ is a maximal filled 3-cycle in $G$, then the vertex labeled 0 in $C$ is adjacent to all the other vertices of the inside graph $G(C)$ of $C$; (See Fig. 8.)

Since every edge of $G^*$ is oblique in $D^*$, every inner facial 3-cycle $C$ of $G^*$ is drawn as a triangle $T$ having three oblique sides. Furthermore, two angles in $C$

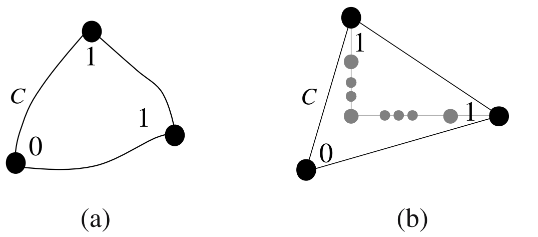


**Fig. 8.** Labelings of (a) a non-filled 3-cycle $C$ and (b) a filled 3-cycle $C$

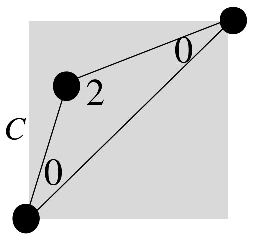


**Fig. 9.** A triangle such that an angle contain two half-lines

contain exactly one half-line and the other angle does not contain any half-line as illustrated in Fig. 8(b); if an angle in $C$ contains two half-lines, then the vertex of the angle would be in the proper inside of the rectangle-of-influence of the longest edge of $T$ as illustrated in Fig. 9. Hence $C$ has labels $0, 1$ and 1 in the labeling $L(D^*)$. If $C$ is a maximal filled 3-cycle in $G$, then $G(C)$ is a triangulated plane graph and the vertex of $C$ labeled 0 is adjacent to all the other vertices of $G(C)$ as shown in Section 2. Thus $L(D^*)$ satisfies Condition (b).

Thus it is necessary for $G$ to have an open RI-drawing that $G^*$ has a labeling satisfying Conditions (a) and (b). However, the converse is not true. We will show in Section 3.2 that $G$ has an open RI-drawing if $G^*$ has a labeling satisfying Conditions (a), (b) and the following additional condition:

(c) Every outer angle has label $2, 3$ or 4.

A labeling of $G^*$ satisfying Conditions (a)–(c) is called a *good labeling*. (The good labeling has a close relation with the regular edge-labeling of Kant and He [10].) We thus have the following theorem.

**Theorem 2.** An inner triangulated plane graph $G$ has an open RI-drawing if $G^*$ has a good labeling.

One may prefer to draw the outer facial cycle of $G$ as a convex polygon, for which each outer angle contains two, three or four half-lines. We say that an open RI-drawing $D$ of $G$ is *good* if each outer angle contains two, three or four half-lines. For example, the drawings in Figs. 1(a), 1(b), 2(b) and 3(b) are good open RI-drawings, while an open RI-drawing having the non-convex outer facial polygon in Fig. 6 is not good. It should be noted that the outer facial polygon of a good open RI-drawing is not necessary a convex polygon. Indeed our result implies that $G$ has a good open RI-drawing if and only if $G^*$ has a good labeling.

## 3.2 Computing an Open RI-Drawing from a Good Labeling

Suppose that $G^*$ has a good labeling $L^*$ as illustrated in Fig. 7(a). Remember that we assume that each triangulated plane graph $G(C_i)$ has an open

RI-drawing. We first obtain an open RI-drawing $D_i$ of each $G(C_i)$ as in Section 2. We then construct an open RI-drawing $D^*$ of $G^*$ from $L^*$, as illustrated in Fig. 7(b). We finally embed in $D^*$ each drawing $D_i$ after adjusting the size of $D_i$ to the triangular drawing of $C_i$ in $D^*$, as illustrated in Fig. 2(b). We claim that the resulting drawing is an open RI-drawing of $G$.

Our algorithm for constructing $D^*$ from $L^*$ consists of the following three steps.

**(Step 1)** Directing each edge $(u, v)$ of $G^*$, we construct a directed graph $G_x$ as illustrated in Fig. 10(a); $u \to v$ if $x(u) < x(v)$ must hold in an open RI-drawing $D^*$ of $G^*$ realizing the labeling $L^*$, where $x(u)$ and $x(v)$ are $x$-coordinates of $u$ and $v$, respectively. Similarly, we construct a directed graph $G_y$ as illustrated in Fig. 10(c). More precisely, we construct $G_x$ and $G_y$ as follows.

Let $v_1$ and $v_2$ be any two outer vertices consecutively appearing clockwise on the outer facial cycle of $G^*$. A drawing obtained from an open RI-drawing $D^*$ of $G$ by rotating it 90°, 180° or 270° is also an open RI-drawing. Therefore, one may assume without loss of generality that $x(v_1) < x(v_2)$ and $y(v_1) < y(v_2)$ in $D^*$. Let $C = v_1 v_2 v_3$ be the inner facial 3-cycle of $G^*$ having the edge $(v_1, v_2)$. Then the good labeling $L^*$ assigns label 0 to one of the vertices $v_1, v_2$ and $v_3$ of $C$ and assigns label 1 to the other two vertices. If $v_1$ has label 0, then we decide that $x(v_1) < x(v_2) < x(v_3)$ and $y(v_1) < y(v_3) < y(v_2)$ and hence $v_1 \to v_2$, $v_1 \to v_3$ and $v_2 \to v_3$ in $G_x$ and $v_1 \to v_2$, $v_1 \to v_3$ and $v_3 \to v_2$ in $G_y$, as illustrated in Fig. 11(a). If $v_2$ has label 0, then we decide that $v_1 \to v_2$, $v_1 \to v_3$ and $v_3 \to v_2$ in $G_x$ and $v_3 \to v_1$, $v_3 \to v_2$ and $v_1 \to v_2$ in $G_y$. If $v_3$ has label 0, then we decide that $v_1 \to v_2$, $v_1 \to v_3$ and $v_2 \to v_3$ in $G_x$ and $v_3 \to v_1$, $v_3 \to v_2$ and $v_1 \to v_2$ in $G_y$. Thus we direct each edge of $C$ for $G_x$ and $G_y$. We then direct the edges of each inner facial 3-cycle sharing an edge with $C$ for $G_x$ and $G_y$. Repeating the operation for each inner facial 3-cycle of $G$, we obtain a directed graph $G_x$ and $G_y$. One can show that each of $G_x$ and $G_y$ is acyclic and has exactly one vertex of in-degree zero, and every other vertex has in-degree one or more. (Condition (c) is crucial in this proof, which is omitted in this extended abstract, due to the page limitation.)

**(Step 2)** For each edge $e = u \to v$ of $G_x$, we assign an integer weight $w(e)$ to $e$. The weight $w(e)$ implies that $x(u) + w(e) \le x(v)$ in $D^*$. We decide $w(e)$ as follows. If an inner facial cycle $C$ of $G^*$ is not filled in $G$, then we give, as a weight $w(e)$, either 1 or 2 to each edge $e$ of $C$, as illustrated in Fig. 11(a). If $C$ is filled in $G$, then we assign a weight $w(e)$ to each edge $e$ of $C$, as illustrated in Fig. 11(b); the value $w(e)$ depends on both the number of vertices in $G(C)$ and the index $c$ in Section 2. Since each inner edge $e$ receives two weights from the two facial cycles containing $e$, we assign $e$ the larger one as $w(e)$.

**(Step 3)** Let $s_x$ be the *source* of $G_x$, that is, the vertex having in-degree zero. Since $G_x$ is acyclic and every vertex $u$ other than $s_x$ has in-degree one or more, one can find in linear time the longest path from $s_x$ to each vertex $u$ in $G_x$. We decide the $x$-coordinate $x(u)$ of $u$ to be the length of the longest path. Similarly, we compute the $y$-coordinate $y(u)$. Thus we obtain a drawing $D^*$ of $G^*$, as illustrated in Fig. 7(b).

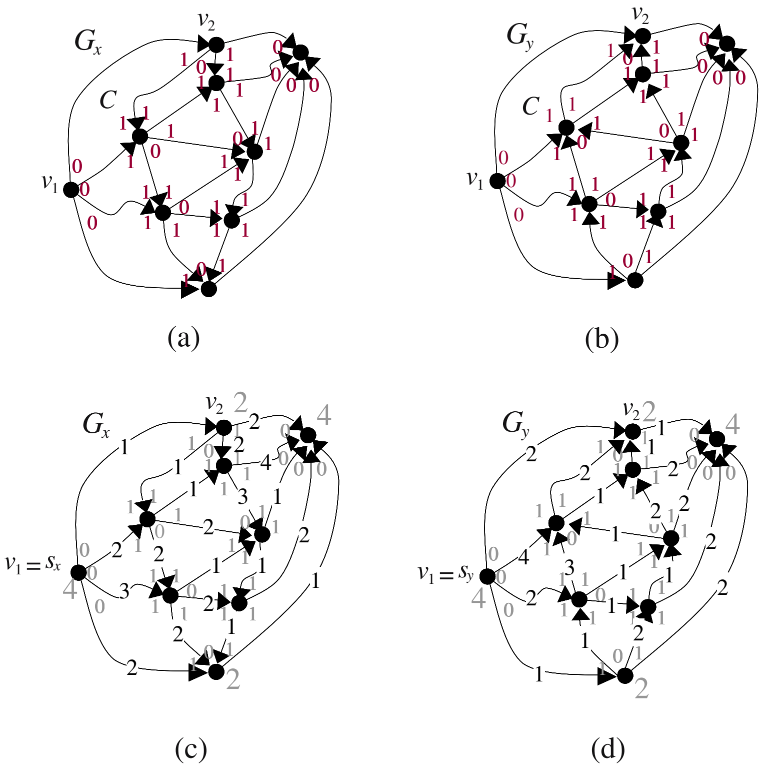


**Fig. 10.** (a) Directed graph $G_x$, (b) directed graph $G_y$, (c) weights in $G_x$, and (d) weights in $G_y$

In order to verify Theorem 2, it suffices to prove that the drawing $D^*$ realizes a given good labeling $L^*$ of $G^*$, that is, $L(D^*) = L^*$, and that the drawing $D$ obtained from $D^*$ and $D_i$ is a good open RI-drawing of $G$. The proof is omitted in this extended abstract, due to the page limitation. One can easily show that $D$ is drawn on an $(n-1) \times (n-1)$ grid.

One can construct in linear time a good open RI-drawing $D^*$ of $G^*$ from a given good labeling $L^*$ of $G^*$. Therefore, one can construct a good open RI-drawing $D$ of $G$ from $L^*$ in linear time.

## 3.3 Algorithm for Computing a Good Labeling

In this subsection we show how to find a good labeling of $G^*$.

We assign each angle of $G^*$ with label $0, 1, x$ or $y$ as illustrated in Fig. 12(a). Labels $x$ and $y$ are undecided at this moment; $x$ will be decided to be 0 or 1 and $y$ to be $2, 3$ or 4. For every inner facial 3-cycle $C$ of $G^*$ that is not filled in $G$, we assign a label $x$ to each of the three angles in $C$. For every inner facial 3-cycle $C = uvw$ of $G^*$ that is filled in $G$, we assign labels as follows: if exactly

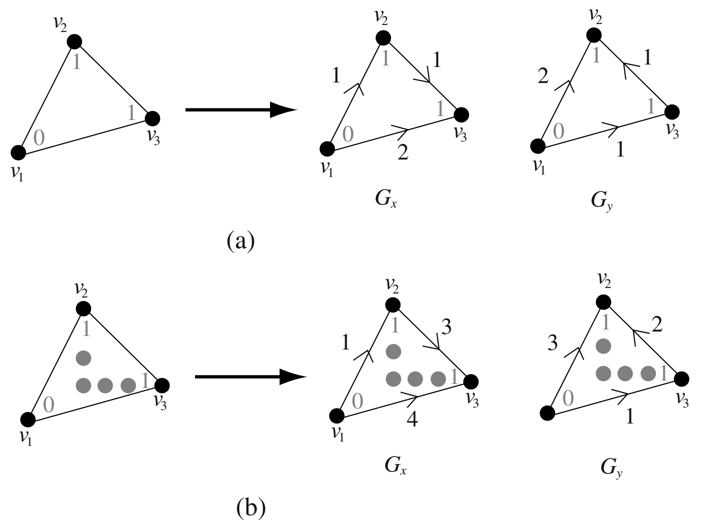


**Fig. 11.** Directions and weights $w(e)$ of edges $e$ in $G_x$ and $G_y$; (a) non-filled 3-cycle $C$, and (b) filled 3-cycle $C$

one of $u, v$ and $w$ is adjacent to all the other vertices of $G(C)$ as the case of $C_1$ in Fig. 2(a), then we assign 0 to the vertex and assign 1 to each of the other two vertices; if exactly two are adjacent to all the other vertices of $G(C)$ as the case of $C_2$ in Fig. 2(a), then we assign label $x$ to each of them and assign 1 to the other; if each of $u, v$ and $w$ is adjacent to all the other vertices of $G(C)$, then $G(C) = K_4$ and hence we assign a label $x$ to each of $u, v$ and $w$. We finally assign a label $y$ to each of the outer angles. Our problem is to determine values of all $x$'s and $y$'s so that the resulting labeling of $G^*$ satisfies Conditions (a)–(c).

Let $G_f$ be a new graph constructed from $G^*$ as illustrated in Fig. 12(b). (The detailed construction is omitted in this extended abstract.) Let $f$ be an appropriately chosen function $V(G_f) \rightarrow \{0, 1, \cdots, 4\}$; $f(v)$ is attached to each vertex $v$ in Fig. 12(b). An *$f$-factor* of $G_f$, drawn by solid lines in Fig. 12(b), is a spanning subgraph of $G_f$ in which each vertex $v$ has degree $f(v)$ [7]. We can show that $G^*$ has a good labeling, as illustrated in Fig. 12(d), if and only if $G_f$ has an $f$-factor.

Let $G_d$ be a new graph constructed from $G_f$ and $f$, as illustrated in Fig. 12(c). We can show that $G_f$ has an $f$-factor if and only if $G_d$ has a perfect matching. A perfect matching of $G_d$ is drawn by thick lines in Fig. 12(c). Since $G_d$ is a bipartite graph and has $O(n)$ vertices and edges, one can determine in time $O(n^{1.5}/\log n)$ whether $G_d$ has a perfect matching [8,9].

One can construct a good labeling of $G^*$ from an $f$-factor of $G_f$ or a perfect matching of $G_d$ in linear time. We thus have the following theorem.

**Theorem 3.** For an inner triangulated plane graph $G$, one can determine whether $G^*$ has a good labeling and, if so, compute a good labeling $L^*$ of $G^*$ in time $O(n^{1.5}/\log n)$. From $L^*$ one can construct an open RI-drawing of $G$ on an $(n-1) \times (n-1)$ grid in linear time.

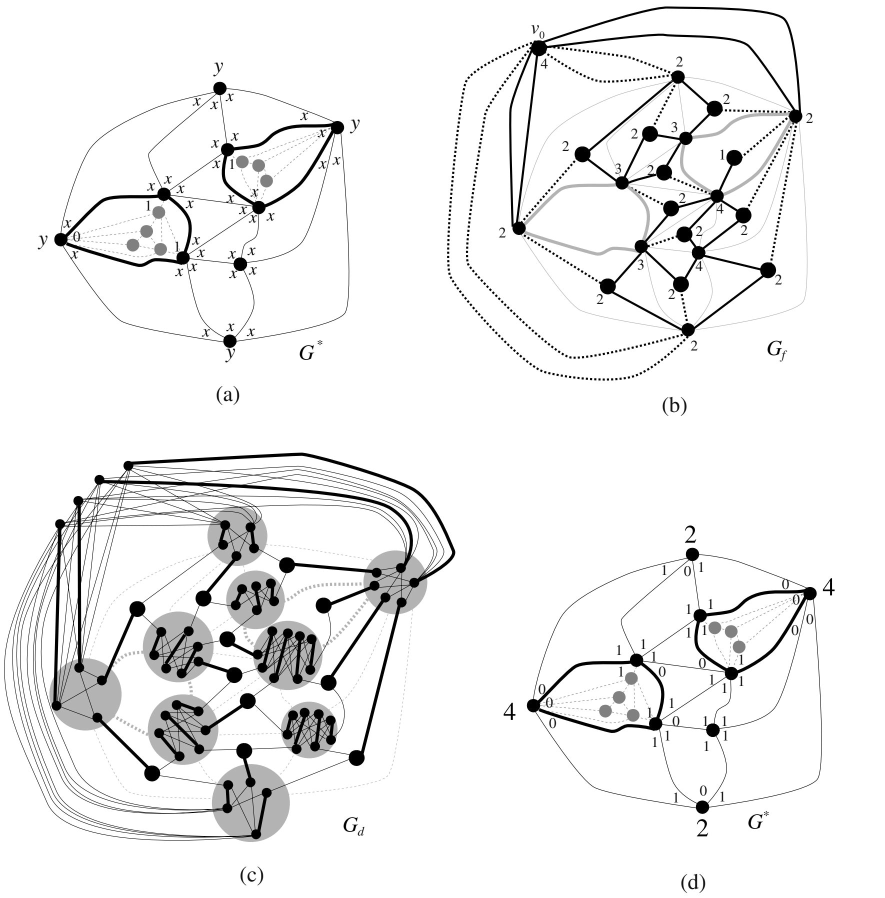


**Fig. 12.** (a) A labeling of $G^*$ by labels 0,1,$x$ and $y$, (b) an $f$-factor of $G_f$, (c) a perfect matching of a decision graph $G_d$, and (d) a good labeling of $G^*$

## References

1. T. Biedl, A. Bretscher and H. Meijer, *Rectangle of influence drawings of graphs without filled 3-cycles*, Proceedings of Graph Drawing 1999, LNCS 1731, Springer, pp. 359-368, 2000.
2. T. Biedl, G. Kant, and M. Kaufmann, *On triangulating planar graphs under the four-connectivity constraint*, Algorithmica, 19, 4, pp. 427-446, 1997.
3. M. Chrobak and G. Kant, *Convex grid darwings of 3-connected planar graphs*, Int. J. Comput. Geom Appl., 7, 3, pp. 211-223, 1997.
4. G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, Upper Saddle River, NJ 07458, 1999.

5. G. Di Battista, W. Lenhart and G. Liotta, *Proximity drawability:A survey*, Proc. Graph Drawing 1994, Lect. Note in Computer Science, LNCS, Number 894, pp. 328-339, Springer-Verlag, 1995.
6. H. de Fraysseix, J. Pach, R. Pollack, *How to draw a graph on a grid*, Combinatorica, 10, pp. 41-51, 1990.
7. F. Harary, *Graph Theory*, Addison-Wesley, Reading, Massachusetts, 1969.
8. D. S. Hochbaum, *Faster pseudoflow-based algorithms for the bipartite matching and the closure problems*, Abstract, CORS/SCRO-INFORMS Joint Int. Meeting, Banff, Canada, p. 46, May 16-19, 2004.
9. D. S. Hochbaum and B. G. Chandran, *Further below the flow decomposition barrier of maximum flow for bipartite matching and maximum closure*, Working paper, 2004.
10. G. Kant and X. He, *Regular edge-labeling of 4-connected plane graphs and its applications in graph drawing problems*, Theoretical Computer Science, 172, pp.175-193, 1997.
11. G. Liotta, A. Lubiw, H. Meijer and S. H. Whitesides, *The rectangle of influence drawability problem*, Comput. Geom. Theory and Applications, 10, 1, pp.1-22, 1998.
12. K. Miura and T. Nishizeki, *Rectangle-of-influence drawings of four-connected plane graphs*, Proc. of Information Visualisation 2005, Asia-Pacific Symposium on Information Visualisation (APVIS2005), ACS Vol 45, pp.71-76, 2005.
13. K. Miura, S. Nakano and T. Nishizeki, *Grid drawings of four-connected plane graphs*, Discrete & Computational Geometry, 26, 1, pp.73-87, 2001.
14. T. Nishizeki and Md. S. Rahman, *Planar Graph Drawing*, World Scientific, Singapore, 2004.
15. W. Schnyder, *Embedding planar graphs in the grid*, Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms, San Francisco, pp.138-147, 1990.

# Planar Decompositions and the Crossing Number of Graphs with an Excluded Minor*

David R. Wood[1,**] and Jan Arne Telle[2]

[1] Departament de Matemática Aplicada II, Universitat Politècnica de Catalunya, Barcelona, Spain
david.wood@upc.edu

[2] Department of Informatics, The University of Bergen, Bergen, Norway
Jan.Arne.Telle@ii.uib.no

**Abstract.** Tree decompositions of graphs are of fundamental importance in structural and algorithmic graph theory. Planar decompositions generalise tree decompositions by allowing an arbitrary planar graph to index the decomposition. We prove that every graph that excludes a fixed graph as a minor has a planar decomposition with bounded width and a linear number of bags.

The crossing number of a graph is the minimum number of crossings in a drawing of the graph in the plane. We prove that planar decompositions are intimately related to the crossing number, in the sense that a graph with bounded degree has linear crossing number if and only if it has a planar decomposition with bounded width and linear order. It follows from the above result about planar decompositions that every graph with bounded degree and an excluded minor has linear crossing number.

Analogous results are proved for the convex and rectilinear crossing numbers. In particular, every graph with bounded degree and bounded tree-width has linear convex crossing number, and every $K_{3,3}$-minor-free graph with bounded degree has linear rectilinear crossing number.

## 1 Introduction

The *crossing number* of a graph $G$, denoted by $\mathsf{cr}(G)$, is the minimum number of crossings in a drawing[1] of $G$ in the plane; see the survey [16]. Crossing number is an important measure of non-planarity, with applications in discrete and computational geometry, graph visualisation, and VLSI circuit design.

* The full version of this extended abstract is reference [17].

** The research of David Wood is supported by a Marie Curie Fellowship of the European Community under contract 023865, and by the projects MCYT-FEDER BFM2003-00368 and Gen. Cat 2001SGR00224.

[1] A *drawing* of a graph represents each vertex by a distinct point in the plane, and represents each edge by a simple closed curve between its endpoints, such that the only vertices an edge intersects are its own endpoints, and no three edges intersect at a common point (except at a common endpoint). A *crossing* is a point of intersection between two edges (other than a common endpoint). Undefined terminology can be found in the monographs [5, 10].

Upper bounds on the crossing number are the focus of this paper. Obviously $\mathsf{cr}(G) \leq \binom{\|G\|}{2}$ for every graph $G$, where $|G| := |V(G)|$ and $\|G\| := |E(G)|$. A graph family $\mathcal{F}$ has *linear* crossing number if for some constant $c$, every graph $G \in \mathcal{F}$ has crossing number $\mathsf{cr}(G) \leq c\,|G|$. For example, Pach and Tóth [11] proved that graphs of bounded genus and bounded degree have linear crossing number. Our main result states that bounded-degree graphs that exclude a fixed graph as a minor have linear crossing number.

**Theorem 1.** *For every graph $H$ there is a constant $c = c(H)$, such that every $H$-minor-free graph $G$ has crossing number at most $c\,\Delta(G)^2\,|G|$.*

Theorem 1 implies the above-mentioned result of Pach and Tóth [11], since graphs of bounded genus exclude a fixed graph as a minor (although the dependence on $\Delta$ is different in the two proofs; see Section 5). Moreover, there are graphs with a fixed excluded minor and unbounded genus. For other recent work on minors and crossing number see [3, 8].

Note that the assumption of bounded degree in Theorem 1 is unavoidable. For example, the complete bipartite graph $K_{3,n}$ has no $K_5$-minor, yet has $\Omega(n^2)$ crossing number [12]. Conversely, bounded degree does not by itself guarantee linear crossing number. For example, a random cubic graph on $n$ vertices has $\Omega(n)$ bisection width, which implies that it has $\Omega(n^2)$ crossing number. Also note that $c \leq \frac{20}{3}$ in Theorem 1 with $H = K_5$; see [17]. The proof of Theorem 1 is based on *planar decompositions*, which are introduced in the next section.

## 2 Graph Decompositions

Let $G$ and $D$ be graphs, such that each vertex of $D$ is a set of vertices of $G$ (called a *bag*). We allow distinct vertices of $D$ to be the same set of vertices in $G$; that is, $V(D)$ is a multiset. For each vertex $v$ of $G$, let $D(v)$ be the subgraph of $D$ induced by the bags that contain $v$. Then $D$ is a *decomposition*[2] of $G$ if:

– $D(v)$ is connected and nonempty for each vertex $v$ of $G$, and
– $D(v)$ and $D(w)$ touch[3] for each edge $vw$ of $G$.

Let $D$ be a decomposition of a graph $G$. The *width* of $D$ is the maximum cardinality of a bag. The *order* of $D$ is the number of bags. $D$ has *linear order* if its order is $\mathcal{O}(|G|)$. If the graph $D$ is a tree, then the decomposition $D$ is a *tree decomposition*. If the graph $D$ is a cycle, then the decomposition $D$ is a *cycle decomposition*. The decomposition $D$ is *planar* if the graph $D$ is planar. The *genus* of the decomposition $D$ is the genus of the graph $D$.

A decomposition $D$ of a graph $G$ is *strong* if $D(v)$ and $D(w)$ intersect for each edge $vw$ of $G$. The *tree-width* of $G$, denoted by $\mathsf{tw}(G)$, is 1 less than the minimum

[2] Decompositions, when $D$ is a tree, were introduced by Robertson and Seymour. Diestel and Kühn [6] first generalised the definition for arbitrary graphs $D$.

[3] Subgraphs $A$ and $B$ of a graph $G$ *intersect* if $V(A) \cap V(B) \neq \emptyset$, and $A$ and $B$ *touch* if they intersect or $v \in V(A)$ and $w \in V(B)$ for some edge $vw$ of $G$.

width of a strong tree decomposition of $G$. For example, a graph has tree-width 1 if and only if it is a forest. Graphs with tree-width 2 (called *series-parallel*) are planar, and are characterised as those graphs with no $K_4$-minor. Tree-width is particularly important in structural and algorithmic graph theory.

For applications to crossing number, tree decompositions are not powerful enough: even the $n \times n$ planar grid has tree-width $n$. Lemmas 10 and 11 in Section 4 prove the following theorem, which says that planar decompositions are the right type of decomposition for applications to crossing number.

**Theorem 2.** *A family of graphs with bounded degree has linear crossing number if and only if every graph in the family has a planar decomposition with bounded width and linear order.*

Every tree $T$ satisfies the Helly property: every collection of pairwise intersecting subtrees of $T$ have a vertex in common. It follows that if a tree $T$ is a strong decomposition of $G$ then every clique of $G$ is contained in some bag of $T$. Other graphs do not have this property. It will be desirable (for performing $k$-sums in Section 3) that (non-tree) decompositions have a similar property. We therefore introduce the following definitions.

For $p \geq 0$, a *$p$-clique* is a clique of cardinality $p$. A *$(\leq p)$-clique* is a clique of cardinality at most $p$. For $p \geq 2$, a decomposition $D$ of a graph $G$ is a *$p$-decomposition* if each $(\leq p)$-clique of $G$ is a subset of some bag of $D$, or is a subset of the union of two adjacent bags of $D$. An $\omega(G)$-decomposition of $G$ is called an *$\omega$-decomposition*, where $\omega(G)$ is the maximum cardinality of a clique of $G$. A $p$-decomposition $D$ of $G$ is *strong* if each $(\leq p)$-clique of $G$ is a subset of some bag of $D$. Observe that a (strong) 2-decomposition is the same as a (strong) decomposition, and a (strong) $p$-decomposition also is a (strong) $q$-decomposition for all $q \in [2, p]$.

In Section 6, we prove the following theorem, which is one of the main contributions of the paper.

**Theorem 3.** *For every graph $H$ there is an integer $k = k(H)$, such that every $H$-minor-free graph $G$ has a planar $\omega$-decomposition of width $k$ and order $|G|$.*

## 3 Manipulating Decompositions

In this section we describe four tools for manipulating graph decompositions. Our first tool describes the effect of contracting an edge in a decomposition. The elementary proof is in the full paper.

**Lemma 4 ([17]).** *Suppose that $D$ is a planar (strong) $p$-decomposition of a graph $G$ with width $k$. Say $XY$ is an edge of $D$. Then the decomposition $D'$ obtained by contracting the edge $XY$ into the vertex $X \cup Y$ is a planar (strong) $p$-decomposition of $G$ with width* $\max\{k, |X \cup Y|\}$. *In particular, if $|X \cup Y| \leq k$ then $D'$ also has width $k$.*

**Lemma 5.** *Suppose that a graph $G$ has a (strong) planar $p$-decomposition $D$ of width $k$ and order at most $c|G|$ for some $c \geq 1$. Then $G$ has a (strong) planar $p$-decomposition of width $c'k$ and order $|G|$, for some $c'$ depending only on $c$.*

*Proof.* Without loss of generality, $D$ is a planar triangulation. By a result of Biedl et al. [1], $D$ has a matching $M$ of at least $\frac{1}{3}|D|$ edges. Applying Lemma 4 to each edge of $M$, we obtain a (strong) planar $p$-decomposition of $G$ with width at most $2k$ and order at most $\frac{2}{3}|D|$. By induction, for every integer $i \geq 1$, $G$ has a (strong) planar $p$-decomposition of width $2^i k$ and order at most $(\frac{2}{3})^i|D|$. With $i := \lceil \log_{3/2} c \rceil$, the assumption that $|D| = c|G|$ implies that $G$ has a (strong) planar $p$-decomposition of width $2^i k$ and order $|G|$. □

Our second tool describes how two decompositions can be composed.

**Lemma 6.** *Suppose that $D$ is a (strong) $p$-decomposition of a graph $G$ with width $k$, and that $J$ is a decomposition of $D$ with width $\ell$. Then $G$ has a (strong) $p$-decomposition isomorphic to $J$ with width $k\ell$.*

*Proof.* Let $J'$ be the graph isomorphic to $J$ that is obtained by renaming each bag $Y \in V(J)$ by $Y' := \{v \in V(G) : v \in X \in Y \text{ for some } X \in V(D)\}$. There are at most $\ell$ vertices $X \in Y$, and at most $k$ vertices $v \in X$. Thus each bag of $J'$ has at most $k\ell$ vertices. First we prove that $J'(v)$ is connected for each vertex $v$ of $G$. Let $A'$ and $B'$ be two bags of $J'$ that contain $v$. Let $A$ and $B$ be the corresponding bags in $D$. Thus $v \in X_1$ and $v \in X_t$ for some bags $X_1, X_t \in V(D)$ such that $X_1 \in A$ and $X_t \in B$ (by the construction of $J'$). Since $D(v)$ is connected, there is a path $X_1, X_2, \dots, X_t$ in $D$ such that $v$ is in each $X_i$. In particular, each $X_i X_{i+1}$ is an edge of $D$. Now $J(X_i)$ and $J(X_{i+1})$ touch in $J$. Thus there is path in $J$ between any vertex of $J$ that contains $X_1$ and any vertex of $J$ that contains $X_t$, such that every bag in the path contains some $X_i$. In particular, there is a path $P$ in $J$ between $A$ and $B$ such that every bag in $P$ contains some $X_i$. Let $P' := \{Y' : Y \in P\}$. Then $v \in Y'$ for each bag $Y'$ of $P'$ (by the construction of $J'$). Thus $P'$ is a connected subgraph of $J'$ that includes $A'$ and $B'$, and $v$ is in every such bag. Therefore $J'(v)$ is connected. In the full paper [17] we prove that for each ($\leq p$)-clique $C$ of $G$, (a) $C$ is a subset of some bag of $J'$, or (b) $C$ is a subset of the union of two adjacent bags of $J'$.
Moreover, if $D$ is strong then case (a) always occurs. □

The third tool converts a decomposition into an $\omega$-decomposition with a small increase in the width. A graph $G$ is *$d$-degenerate* if every subgraph of $G$ has a vertex of degree at most $d$.

**Lemma 7.** *Every $d$-degenerate graph $G$ has a strong $\omega$-decomposition isomorphic to $G$ of width at most $d + 1$.*

*Proof.* It is well known (and easily proved) that $G$ has an acyclic orientation such that each vertex has indegree at most $d$. Replace each vertex $v$ by the bag $\{v\} \cup N_G^-(v)$. Every subgraph of $G$ has a *sink*. Thus every clique is a subset of some bag. The set of bags that contain a vertex $v$ are indexed by $\{v\} \cup N_G^+(v)$, which induces a connected subgraph in $G$. Thus we have a strong $\omega$-decomposition. Each bag has cardinality at most $d+1$. □

Lemmas 6 and 7 imply:

**Lemma 8.** *Suppose that $D$ is a decomposition of a $d$-degenerate graph $G$ of width $k$. Then $G$ has a strong $\omega$-decomposition isomorphic to $D$ of width $k(d+1)$.*

Our fourth tool describes how to determine a planar decomposition of a clique-sum of two graphs, given planar decompositions of the summands[4]. Let $G_1$ and $G_2$ be disjoint graphs. Suppose that $C_1$ and $C_2$ are $k$-cliques of $G_1$ and $G_2$ respectively, for some integer $k \geq 0$. Let $C_1 = \{v_1, v_2, \dots, v_k\}$ and $C_2 = \{w_1, w_2, \dots, w_k\}$. Let $G$ be a graph obtained from $G_1 \cup G_2$ by identifying $v_i$ and $w_i$ for each $i \in [1, k]$, and deleting an arbitrary (possibly empty) subset of the edges between vertices in $C_1$ $(= C_2)$. Then $G$ is a *$k$-sum* of $G_1$ and $G_2$. An $\ell$-sum for some $\ell \leq k$ is called a *$(\leq k)$-sum*. For example, if $G_1$ and $G_2$ are planar then it is easily seen that every $(\leq 2)$-sum of $G_1$ and $G_2$ is also planar.

**Lemma 9.** *Suppose that for integers $p \leq q$, a graph $G$ is a $(\leq p)$-sum of graphs $G_1$ and $G_2$, and each $G_i$ has a (strong) planar $q$-decomposition $D_i$ of width $k_i$. Then $G$ has a (strong) planar $q$-decomposition of width $\max\{k_1, k_2\}$ and order $|D_1| + |D_2|$.*

*Proof.* Let $C := V(G_1) \cap V(G_2)$. Then $C$ is a $(\leq p)$-clique, and thus a $(\leq q)$-clique, of both $G_1$ and $G_2$. Thus for each $i$, (1) $C \subseteq X_i$ for some bag $X_i$ of $D_i$, or (2) $C \subseteq X_i \cup Y_i$ for some edge $X_iY_i$ of $D_i$. If (1) is applicable, which is the case if $D_i$ is strong, then consider $Y_i := X_i$ in what follows.

Let $D$ be the graph obtained from the disjoint union of $D_1$ and $D_2$ by adding edges $X_1X_2$, $X_1Y_2$, $Y_1X_2$, and $Y_1Y_2$. By considering $X_1Y_1$ to be on the outerface of $G_1$ and $X_2Y_2$ to be on the outerface of $G_2$, observe that $D$ is planar, as illustrated in Figure 1.

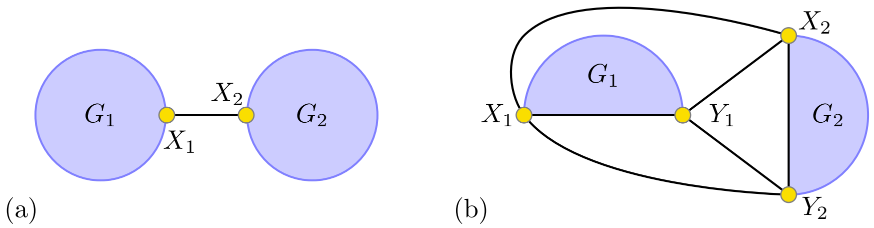


**Fig. 1.** Sum of (a) strong planar decompositions, (b) planar decompositions

We now prove that $D(v)$ is connected for each vertex $v$ of $G$. If $v \notin V(G_1)$ then $D(v) = D_2(v)$, which is connected. If $v \notin V(G_2)$ then $D(v) = D_1(v)$, which

[4] Leaños and Salazar [9] recently proved related results on the additivity of crossing numbers.

is connected. Otherwise, $v \in C$. Thus $D(v) = D_1(v) \cup D_2(v)$. Since $v \in X_1 \cup Y_1$ and $v \in X_2 \cup Y_2$, and $X_1, Y_1, X_2, Y_2$ induce a connected subgraph ($\subseteq K_4$) in $D$, we have that $D(v)$ is connected.

Each ($\leq q$)-clique $B$ of $G$ is a ($\leq q$)-clique of $G_1$ or $G_2$. Thus $B$ is a subset of some bag of $D$, or $B$ is a subset of the union of two adjacent bags of $D$. Moreover, if $D_1$ and $D_2$ are both strong, then $B$ is a subset of some bag of $D$. Therefore $D$ is a $q$-decomposition of $G$, and if $D_1$ and $D_2$ are both strong then $D$ is also strong. The width and order of $D$ are obviously as claimed. □

## 4 Planar Decompositions and the Crossing Number

The following lemma is the key link between planar decompositions and the crossing number of a graph.

**Lemma 10.** *Suppose that $D$ is a planar decomposition of a graph $G$ of width $k$. Then the crossing number of $G$ satisfies*

$$\mathsf{cr}(G) \;\leq\; 2\,\Delta(G)^2 \sum_{X \in V(D)} \binom{|X|+1}{2} \;\leq\; k(k+1)\,\Delta(G)^2\,|D| \enspace .$$

*Proof.* Fix a straight-line drawing of $D$ with no crossings. Let $\epsilon > 0$. Let $R_\epsilon(X)$ be the open disc of radius $\epsilon$ centred at each vertex $X$ in the drawing of $D$. For each edge $XY$ of $D$, let $R_\epsilon(XY)$ be the union of all segments with one endpoint in $R_\epsilon(X)$ and one endpoint in $R_\epsilon(Y)$. For some $\epsilon > 0$, $R_\epsilon(X) \cap R_\epsilon(Y) = \emptyset$ for all distinct bags $X$ and $Y$ of $D$, and $R_\epsilon(XY) \cap R_\epsilon(AB) = \emptyset$ for all edges $XY$ and $AB$ of $D$ that have no endpoint in common.

For each vertex $v$ of $G$, choose a bag $S_v$ of $D$ that contains $v$. For each vertex $v$ of $G$, choose a point $p(v) \in R_\epsilon(S_v)$, and for each bag $X$ of $D$, choose a set $P(X)$ of $\sum_{v \in X} \deg_G(v)$ points in $R_\epsilon(X)$, so that no two points coincide, no three points are collinear, and no three segments, each connecting two points, cross at a common point. These points can be chosen iteratively since each disc $R_\epsilon(X)$ is 2-dimensional[5], but the set of excluded points is 1-dimensional.

Draw each vertex $v$ at $p(v)$. For each edge $vw$ of $G$, a simple polyline $L(vw) = (p(v), x_1, x_2, \dots, x_a, y_1, y_2, \dots, y_b, p(w))$, defined by its endpoints and bends, is a *feasible* representation of $vw$ if:

(1) each bend $x_i$ is in $P(X_i)$ for some bag $X_i$ containing $v$,
(2) each bend $y_i$ is in $P(Y_i)$ for some bag $Y_i$ containing $w$,
(3) the bags $S_v, X_1, X_2, \dots, X_a, Y_1, Y_2, \dots, Y_b, S_w$ are distinct (unless $S_v = S_w$ in which case $a = b = 0$), and
(4) consecutive bends in $L(vw)$ occur in adjacent bags of $D$.

Since $D(v)$ and $D(w)$ touch, there is a feasible polyline that represents $vw$.

[5] Let $Q$ be a nonempty set of points in the plane. Then $Q$ is 2-*dimensional* if it contains a disk of positive radius; $Q$ is 1-*dimensional* if it is not 2-dimensional but contains a finite curve; otherwise $Q$ is 0-*dimensional.*

A drawing of $G$ is *feasible* if every edge of $G$ is represented by a feasible polyline, and no two bends coincide. Since each $|P(X)| = \sum_{v \in X} \deg(v)$, there is a feasible drawing. In particular, no edge passes through a vertex and no three edges have a common crossing point.

By properties (1)–(4), each segment in a feasible drawing is contained within $R_\epsilon(X)$ for some bag $X$ of $D$, or within $R_\epsilon(XY)$ for some edge $XY$ of $D$. Consider a crossing in $G$ between edges $vw$ and $xy$. Since $D$ is drawn without crossings, the crossing point is contained within $R_\epsilon(X)$ for some bag $X$ of $D$, or within $R_\epsilon(XY)$ for some edge $XY$ of $D$. Thus some endpoint of $vw$, say $v$, and some endpoint of $xy$, say $x$, are in a common bag $X$. In this case, charge the crossing to the 5-tuple $(vw, v, xy, x, X)$.

At most four crossings are charged to each 5-tuple $(vw, v, xy, x, X)$, since by property (4), each of $vw$ and $xy$ have at most two segments that intersect $R_\epsilon(X)$ (which might pairwise cross). We prove in [17] that in a feasible drawing that minimises the total (Euclidean) length of the edges (with $\{p(v) : v \in V(G)\}$ and $\{P(X) : X \in V(D)\}$ fixed), at most two crossings are charged to each such 5-tuple. Thus the number of crossings is at most twice the number of 5-tuples. Therefore the number of crossings is at most

$$2 \sum_{X \in V(D)} \sum_{v,x \in X} \deg_G(v) \cdot \deg_G(x) \ \leq\ 2\,\Delta(G)^2 \sum_{X \in V(D)} \binom{|X|+1}{2} \ . \qquad \square$$

Note that the bound on the crossing number in Lemma 10 is within a constant factor of optimal for the complete graph [17]. The following converse result to Lemma 10 is proved by replacing each crossing by a bag.

**Lemma 11 ([17]).** *Every graph $G$ has a planar decomposition of width 2 and order $|G| + \mathsf{cr}(G)$.*

## 5 Graphs Embedded in a Surface

Let $\mathbb{S}_\gamma$ be the orientable surface with $\gamma \geq 0$ handles. A *cycle* in $\mathbb{S}_\gamma$ is a closed curve in the surface. A cycle is *contractible* if it is contractible to a point in the surface. A noncontractible cycle is *separating* if it separates $\mathbb{S}_\gamma$ into two connected components.

The (*orientable*) *genus* of a graph $G$ is the minimum $\gamma$ such that $G$ has a 2-cell embedding in $\mathbb{S}_\gamma$. Let $G$ be a graph embedded in $\mathbb{S}_\gamma$. In what follows, by a *face* we mean the set of vertices on the boundary of the face. Let $F(G)$ be the set of faces in $G$. A *noose* of $G$ is a cycle $C$ in $\mathbb{S}_\gamma$ that does not intersect the interior of an edge of $G$. Let $V(C)$ be the set of vertices of $G$ intersected by $C$. The *length* of $C$ is $|V(C)|$.

Pach and Tóth [11] proved that, for some constant $c_\gamma$, the crossing number of every graph $G$ of genus $\gamma$ satisfies

$$\mathsf{cr}(G) \ \leq\ c_\gamma \sum_{v \in V(G)} \deg(v)^2 \ \leq\ 2c_\gamma\, \Delta(G)\, \|G\| \ . \qquad (1)$$

The constant $c_\gamma$ was subsequently improved by Djidjev and Vrťo [7]. It is well known [17] that $\|G\| \leq (\sqrt{3\gamma}+3)|G|-6$. Thus

$$\mathsf{cr}(G) \ \leq \ c_\gamma \, \Delta(G) \, |G| \ . \tag{2}$$

By Lemma 11, $G$ has a planar decomposition of width 2 and order $c_\gamma \, \Delta(G) \, |G|$. We now provide an analogous result without the dependence on $\Delta(G)$, but at the expense of an increased bound on the width.

**Theorem 12.** *Every graph $G$ with genus $\gamma$ has a planar decomposition of width $2^\gamma$ and order $3^\gamma|G|$.*

The key to the proof of Theorem 12 is the following lemma, whose proof is inspired by similar ideas of Pach and Tóth [11].

**Lemma 13.** *Let $G$ be a graph with a 2-cell embedding in $\mathbb{S}_\gamma$ for some $\gamma \geq 1$. Then $G$ has a decomposition of width 2, genus at most $\gamma-1$, and order $3|G|$.*

*Proof.* Since $\gamma \geq 1$, $G$ has a noncontractible nonseparating noose. Let $C$ be a noncontractible nonseparating noose of minimum length $k := |V(C)|$. Orient $C$ and let $V(C) := (v_1, v_2, \dots, v_k)$ in the order around $C$. For each vertex $v_i \in V(C)$, let $E^\ell(v_i)$ and $E^r(v_i)$ respectively be the set of edges incident to $v_i$ that are on the left-hand side and right-hand side of $C$ (with respect to the orientation). Cut the surface along $C$, and attach a disk to each side of the cut. Replace each vertex $v_i \in V(C)$ by two vertices $v_i^\ell$ and $v_i^r$ respectively incident to the edges in $E^\ell(v_i)$ and $E^r(v_i)$. Embed $v_i^\ell$ on the left-hand side of the cut, and embed $v^r$ on the right-hand side of the cut. We obtain a graph $G'$ embedded in a surface of genus at most $\gamma-1$ (since $C$ is nonseparating).

Let $L := \{v_i^\ell : v \in V(C)\}$ and $R := \{v_i^r : v \in V(C)\}$. By Menger's Theorem, the maximum number of disjoint paths between $L$ and $R$ in $G'$ equals the minimum number of vertices that separate $L$ from $R$ in $G'$. Let $Q$ be a minimum set of vertices that separate $L$ from $R$ in $G'$. Then there is a noncontractible nonseparating noose in $G$ that only intersects vertices in $Q$. (It is nonseparating in $G$ since $L$ and $R$ are identified in $G$.) Thus $|Q| \geq k$ by the minimality of $|V(C)|$. Hence there exist $k$ disjoint paths $P_1, P_2, \dots, P_k$ between $L$ and $R$ in $G'$, where the endpoints of $P_i$ are $v_i^\ell$ and $v_{\sigma(i)}^r$, for some permutation $\sigma$ of $[1,k]$. In the disc with $R$ on its boundary, draw an edge from each vertex $v_{\sigma(i)}^r$ to $v_i^r$ such that no three edges cross at a single point and every pair of edge cross at most once. Add a new vertex $x_{i,j}$ on each crossing point between edges $v_{\sigma(i)}^r v_i^r$ and $v_{\sigma(j)}^r v_j^r$. Let $G''$ be the graph obtained. Then $G''$ is embedded in $S_{\gamma-1}$.

We now make $G''$ a decomposition of $G$. Replace $v_i^\ell$ by $\{v_i\}$ and replace $v_i^r$ by $\{v_i\}$. Replace every other vertex $v$ of $G$ by $\{v\}$. Replace each 'crossing' vertex $x_{i,j}$ by $\{v_i, v_j\}$. Now for each vertex $v_i \in V(C)$, add $v_i$ to each bag on the path $P_i$ from $v_i^\ell$ to $v_{\sigma(i)}^r$. Thus $G''(v_i)$ is a (connected) path. Clearly $G''(v)$ and $G''(w)$ touch for each edge $vw$ of $G$. Hence $G''$ is a decomposition of $G$ with genus at most $\gamma-1$. Since the paths $P_1, P_2, \dots, P_k$ are pairwise disjoint, the width of the decomposition is 2.

It remains to bound the order of $G''$. Let $n := |G|$. Observe that $G''$ has at most $n+k+\binom{k}{2}$ vertices. One of the paths $P_i$ has at most $\frac{n+k}{k}$ vertices. For ease of counting, add a cycle to $G'$ around $R$. Consider the path in $G'$ that starts at $v_i^\ell$, passes through each vertex in $P_i$, and then takes the shortest route from $v^r_{\sigma(i)}$ around $R$ back to $v_i^r$. The distance between $v^r_{\sigma(i)}$ and $v_i^r$ around $R$ is at most $\frac{k}{2}$. This path in $G'$ forms a noncontractible nonseparating noose in $G$ (since if two cycles in a surface cross in exactly one point, then both are noncontractible).

The length of this noose in $G$ is at most $\frac{n+k}{k} - 1 + \frac{k}{2}$ (since $v_i^\ell$ and $v_i^r$ both appeared in the path). Hence $\frac{n+k}{k} - 1 + \frac{k}{2} \geq k$ by the minimality of $|V(C)|$. Thus $k \leq \sqrt{2n}$. Therefore $G''$ has at most $n + \sqrt{2n} + \binom{\sqrt{2n}}{2} \leq 3n$ vertices. □

*Proof of Theorem 12.* We proceed by induction on $\gamma$. If $\gamma = 0$ then $G$ is planar, and $G$ itself is a planar decomposition of width $1 = 2^0$ and order $n = 3^0 n$. Otherwise, by Lemma 13, $G$ has a decomposition $D$ of width 2, genus $\gamma - 1$, and order $3n$. By induction, $D$ has a planar decomposition of width $2^{\gamma-1}$ and order $3^{\gamma-1}(3n) = 3^\gamma n$. By Lemma 6 with $p = k = 2$, and $\ell = 2^{\gamma-1}$, $G$ has a planar decomposition of width $2 \cdot 2^{\gamma-1} = 2^\gamma$ and order $3^\gamma n$. □

Theorem 12 and Lemma 10 imply that every graph $G$ with genus $\gamma$ has crossing number $\mathsf{cr}(G) \leq 12^\gamma \,\Delta(G)^2\, |G|$, which for fixed $\gamma$, is weaker than the bound of Pach and Tóth [11] in (2). The advantage of our approach is that it generalises for graphs with an arbitrary excluded minor.

## 6 *H*-Minor-Free Graphs

For integers $h \geq 1$ and $\gamma \geq 0$, Robertson and Seymour [13] defined a graph $G$ to be *h-almost embeddable* in $\mathbb{S}_\gamma$ if $G$ has a set $X$ of at most $h$ vertices such that $G \setminus X$ can be written as $G_0 \cup G_1 \cup \cdots \cup G_h$ such that:

- $G_0$ has an embedding in $\mathbb{S}_\gamma$,
- the graphs $G_1, G_2, \dots, G_h$ (called *vortices*) are pairwise disjoint,
- there are faces $F_1, F_2, \dots, F_h$ of the embedding of $G_0$ in $\mathbb{S}_\gamma$, such that each $F_i = V(G_0) \cap V(G_i)$,
- if $F_i = (u_{i,1}, u_{i,2}, \dots, u_{i,|F_i|})$ in clockwise order about the face, then $G_i$ has a strong $|F_i|$-cycle decomposition $Q_i$ of width $h$, such that each vertex $u_{i,j}$ is in the $j$-th bag of $Q_i$.

The following 'characterisation' of $H$-minor-free graphs is a deep theorem by Robertson and Seymour [13].

**Theorem 14 ([13]).** *For every graph $H$ there is a positive integer $h = h(H)$, such that every $H$-minor-free graph $G$ can be obtained by $(\leq h)$-sums of graphs that are $h$-almost embeddable in some surface in which $H$ cannot be embedded.*

**Lemma 15.** *Every graph $G$ that is $h$-almost embeddable in $\mathbb{S}_\gamma$ has a planar decomposition of width $h(2^\gamma + 1)$ and order $3^\gamma\,|G|$.*

*Proof.* By Theorem 12, $G_0$ has a planar decomposition $D$ of width at most $2^\gamma$ and order $3^\gamma\,|G_0| \leq 3^\gamma\,|G|$. We can assume that $D$ is connected. For each vortex $G_i$, add each vertex in the $j$-th bag of $Q_i$ to each bag of $D$ that contains $u_{i,j}$. The bags of $D$ now contain at most $2^\gamma h$ vertices. Now add $X$ to every bag. The bags of $D$ now contain at most $(2^\gamma+1)h$ vertices. For each vertex $v$ that is not in a vortex, $D(v)$ is unchanged by the addition of the vortices, and is thus connected. For each vertex $v$ in a vortex $G_i$, $D(v)$ is the subgraph of $D$ induced by the bags (in the decomposition of $G_0$) that contain $u_{i,j}$, where $v$ is in the $j$-th bag of $Q_i$. Now $Q_i(v)$ is a connected subgraph of the cycle $Q_i$, and for each vertex $u_{i,j}$, the subgraphs $G_0(u_{i,j})$ and $G_0(u_{i,j+1})$ touch. Thus $D(v)$ is connected. (This argument is similar to that used in Lemma 6.) $D(v)$ is connected for each vertex $v \in X$ since $D$ itself is connected. $\square$

**Lemma 16.** *For all integers $h \geq 1$ and $\gamma \geq 0$ there is a constant $d = d(h,\gamma)$, such that every graph $G$ that is $h$-almost embeddable in $\mathbb{S}_\gamma$ is $d$-degenerate.*

*Proof.* If $G$ is $h$-almost embeddable in $\mathbb{S}_\gamma$ then every subgraph of $G$ is $h$-almost embeddable in $\mathbb{S}_\gamma$. Thus it suffices to prove that if $G$ has $n$ vertices and $m$ edges, then its average degree $\frac{2m}{n} \leq d$. Say each $G_i$ has $m_i$ edges. $G$ has at most $hn$ edges incident to $X$. Thus $m \leq hn + \sum_{i=0}^{h} m_i$. Now $m_0 < (\sqrt{3\gamma}+3)n$. Both endpoints of an edge of a vortex $G_i$ is in some bag of $Q_i$. Thus $m_i \leq \binom{h}{2}|F_i|$. Since $G_1, G_2, \dots, G_h$ are pairwise disjoint, $\sum_{i=1}^{h} m_i \leq \binom{h}{2}n$. Thus $m < (h + \sqrt{3\gamma} + 3 + \binom{h}{2})n$. Taking $d = h(h+1) + 2\sqrt{3\gamma} + 6$ we are done. $\square$

Lemmas 8, 15 and 16 imply:

**Corollary 17.** *For all integers $h \geq 1$ and $\gamma \geq 0$ there is a constant $k = k(h,\gamma) \geq \gamma$, such that every graph $G$ that is $h$-almost embeddable in $\mathbb{S}_\gamma$ has a planar $\omega$-decomposition of width $k$ and order $3^\gamma\,|G|$.* $\square$

Now we bring in $(\leq h)$-sums.

**Lemma 18.** *For all integers $h \geq 1$ and $\gamma \geq 0$, every graph $G$ that can be obtained by $(\leq h)$-sums of graphs that are $h$-almost embeddable in $\mathbb{S}_\gamma$ has a planar $\omega$-decomposition of width $k$ and order* $\max\{1, 3^\gamma(h+1)(|G|-h)\}$, *where $k = k(h,\gamma)$ from Corollary 17.*

*Proof.* If $|G| \leq h$ then the decomposition of $G$ with all its vertices in a single bag satisfies the claim (since $k \geq h$). Now assume that $|G| \geq h+1$. If $G$ is $h$-almost embeddable in $\mathbb{S}_\gamma$, then by Corollary 17, $G$ has a planar $\omega$-decomposition of width $k$ and order $3^\gamma|G| \leq 3^\gamma(h+1)(|G|-h)$. Otherwise, $G$ is a $(\leq h)$-sum of graphs $G_1$ and $G_2$, each of which, by induction, has a planar $\omega$-decomposition of width $k$ and order $\max\{1, 3^\gamma(h+1)(|G_i|-h)\}$. By Lemma 9, $G$ has a planar $\omega$-decomposition $D$ of width $k$ and order $|D| = \max\{1, 3^\gamma(h+1)(|G_1|-h)\} + \max\{1, 3^\gamma(h+1)(|G_2|-h)\}$. Without loss of generality, $|G_1| \leq |G_2|$. If $|G_2| \leq h$ then $|D| = 2 \leq 3^\gamma(h+1)(|G|-h)$. If $|G_1| \leq h$ and $|G_2| \geq h+1$, then $|D| = 1 + 3^\gamma(h+1)(|G_2|-h) \leq 3^\gamma(h+1)(|G|-h)$. Otherwise, both $|G_1| \geq h+1$ and $|G_2| \geq h+1$. Thus $|D| \leq 3^\gamma(h+1)(|G_1|+|G_2|-2h) \leq 3^\gamma(h+1)(|G|-h)$. $\square$

*Proof of Theorem 3.* Let $h = h(H)$ from Theorem 14. Let $\mathbb{S}_\gamma$ be the surface in Theorem 14 in which $H$ cannot be embedded. By Theorem 14, $G$ can be obtained by $(\leq h)$-sums of graphs that are $h$-almost embeddable in $\mathbb{S}_\gamma$. By Lemma 18, $G$ has a planar $\omega$-decomposition of width $k$ and order $3^\gamma(h+1)|G|$, where $k = k(h,\gamma)$ from Corollary 17. By Lemma 5, $G$ has a planar $\omega$-decomposition of width $k'$ and order $|G|$, for some $k'$ only depending on $k$, $\gamma$ and $h$. □

Observe that Lemma 10 and Theorem 3 prove Theorem 1.

## 7 Complementary Results

A graph drawing is *rectilinear* (or *geometric*) if each edge is represented by a straight line-segment. The *rectilinear crossing number* of a graph $G$, denoted by $\overline{\mathsf{cr}}(G)$, is the minimum number of crossings in a rectilinear drawing of $G$; see [2, 14]. A rectilinear drawing is *convex* if the vertices are positioned on a circle. The *convex* (or *outerplanar*) *crossing number* of a graph $G$, denoted by $\mathsf{cr}^\star(G)$, is the minimum number of crossings in a convex drawing of $G$; see [4, 15]. Obviously $\mathsf{cr}(G) \leq \overline{\mathsf{cr}}(G) \leq \mathsf{cr}^\star(G)$ for every graph $G$. *Linear* rectilinear and *linear* convex crossing numbers are defined in an analogous way to linear crossing number.

It is unknown whether an analogue of Theorem 1 holds for rectilinear crossing number[6]. On the other hand, we prove such a result for $K_{3,3}$-minor-free graphs.

**Theorem 19 ([17]).** *Every $K_{3,3}$-minor-free graph $G$ has a rectilinear drawing in which each edge crosses at most $2\,\Delta(G)$ other edges. Hence $\overline{\mathsf{cr}}(G) \leq \Delta(G)\,\|G\| \leq \Delta(G)\,(3\,|G| - 5)$.*

An analogue of Theorem 1 for convex crossing number does not hold, even for planar graphs, since Shahrokhi et al. [15] proved that the $n \times n$ planar grid $G_n$ (which has maximum degree 4) has convex crossing number $\Omega(|G_n| \log |G_n|)$. Now, $G_n$ has tree-width $n$. In the following sense, we prove that large tree-width necessarily forces up the convex crossing number.

**Theorem 20 ([17]).** *Every graph $G$ with degree at most $\Delta$ and tree-width at most $k$ has a convex drawing in which each edge crosses $\mathcal{O}(k\Delta^2)$ other edges. Hence $\mathsf{cr}^\star(G) \leq \mathcal{O}(k\Delta^2\|G\|) \leq \mathcal{O}(k^2\Delta^2|G|)$. Conversely, suppose that a graph $G$ has a convex drawing such that whenever two edges $e$ and $f$ cross, $e$ or $f$ crosses at most $\ell$ edges. Then $G$ has tree-width at most $3\ell + 11$.*

In particular, graphs of bounded degree and bounded tree-width have linear convex crossing number. Again, the assumption of bounded degree is necessary since $K_{3,n}$ has tree-width 3 and crossing number $\Omega(n^2)$.

[6] The crossing number and rectilinear crossing number are not related in general. In particular, for every integer $k \geq 4$, Bienstock and Dean [2] constructed a graph $G_k$ with crossing number 4 and rectilinear crossing number $k$. It is easily seen that $G_k$ has no $K_{14}$-minor. However, the maximum degree of $G_k$ increases with $k$. Thus $G_k$ is not a counterexample to an analogue of Theorem 1 for rectilinear crossing number.

**Acknowledgements.** Thanks to János Pach and Géza Tóth for explaining their proofs in reference [11].

## References

[1] Therese Biedl, Erik D. Demaine, Christian A. Duncan, Rudolf Fleischer, and Stephen G. Kobourov. Tight bounds on maximal and maximum matchings. *Discrete Math.*, 285(1-3):7–15, 2004.

[2] Daniel Bienstock and Nathaniel Dean. Bounds for rectilinear crossing numbers. *J. Graph Theory*, 17(3):333–348, 1993.

[3] Drago Bokal, Gašper Fijavž, and Bojan Mohar. The minor crossing number. *SIAM J. Discrete Math.*, 20(2):344–356, 2006.

[4] Éva Czabarka, Ondrej Sýkora, László A. Székely, and Imrich Vrt'o. Outerplanar crossing numbers, the circular arrangement problem and isoperimetric functions. *Electron. J. Combin.*, 11(1):R81, 2004.

[5] Reinhard Diestel. *Graph theory*, vol. 173 of *Graduate Texts in Mathematics.* Springer, 2nd edn., 2000.

[6] Reinhard Diestel and Daniela Kühn. Graph minor hierarchies. *Discrete Appl. Math.*, 145(2):167–182, 2005.

[7] Hristo N. Djidjev and Imrich Vrťo. Planar crossing numbers of genus $g$ graphs. In *Proc. 33rd International Colloquium on Automata, Languages and Programming* (ICALP '06), vol. 4051 of *Lecture Notes in Comput. Sci.*, pp. 419–430. Springer, 2006.

[8] Enrique Garcia-Moreno and Gelasio Salazar. Bounding the crossing number of a graph in terms of the crossing number of a minor with small maximum degree. *J. Graph Theory*, 36(3):168–173, 2001.

[9] Jesús Leaños and Gelasio Salazar. On the additivity of crossing numbers of graphs, 2006.
`http://www.ifisica.uaslp.mx/~gsalazar/RESEARCH/additivity.pdf`.

[10] Bojan Mohar and Carsten Thomassen. *Graphs on surfaces.* Johns Hopkins University Press, Baltimore, U.S.A., 2001.

[11] János Pach and Géza Tóth. Crossing number of toroidal graphs. *Proc. 13th International Symp. on Graph Drawing* (GD '05), vol. 3843 of *Lecture Notes in Comput. Sci.*, pp. 334–342. Springer, 2006.

[12] R. Bruce Richter and Jozef Širáň. The crossing number of $K_{3,n}$ in a surface. *J. Graph Theory*, 21(1):51–54, 1996.

[13] Neil Robertson and Paul D. Seymour. Graph minors. XVI. Excluding a non-planar graph. *J. Combin. Theory Ser. B*, 89(1):43–76, 2003.

[14] Edward R. Scheinerman and Herbert S. Wilf. The rectilinear crossing number of a complete graph and Sylvester's "four point problem" of geometric probability. *Amer. Math. Monthly*, 101(10):939–943, 1994.

[15] Farhad Shahrokhi, Ondrej Sýkora, Laszlo A. Székely, and Imrich Vrt′o. The gap between crossing numbers and convex crossing numbers. *Towards a Theory of Geometric Graphs*, vol. 342 of *Contemporary Mathematics*, pp. 249–258. Amer. Math. Soc., 2004.

[16] László A. Székely. A successful concept for measuring non-planarity of graphs: the crossing number. *Discrete Math.*, 276(1-3):331–352, 2004.

[17] David R. Wood and Jan Arne Telle. Planar decompositions and the crossing number of graphs with an excluded minor. 2006.
`http://www.arxiv.org/math/0604467`.

# On the Crossing Number of Almost Planar Graphs

Petr Hliněný[1,⋆] and Gelasio Salazar[2,⋆⋆]

[1] Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
hlineny@fi.muni.cz
[2] Instituto de Física, Universidad Autónoma de San Luis Potosí
San Luis Potosí, Mexico, 78000
gsalazar@ifisica.uaslp.mx

**Abstract.** Crossing minimization is one of the most challenging algorithmic problems in topological graph theory, with strong ties to graph drawing applications. Despite a long history of intensive research, no practical "good" algorithm for crossing minimization is known (that is hardly surprising, since the problem itself is NP-complete). Even more surprising is how little we know about a seemingly simple particular problem: to minimize the number of crossings in an *almost planar* graph, that is, a graph with an edge whose removal leaves a planar graph. This problem is in turn a building block in an "edge insertion" heuristic for crossing minimization. In this paper we prove a constant factor approximation algorithm for the crossing number of almost planar graphs with bounded degree. On the other hand, we demonstrate nontriviality of the crossing minimization problem on almost planar graphs by exhibiting several examples, among them new families of crossing critical graphs which are almost planar and projective.

**Keywords:** crossing number, crossing minimization, planarization, crossing-critical graphs.

**2000 Math Subject Classification:** 05C10, 05C62, 68R10.

## 1 Introduction

We assume that the reader is familiar with the standard notation of terminology of graph theory, and especially with topological graphs, see [11]. In this paper we consider finite graphs, with multiple edges allowed.

The *crossing number* $\mathrm{cr}(G)$ of a graph $G$ is the minimum number of pairwise edge crossings in a drawing of $G$ in the plane (thus, a graph is planar if and only if its crossing number is 0). A drawing of $G$ with $\mathrm{cr}(G)$ crossings is *(crossing-) optimal.* Crossing number problems were introduced by Turán, whose work in a brick factory during the Second World War led him to inquire about the

⋆ Supported in part by the Institute of Theoretical Computer Science, project 1M0545.
⋆⋆ Supported by CONACYT Grant 45903.

crossing number of the complete bipartite graphs $K_{m,n}$. It is remarkable that this long-standing particular question is still open. Not surprisingly, exact crossing numbers are in general very difficult to compute.

Nowadays, computing crossing numbers has important applications in VLSI design, and, naturally, in graph drawing. The algorithmic problem of *crossing minimization* is given as follows:

*Input:* A (multi)graph $G$ and an integer $k$.
*Question:* Is $\mathrm{cr}(G) \le k$ ? (Possibly: if so, find an optimal drawing.)

The problem is in NP since one could guess the optimal drawing, replace the crossings in it with new (degree 4, subdividing) vertices, and verify planarity of the resulting graph. It has been proved by Garey and Johnson [4] that crossing minimization is NP-complete if $k$ is a part of the input. The same assertion has been proved true later by the first author [10] both for cubic graphs and for the minor-monotone version (cf. [1]) of crossing number. An important, stubborn open problem is to determine whether the crossing number of graphs with bounded tree-width can be computed in polynomial time.

On the positive side, a surprising result from Grohe states that the crossing number is an FPT parameter.

**Theorem 1.1 (Grohe [6]).** *One can decide whether* $\mathrm{cr}(G) \le k$ *for an* $n$*-vertex graph* $G$ *in time* $O\big(f(k) \cdot n^2\big)$.

Grohe's algorithm is the only efficient algorithm (given fixed $k$) known so far for computing exact crossing numbers. Unfortunately, this algorithm is not usable in practice, not even for relatively small values of $k$, since $f$ is double exponential in $k$ and, moreover, the "hidden constants" are very large.

Regarding approximability results, the best result known to date is a (polynomial time) $\log^3 n$ approximation algorithm by Even, Guha and Schieber [3]. Constant factor approximation algorithms are known only for particular families of graphs, such as projective graphs with bounded degree [5].

Our paper brings two new main results to the theory of crossing numbers of almost planar graphs: First, Theorem 2.2 proves that a known heuristic for crossing minimization of an almost planar graph $G + e$ — take a suitable planar embedding of $G$ and insert $e$ to it — is a provably good approximation on graphs of bounded degrees. Second (on the negative side), Theorem 3.4 brings new rich families of $k$-crossing-critical graphs which are both almost planar and projective, that is, as close to planarity as one can reasonably imagine.

## 2 Approximating the Crossing Number of Bounded–Degree Almost Planar Graphs

Currently, it seems that the best known general–purpose practical heuristic approach to crossing minimization on a graph $G$ is the following: First, delete from $G$ some (small set of) edges $F$, so that $G' = G - F$ is planar. Then, take an edge $f \in F$ and a suitable planar embedding of $G'$, and find a way of inserting $f$ back to the drawing of $G'$ with the smallest number of crossings (using a shortest-path algorithm on the topological dual of $G'$). After that, create a new graph $G''$ from $G' + f$ by replacing the crossings with new vertices, and iterate the process with $G''$ and $F \setminus \{f\}$.

This heuristic algorithm outlines the following interesting subproblem on almost-planar graphs (recall that a graph is *almost planar* if it has an edge whose removal leaves a planar graph), hereafter called the *one-edge crossing minimization*:

*Input:* A *planar* graph $G$ and two nonadjacent vertices $u, v$ of $G$.
*Problem:* Find an optimal drawing of $G + uv$ (i.e. $G$ plus the edge $uv$), that is, a drawing with the minimum number of crossings.

Although we firmly believe that computing the crossing number of an almost planar graph cannot be an NP–hard problem, all our efforts to get a polynomial time algorithm failed (even for graphs with bounded degrees). Thus we moved on to investigate whether such a crossing number can be approximated by a polynomial time algorithm. Our aim in this section is to show the existence of such an approximation algorithm, for graphs with bounded degrees.

Attempts to solve the one-edge crossing minimization problem, in turn, have brought a closely related subproblem with the same input, hereafter called the *one-edge bridging minimization* for distinction. This modification asks for a planar embedding of $G$ such that inserting the edge $uv$ to it yields the minimum possible number of crossings. Let $\mathrm{br}(G, uv)$ denote the minimum number of crossings in the bridging minimization problem.

The one-edge bridging minimization problem has been completely solved, giving a linear-time optimal algorithm for it, by Gutwenger, Mutzel and Weiskircher [7,8]. As they observe, that algorithm does not necessarily yield an optimal solution to the crossing minimization problem, as the counterexample at the end of [8] (thereby attributed to G. Farr) shows. This is summarized in the following statement.

**Proposition 2.1.** *For each $k > 2$ there is a planar graph $G$, and vertices $u, v$, such that $\mathrm{cr}(G + uv) = 2$, but $\mathrm{br}(G, uv) = k$.*

It is interesting to mention that this was asked as an open question in the earlier conference version [7]. We have independently found a somewhat simpler counterexample, which is (for $k = 5$) illustrated in Fig. 1.

We note that both Farr's construction and our example make essential use of vertices of large degree (of order at least $2k$). It is thus natural to ask whether large degree vertices are an essential part of any such examples. Our following result settles this question — large degree vertices are indeed unavoidable.

**Theorem 2.2.** *Suppose that $G$ is a planar graph with maximum degree $\Delta$, and let $u, v$ be nonadjacent vertices of $G$. Then the one-edge bridging minimization problem on $G$ and $uv$ has an optimal solution with $\mathrm{br}(G, uv) \leq \Delta \cdot \mathrm{cr}(G + uv)$.*

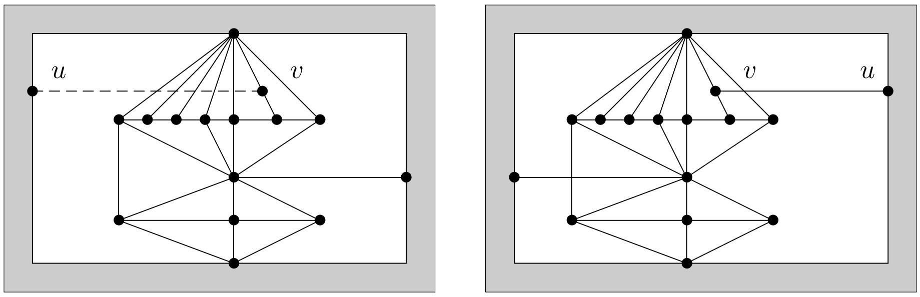


**Fig. 1.** A counterexample showing that a solution to one-edge bridging minimization (left, dashed edge $uv$) can be arbitrarily far from the crossing number (right). The shadow area stands for a sufficiently dense planar part.

We prove the theorem later in this section, using further Lemma 2.4. Our statement has the following nice consequence:

**Corollary 2.3.** *There is a polynomial time approximation algorithm for computing the crossing number of an almost planar graph with bounded degrees.*

*Proof.* Let $G$ be an almost planar graph with maximum degree $\Delta$. Apply any efficient planarity algorithm to find an edge $e$ of $G$ such that $G - e$ is planar. Then apply the linear time algorithm for one-edge bridging minimization from [8] to obtain a planar embedding of $G - e$ such that inserting $e$ into it yields the minimum possible number of crossings, say $k$. Obviously $\mathrm{cr}(G) \leq k$, and it follows from Theorem 2.2 that $k/\Delta \leq \mathrm{cr}(G)$. ∎

If $\Psi, \Psi'$ are embeddings of the same graph $G$, then $\Psi'$ is a *mirror* embedding of $\Psi$ if there is an orientation reversing homeomorphism taking $\Psi$ to $\Psi'$.

Suppose that $G$ has a 2–cut $\{x, y\}$, and let $G_1, G_2$ be subgraphs of $G$ such that $G$ is the sum of $G_1$ and $G_2$ along $x$ and $y$ (that is, $G_1$ and $G_2$ are edge-disjoint and share only $x$ and $y$, and $G = G_1 \cup G_2$). A (Whitney) *flipping* at $x, y$ is a re-embedding of $G$ such that the embedding of $G_2$ is unchanged and the embedding of $G_1$ is a mirror of the original embedding. We say that $G_1, G_2$ are the *sides* of the flipping. Any two (combinatorial) embeddings of a 2-connected planar graph can be transformed to each other by a sequence of flippings. We call a flipping *prime* if one of $G_1$ or $G_2$ has no cut-vertex separating $x$ from $y$. Obviously, every flipping can be decomposed into prime flippings.

**Lemma 2.4.** *Suppose $H$ is a connected plane graph (i.e. actually embedded in the plane), and $e, f$ are two edges not belonging to $H$ but connecting vertices of $H$, such that $H + f$ is a planar graph. If $e$ can be drawn in $H$ with $\ell$ crossings, then there is a planar embedding of $H + f$ in which $e$ can be drawn with at most $\ell + 2 \cdot \lfloor \Delta(H)/2 \rfloor$ crossings.*

It is worth noting that the statement may be false if $H$ is disconnected and $e, f$ join vertices from two distinct components.

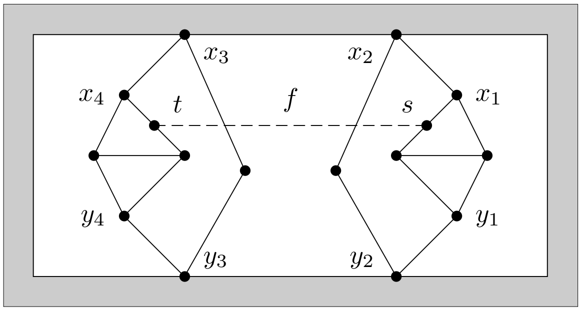


**Fig. 2.** An example of a planar graph in which four flippings are necessary before the edge $f$ (with endpoints $s$ and $t$) can be embedded in without crossings

*Proof.* First we assume that $H$ is 2-connected. Consider a sequence of flippings which turn $H$ into an embedding $H'$ such that $H' + f$ is plane (although we talk about a sequence, the order of the flippings actually does not matter to us). As noted above, we may assume the flippings under consideration are prime. Note that each of the sides of any relevant flipping contains one endpoint of $f$. We naturally order by inclusion the set of all the sides (of all the relevant flippings) containing an end $s$ of $f$, say as $S_1 \subset S_2 \subset \ldots \subset S_m$ (see also Fig. 2). Let $x_i, y_i$ be the cut-vertices at which the flipping involving $S_i$ occurs.

Observe further that it is enough to consider those flippings that have exactly one endpoint of $e$ on each side as well. (If there is no such flipping, then $e$ can be drawn with the same number of crossings in plane $H' + f$ as in $H$ itself.) Let $i$, $1 \leq i \leq m$, be the smallest index such that $S_i$ contains an endpoint of $e$. Thus, no endpoint of $e$ is in $S_{i-1}$ if $i > 1$, and exactly one endpoint $u$ of $e$ is in $S_i$. After we apply the flippings of $S_1, \ldots, S_{i-1}$, the vertex $s$ has to appear on the unbounded face of $S_i$ due to planarity of $H + f$ (note that those flippings do not affect the drawing or number of crossings on the edge $e$). We look at the "half-edge" $e_0 \subset e$ from the end $u$ till reaching the unbounded face of $S_i$ (for this moment we regard the edge $e$ as a topological object in the drawing $H + e$): We extend the curve $e_0$ to $e_1$ so that its loose end appears in a close neighbourhood of the vertex $s$ on the unbounded face of $S_i$, which takes only at most $\lfloor d_H(x_i)/2 \rfloor$ or $\lfloor d_H(y_i)/2 \rfloor$ additional crossings on $e_1$ when passing by either vertex $x_i$ or $y_i$, respectively. (See Fig. 3.)

A symmetric argument can be simultaneously used to argue that the other "half-edge" $e_0' \subset e$ from the end $v$ of the edge $e = uv$ can be redrawn as $e_1'$ such that its loose end appears in a close neighbourhood of the endpoint $t$ of the edge $f$. Then we apply the remaining flippings and embed the edge $f$ into $H'$. Finally, we join the loose ends of $e_1$ and $e_1'$ together along the (uncrossed) edge $f$ in $H' + f$, producing no additional crossing with $f$ since we have had above a choice of redrawing of $e_1$ either by $x_i$ or $y_i$. In this way we get a drawing of $e$ inside the plane graph $H' + f$ with at most $2 \cdot \lfloor \Delta(H)/2 \rfloor$ additional crossings.

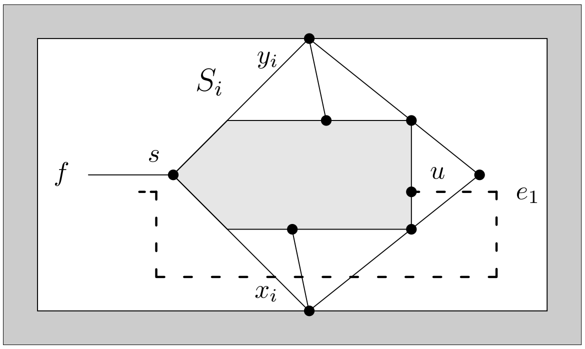


**Fig. 3.** An illustration of the proof of Lemma 2.4

At last we have to consider connected $H$ which is not 2-connected. Although all deep arguments have been used already above, some boring technical details are still necessary, and we only sketch them here for simplicity. Imagine $H$ decomposed into blocks. If the decomposition contains a leaf block not incident with $e, f$, then we may simply delete it. If the decomposition contains a leaf block incident with $f$ but not with $e$, then we may contract this block into one vertex without changing our problem. If the decomposition contains a leaf block incident with $e$ but not with $f$, then similarly we may contract this block into one vertex and just make a note of the number of crossings $e$ had with this block.

After processing the above reductions, we either arrive at a 2-connected graph, or we get a graph with precisely two leaf blocks $B_1, B_2$, both incident with $e$ and $f$. Then we adapt the above "half-edge" argument to this situation: We redraw the half-curve of $e$ from $B_1$ so that its loose end appears in a close neighbourhood of the cut-vertex $b_1$ of the block $B_1$, in the unbounded face. Symmetrically, we redraw the half-curve of $e$ from $B_2$ so that its loose end appears in a close neighbourhood of the cut-vertex $b_2$ of $B_2$. Finally we join the two halves of $e$ together in the unbounded face. This again costs at most $2 \cdot \lfloor \Delta(H)/2 \rfloor$ additional crossings. ∎

Having the previous lemma at hand, it is easy to finish the proof of Theorem 2.2. We actually prove a stronger statement (indeed, to see that Theorem 2.2 follows from Lemma 2.5, it suffices to note that $m \le \mathrm{cr}(G + uv) - \ell$ ).

**Lemma 2.5.** *Suppose that $G$ is a planar graph with maximum degree $\Delta$, and let $u, v$ be nonadjacent vertices of $G$. If there is a crossing-optimal drawing of $G + uv$ such that the edge $uv$ has $\ell$ crossings, and removing $m$ edges from this drawing of $G$ makes it plane, then* $\mathrm{br}(G, uv) \le \ell + 2\lfloor \Delta/2 \rfloor m$.

*Proof.* Let $F$ be a set of edges of $G$ such that $G' = G - F$ is plane and $|F| = m$. Obviously, for a crossing-optimal drawing $G + uv$ and minimal $F$, the graph $G'$ is connected. Let $F = \{f_1, \dots, f_m\}$. For $i = 1, 2, \dots, m$, we apply Lemma 2.4

for $H = G' + f_1 \ldots + f_{i-1}$ and $f = f_i$, $e = uv$, and continue with the resulting embedding of $H + f$ in the next iteration until $i = m$. ∎

**Remark.** One may also consider an analogous $k$-edge crossing minimization problem for fixed $k > 1$. Although the related $k$-edge bridging minimization problem [2, Problem 29] seems to have no efficient solution yet; if that is eventually found, then similar arguments could be used to prove it provides a good approximation for crossing minimization in the case of bounded degrees.

## 3 Projective Almost Planar Graphs and Crossing-Critical Graphs

Almost planar graphs are interesting both theoretically and practically. Although it is trivial to construct almost planar graphs with arbitrarily large crossing numbers (a large grid plus an edge joining vertices far apart from each other), a strong objection to such examples is that that the graph thus obtained is, in a way, not as close to being planar as it could be. Indeed, such a graph would have Euler genus 0 (since it is toroidal), thus being one step further away from planar than another large, interesting family, namely the collection of graphs with Euler genus 1. Recall that a graph has Euler genus 1 if and only if it is *projective*, that is, embeddable in the projective plane (equivalently, embeddable in the Möbius band).

Our aim in this section is to make several remarks on the richness of projective almost planar graphs. In our own view, this is the first step in a systematic program to understanding almost planar graphs: as we observed above, among almost planar graphs, projective graphs are arguably the simplest ones. We first observe that almost planar projective graphs can have arbitrary crossing number, and yet be strongly connected and have a small number of vertices (especially when compared to the number of vertices in the "grid example").

**Proposition 3.1.** *For every $k$, there is a simple 4-connected graph on $2k + 4$ vertices which is almost planar and projective, and whose crossing number is $k$.*

*Proof.* Let $D_m$ denote the double-wheel with the rim circuit of length $m$, and let $F_k$ be the graph obtained from $D_{2k+2}$ by joining one pair of opposite vertices on

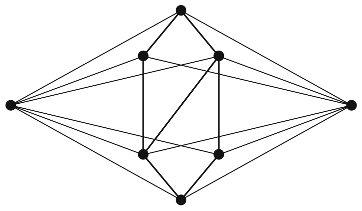


**Fig. 4.** The graph $F_2$ in the proof of Lemma 3.1 – a double-wheel of length 6 with an extra chord

the rim circuit with edge $f$. This graph is projective since one may use a “Möbius twist” applied to $f$ and $k$ consecutive spokes of one of the wheels (between the ends of $f$) to embed it.

By an easy induction on $k$ we show $\mathrm{cr}(F_k) = k$. For $k = 1$, $F_k$ is a subdivision of $K_5$, and so this is true. Again an argument with nonplanarity of $K_5$-subdivisions in $F_k$ shows that in any drawing of $F_k$ there has to be a length-2 path between the centers of the wheels that is crossed. Hence we remove the edges of this path and an opposite path (saving 1 crossing), and apply the inductive assumption for the resulting subdivision of $F_{k-1}$. ∎

If we admit the possibility of multiple edges, analogous ideas can be used to find even smaller graphs with similar properties.

**Proposition 3.2.** *For every $\ell$, there is a graph with $4\ell + 6$ edges which is both almost planar and projective, and whose crossing number is $\ell$.*

*Proof.* This graph ($C_{\ell+2} \oplus K_2^{\ell}$) is obtained from the disjoint union of $C_{\ell+2}$ and $\ell$ parallel edges $K_2^{\ell}$ by adding all edges between them. (Notice that this graph is actually $\ell$-crossing-critical, but since it is not simple, this does not make a “good” crossing-critical family.) The proof is very similar to that of Proposition 3.1. ∎

If, moreover, we bound the maximal degree, we observe the following.

**Proposition 3.3.** *Suppose a graph $G$ is almost planar and projective, and the maximum degree of $G$ is bounded. Then the crossing number of $G$ is bounded as well, and so it can be computed in time $O(n^2)$.*

*Proof.* According to [12], any projective embedding of an almost planar graph $G$ (more generally of an apex graph) has “face-width at most two”. That precisely means the embedding admits a closed noncontractible curve intersecting $G$ just in two vertices $u, v$. Cutting the projective plane along this curve, one gets a planar embedding with two copies of $u$ and two of $v$ on the unbounded face. It is clearly possible to pairwise identify those two copies of $u$ and those of $v$, with an introduction of $\lfloor d_G(u)/2 \rfloor \cdot \lfloor d_G(v)/2 \rfloor$ crossings. Hence $\mathrm{cr}(G)$ is bounded if the degrees are bounded, and we may use Theorem 1.1. ∎

We finally show that almost planar projective graphs are rich enough to provide nontrivial examples of crossing-critical graphs. Recall that a graph $G$ is *$k$-crossing-critical* if $\mathrm{cr}(G) \geq k$ but $\mathrm{cr}(G - e) < k$ for all edges $e \in E(G)$. Crossing-critical graphs are of great importance in the theory of crossing numbers, for them giving an insight into the structural properties that force large crossing numbers. (Notice that, in any graph of crossing number $k$, by successive deleting of suitable edges we always find a $k$-crossing-critical subgraph. Hence if a graph class contains a rich subclass of crossing-critical graphs, then the crossing-minimization problem is likely not trivial on that class.)

A rich family of almost planar simple 3-connected $k$-crossing critical graphs for every $k \geq 3$ has been constructed by the first author in [9]. Looking at our second restriction, the family of [9] is not projective, but we have succeeded in modifying that construction to obtain the following.

**Theorem 3.4.** *For every $k \geq 3$, there is an infinite family of simple 3-connected $k$-crossing critical graphs which are almost planar and projective.*

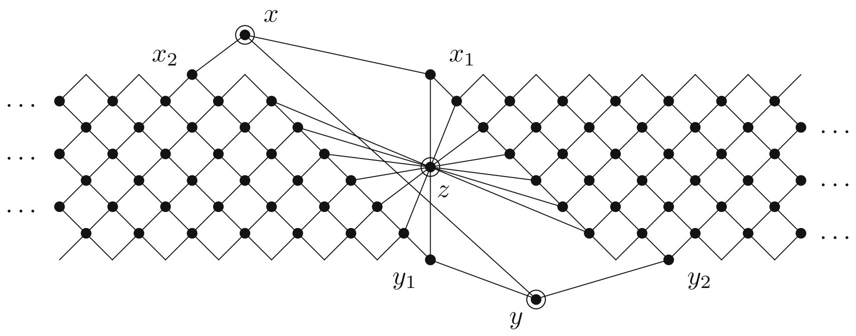


**Fig. 5.** An illustration of a $k$-belt graph, $k = 6$ here. The left- and right-hand sides close together as on a cylinder.

The construction of graphs in Theorem 3.4 is illustrated in Fig. 5, and we call them *$k$-belt graphs*. Formally for $k \geq 1$, a graph $G$ with distinguished three vertices $x, y, z$ is a $k$-belt graph if the following are satisfied:

– $G$ is simple of minimum degree 3 if $k > 2$.
– $B = G - \{x, y\}$ is formed as an edge-disjoint union of $k$ cycles sharing the vertex $z$. These cycles are denoted by $C_1, C_2, \ldots, C_k$ from top to bottom — referring the picture in Fig. 6 left.
– The neighbourhood of $z$ in $B$ looks exactly as depicted in Fig. 5. The neighbourhoods of other vertices of $B$ that are not adjacent to $z$ have all "square-grid" structure (with a small exception at $C_1$ and $C_k$).
– There are two special vertices $x_1, x_2 \in V(C_1) \setminus V(C_2)$ adjacent both to $x$ in $G$, such that $x_1$ is adjacent to $z$ while the distance of $x_2$ to $z$ is at least 3. Another two special vertices $y_1, y_2 \in V(C_k) \setminus V(C_{k-1})$ adjacent both to $y$ in $G$ are defined symmetrically around $z$. Moreover, in the subgraph $B - z$, it holds that $x_1$ is farther to $x_2$ than to $y_2$, and $y_1$ is farther to $y_2$ than to $x_2$. (To better understand this condition, check that it implies the length of $C_1$ is at least $k + 6$.)
– $xy$ is an edge in $G$. (Notice that $G - xy$ is a planar graph.)

We start with some straightforward statements about $G$ (see again Fig. 5).

**Lemma 3.5.** *If $G$ is a $k$-belt graph, $k \geq 1$, then*
*a) $G$ is almost planar, projective, and simple 3-connected if $k \geq 3$,*
*b) $\mathrm{cr}(G) \geq k$ but $\mathrm{cr}(G - e) < k$ for all $e \in E(G)$.*

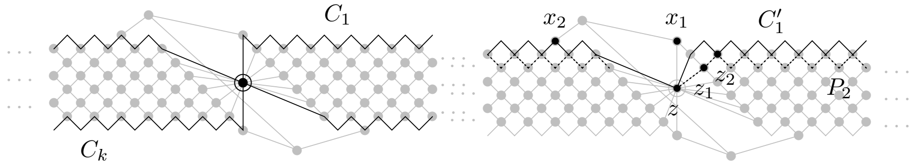


**Fig. 6.** Notation in $k$-belt graphs: (left) the cycles $C_1$ and $C_k$; (right) the cycle $C_1'$, vertices $z, z_1, z_2$, and dotted path $P_2$

To complete the proof of Theorem 3.4, it now remains to argue why a $k$-belt graph cannot be drawn with less than $k$ crossings. For further arguments, we denote by $C_1'$ the cycle in $B$ obtained from $C_1$ by deleting $x_1$ and using instead the edge from $z$ to the other neighbour of $x_1$ (Fig. 6 right). The following is easy to see from the definition of a $k$-belt graph.

**Lemma 3.6.** *Suppose $G$ is a $k$-belt graph, $k \geq 3$, and there is an optimal drawing of $G$ with $\ell$ crossings such that the cycle $C_1'$ is crossed. Let $G'$ be the subgraph formed by the edges $\big(E(G) \setminus E(C_1')\big) \cup \{x_2x_2'\}$, where $x_2'$ is one of the neighbours of $x_2$ in $C_1'$ such that $x_2x_2'$ is not the only crossed edge of $C_1'$. Then $G'$ is a subdivision of a $(k-1)$-belt graph, and* $\mathrm{cr}(G') \leq \ell - 1$.

In the opposite situation, i.e. when $C_1'$ is not crossed, we argue similarly. Let us denote by $z_1$ the neighbour of $z$ on $C_2$ and by $z_2$ the subsequent vertex on $C_2$. See in Fig. 6 right. The path starting with $z, z_1, z_2$ and continuing then along the vertices of $C_2$ up to the last one before returning to $z$ is denoted by $P_2$.

**Lemma 3.7.** *Suppose $G$ is a $k$-belt graph, $k \geq 4$, and there is an optimal drawing of $G$ with $\ell$ crossings such that $C_1'$ is* not *crossed. Then there are at least two distinct crossings on the path $P_2$. Consequently, there is a drawing $G' \subseteq G$ which is a subdivision of a $(k-2)$-belt, and* $\mathrm{cr}(G') \leq \ell - 2$.

*Proof.* Since $G - V(C_1')$ is a connected subgraph, all its vertices have to be drawn in one of the faces bounded by the drawing of $C_1'$. Hence by Jordan's curve theorem, the length-2 path $zz_1z_2$ has to be crossed since it separates the common neighbour of $z$ and $x_1$ from the rest of the drawing. The same can be said about the length-2 path on $P_2$ which connects the two vertices adjacent to $x_2$. Those are the desired two distinct crossings on $P_2$. ∎

**Lemma 3.8.** *Suppose $G$ is a $k$-belt graph, $k = 1$ or $k \geq 3$. Then* $\mathrm{cr}(G) \geq k$.

*Proof.* We proceed by an induction on $k$, similarly as we have done in [9]. Notice that a 1-belt graph is actually a subdivision of $K_{3,3}$, and hence the base of induction holds. The main complication comes from the fact that the inductive statement is false for $k = 2$, and we have to carefully avoid this in our arguments.

First consider $k > 4$, and the statement is true for all $i < k$, $i \neq 2$. Take an optimal drawing of $G$ with $\ell$ crossings. Then, depending on whether the cycle $C_1'$ is crossed, apply Lemma 3.6 or 3.7 straightforwardly: For $a = 1$ or $a = 2$, it is $k - a \leq \mathrm{cr}(G') \leq \ell - a$, and hence $k \leq \ell$. Next consider $k = 4$. If Lemma 3.6 applies to $G$, induction proceeds in the same way. So assume $C_1'$ is not crossed now, and by Lemma 3.7 there are two crossings on the path $P_2$. The same arguments can be applied to symmetrically defined cycle $C_4'$ and path $P_3$ in $G$, and we find additional two crossings on $P_3$ (which is disjoint from $P_2$); altogether $4 = k$ crossings.

It remains to prove the statement for $k = 3$. This is straightforward but slightly too long for this restricted conference paper, and so we only sketch the arguments. If $C_1'$ is not crossed, then we may still consistently apply Lemma 3.7 and induction. Otherwise, considering symmetry, both cycles $C_1'$ and (symmetric) $C_3'$ are crossed. (It may happen that $C_1'$ crosses $C_3'$.) Let a cycle $C_2'$ be obtained from $C_2$ by replacing the two edges from $C_2 \cap (C_1' \cup C_3')$ with the path through $y_1, z, x_1$. If the cycle $C_2'$ is crossed as well, then we apply an inductive step with removing (most of) edges of $C_1'$ and $C_2'$. Finally, if $C_2'$ is not crossed at all, then one can show that $C_1'$ and $C_3'$ carry at least three distinct crossings, since in such case the vertices $x, y, x_2, y_2$ have to be drawn in the same face bounded by the drawing of $C_2'$. ∎

We remark that the above $k$-belt construction could be generalized in a similar way as the construction in [9] was. We however skip such a generalization here to avoid the boring technical details of it.

## 4 Conclusions

In this paper we mainly wanted to attract attention and research to the seemingly simple, but still quite deep and unexplored problem of crossing minimization on almost planar graphs. As our first step, we have shown two new results – one on the positive side (Theorem 2.2 and Corollary 2.3), and the other one somehow negative (Theorem 3.4), indicating richness and nontriviality of the problem we study.

Questions to the effect of whether crossing minimization remains hard even if we focus our attention on restricted families of graphs can be interpreted as partial efforts to answer an admittedly vague, but nonetheless appealing and still wide-open, question: How much "nonplanarity" must one admit into a family of graphs $\mathcal{G}$ in order to guarantee that computing the crossing number of a graph in $\mathcal{G}$ is hard?

We put forward three questions, in order of apparent difficulty.

*Question 4.1.* Is there a polynomial-time algorithm to approximate the crossing number of an almost planar graph?

*Question 4.2.* Is there a polynomial-time algorithm to compute the crossing number of an almost planar graph?

A simplified version of this last question would consider graphs with bounded degree, as we have done in the present paper.

For the next question, recall than a graph $G$ is *apex* if it has a vertex $v$ such that $G - v$ is planar.

*Question 4.3.* Is it an NP–hard problem to compute the crossing number of an apex graph?

We conjecture that all these questions have affirmative answers. We lastly remark that Question 4.3 has been asked also by Mohar [private communication].

## References

1. D. Bokal, G. Fijavz, and B. Mohar, *The minor crossing number*, SIAM Journal on Discrete Mathematics, to appear.
2. F. Brandenburg, D. Eppstein, M.T. Goodrich, S. Kobourov, G. Liotta and P. Mutzel, *Selected Open Problems in Graph Drawing*, In: Graph Drawing, GD 2003, Lecture Notes in Computer Science 2912, Springer Verlag 2004, 515–539.
3. G. Even, S. Guha, B. Schieber, *Improved Approximations of Crossings in Graph Drawings and VLSI Layout Areas*, SIAM J. Comput. **32**(1), 231–252 (2002).
4. M.R. Garey, D.S. Johnson, *Crossing number is NP-complete*, SIAM J. Algebraic Discrete Methods 4 (1983), 312–316.
5. I. Gitler, J. Leaños, and G. Salazar, *The crossing number of a projective graph is quadratic in the face–width*. Manuscript (2006).
6. M. Grohe, *Computing Crossing Numbers in Quadratic Time*, J. Comput. Syst. Sci. 68 (2004), 285–302.
7. C. Gutwenger, P. Mutzel, R. Weiskircher, *Inserting an edge into a planar graph*, Symposium on Discrete Algorithms SODA 2001, SIAM (2001), 246–255.
8. C. Gutwenger, P. Mutzel, R. Weiskircher, *Inserting an edge into a planar graph*, Algorithmica 41 (2005), 289–308.
9. P. Hliněný, *Crossing-critical graphs and path-width*, In: Graph Drawing, 9th Symposium GD 2001, Vienna Austria, September 2001; Lecture Notes in Computer Science 2265, Springer Verlag 2002, 102–114.
10. P. Hliněný, *Crossing number is hard for cubic graphs*, J. of Combinatorial Theory ser. B 96 (2006), 455–471.
11. B. Mohar, C. Thomassen, Graphs on surfaces, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press (2001), Baltimore MD, USA.
12. N. Robertson, P.D. Seymour, R. Thomas, *A survey of linkless embeddings*, In: Graph Structure Theory (N. Robertson, P. Seymour, eds.), Contemporary Mathematics, American Mathematical Society, 1993, 125–136.

# On the Decay of Crossing Numbers

Jacob Fox[1] and Csaba D. Tóth[2]

[1] Department of Mathematics, Princeton University, Princeton, NJ
licht@mit.edu

[2] Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA
toth@math.mit.edu

**Abstract.** The crossing number $\mathrm{cr}(G)$ of a graph $G$ is the minimum number of crossings over all drawings of $G$ in the plane. In 1993, Richter and Thomassen [RT93] conjectured that there is a constant $c$ such that every graph $G$ with crossing number $k$ has an edge $e$ such that $\mathrm{cr}(G-e) \geq k - c\sqrt{k}$. They showed only that $G$ always has an edge $e$ with $\mathrm{cr}(G-e) \geq \frac{2}{5}\mathrm{cr}(G) - O(1)$. We prove that for every fixed $\epsilon > 0$, there is a constant $n_0$ depending on $\epsilon$ such that if $G$ is a graph with $n > n_0$ vertices and $m > n^{1+\epsilon}$ edges, then $G$ has a subgraph $G'$ with at most $(1 - \frac{1}{24\epsilon})m$ edges such that $\mathrm{cr}(G') \geq (\frac{1}{28} - o(1))\mathrm{cr}(G)$.

## 1 Introduction

The crossing number $\mathrm{cr}(G)$ of a (simple) graph $G$ is the minimum possible number of crossings in any drawing of $G$ in the plane. A famous result of Ajtai *et al.* [ACNS82] and Leighton [L84] states that if $G$ is a graph with $n$ vertices and $m \geq 4n$ edges, then

$$\mathrm{cr}(G) \geq \frac{m^3}{64n^2}. \tag{1}$$

For graphs with $n$ vertices and $m \geq \frac{103}{16}n$ edges, Pach *et al.* [PRTT04] improved Inequality (1) by a constant factor to

$$\mathrm{cr}(G) \geq \frac{1024}{31827}\frac{m^3}{n^2}. \tag{2}$$

It is well known that for every positive integer $k$, there is a graph $G$ and an edge $e$ of $G$ such that $\mathrm{cr}(G) = k$ but $G - e$ is planar. In 1993, Richter and Thomassen [RT93] conjectured that there is a constant $c$ such that for every nonempty graph $G$ with crossing number $k$, there is an edge $e$ of $G$ such that $\mathrm{cr}(G-e) \geq k - c\sqrt{k}$. They showed only that $G$ always has an edge $e$ with $\mathrm{cr}(G-e) \geq \frac{2}{5}\mathrm{cr}(G) - O(1)$. Salazar [S00] proved that for every graph $G$ with no vertices of degree 3, there is an edge $e$ of $G$ such that $\mathrm{cr}(G-e) \geq \frac{1}{2}\mathrm{cr}(G) - O(1)$. Pach and G. Tóth [PT00] showed for every connected graph $G$ with $n$ vertices, $m \geq 1$ edges, and every edge $e$ of $G$, that the decay is bounded by

$$\mathrm{cr}(G-e) \geq \mathrm{cr}(G) - m + 1.$$

This, combined with Inequality (2), is better than Richter-Thomassen's bound for graphs with $n$ vertices and $m \geq 8.1n$ edges. By Inequality (1), it also confirms the Richter-Thomassen conjecture for dense graphs, that is, for graphs with $\Omega(n^2)$ vertices.

In this paper, we show that from every graph $G$ that is not too sparse, we can delete a constant fraction of the edges such that the the crossing number of the remaining subgraph $G'$ is at least a constant fraction of the crossing number of $G$.

**Theorem 1.** *For every $\epsilon > 0$, there is a constant $n_0$ depending on $\epsilon$ such that if $G$ is a graph with $n > n_0$ vertices and $m > n^{1+\epsilon}$ edges, then $G$ has a subgraph $G'$ formed by deleting at least $\epsilon m/24$ edges from $G$ such that*

$$\mathrm{cr}(G') \geq \left(\frac{1}{28} - o(1)\right) \mathrm{cr}(G).$$

To prove Theorem 1, we derive in Sections 3 and 4 new lower bounds on the crossing number that improve on Inequality (1) for graphs with highly irregular degree patterns.

## 2 Drawing Edges with the Embedding Method

We use the embedding method along the lines of Leighton [L83], Richter and Thomassen [RT93], Shahrokhi *et al.* [SSSV97], and Székely [S04a]. The embedding method generates a planar drawing (embedding) $D(G)$ of a graph $G$ based on a drawing $D(H)$ of a subgraph $H \subset G$. The drawing $D(G)$ respects $D(H)$ on the edges of $H$ and for every edge $e = (v, w) \in G \setminus H$, the drawing of $e$ follows "infinitesimally close" to a path between $v$ and $w$ in the drawing $D(H)$. We can distinguish two categories of crossings that involve edges of $G \setminus H$ in the drawing $D(G)$. A *first category crossing* arises infinitesimally close to a crossing in $D(H)$. A *second category crossing* arises infinitesimally close to a vertex in $D(H)$.

We illustrate the embedding method with a bound on the minimum decay of the crossing number after deleting *one* edge. This improves on the Richter-Thomassen bound for graphs with $m \geq 7.66n$ edges.

**Proposition 1.** *For every connected graph $G$ with $n$ vertices and $m$ edges, there is an edge $e$ of $G$ such that*

$$\mathrm{cr}(G - e) \geq \frac{p}{p+2}\left(\mathrm{cr}(G) - m + \frac{n}{2}\right),$$

*where $p = \lceil \frac{m}{n-1} - 1 \rceil$.*

The proof of Proposition 1 follows immediately from Proposition 2 and Lemma 1 below. Nagamochi and Ibaraki [NI92] proved the following lemma, which is a slight variant of Mader's theorem, and shows that every graph with $n$ vertices and $m$ edges has a pair of adjacent vertices with at least $\frac{m}{n-1}$ edge-disjoint paths between them.

**Lemma 1 (Mader, Nagamochi and Ibaraki).** *If $G$ is a graph with $m$ edges and $n$ vertices, then there is an edge $e = (v, w)$ of $G$ such that there are at least $\frac{m}{n-1} - 1$ edge-disjoint paths between $v$ and $w$ in $G - e$.*

*Proof.* Delete maximal spanning forests $F_1$, $F_2$, ..., $F_j$ one after the other until all edges are deleted. If $e = (v, w)$ is an edge of $F_j$, then there is a path between $v$ and $w$ in $F_i$ for every $i$, $1 \leq i \leq j$. Hence, there are at least $j - 1$ edge-disjoint paths between $v$ and $w$ that do not pass through $e$. Since each $F_i$ is a forest, it has at most $n - 1$ edges, and so we have $m \leq j(n - 1)$. Substituting, there are at least $\frac{m}{n-1} - 1$ edge-disjoint paths between $v$ and $w$ in $G - e$.

**Proposition 2.** *Let $G$ be a connected graph with $n$ vertices and $m$ edges, and $e = (v, w)$ be an edge of $G$ such that there are $p \geq 1$ edge-disjoint paths between $v$ and $w$ in $G - e$. Then*

$$\mathrm{cr}(G) \leq \left(1 + \frac{2}{p}\right) \mathrm{cr}(G - e) + m - \frac{n}{2}.$$

*Proof.* Let $D$ be a drawing of $G - e$ in the plane with $\mathrm{cr}(G - e)$ crossings. Let $P_1$, $P_2$, ..., $P_p$ be $p$ edge-disjoint paths between $v$ and $w$. Consider the drawing $D_j$ of $G$ in the plane that respects the drawing $D$ of $G - e$ and the edge $e$ follows infinitesimally close to the path $P_j$ between $v$ and $w$ with all loops (and self-crossings) deleted. Let $k_j$ be the number of first category crossings in $D_j$. Since the paths $P_1$, $P_2$, ..., $P_p$ are edge-disjoint, the drawings $D_1$, $D_2$, ..., $D_p$ of $G$ jointly have at most two first category crossings at each crossing of $D$: at most two crossings between edges of $G - e$ and different drawings of $e$, as depicted in Figure 1(a). Hence,

$$\sum_{j=1}^{p} k_j \leq 2\mathrm{cr}(G - e).$$

Therefore, there is an index $j$, $1 \leq j \leq p$, such that $k_j \leq 2\mathrm{cr}(G - e)/p$.

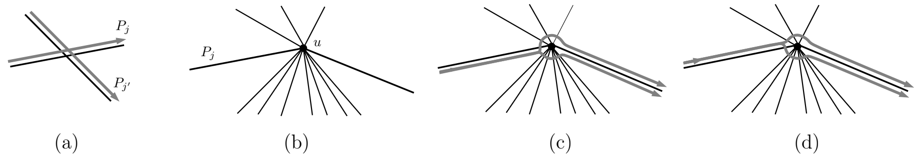


**Fig. 1.** Drawings of edge $e$ along two edge-disjoint path $P_j$ and $P_{j'}$ may give two first category crossings at a crossing of $D$ (a). If a path $P_j$ traverses a vertex $u$ (b), then the edge $e$ drawn along $P_j$ can choose between two possible routes around $u$ (c–d).

At each internal vertex $u$ of a path $P_j$, the drawing of $e$ in $D_j$ can take two possible routes, as depicted in Figure 1 (c–d). The two possible routes have a total of $\deg(u)-1$ second category crossings at $u$. We draw $e$ along the route with fewer second category crossings, and so there are at most $\frac{1}{2}(\deg(u)-1)$ crossing at vertex $u$. Hence, the total number of second category crossings is at most $m - \frac{n}{2}$. Therefore, in the drawing $D_j$ of $G$, there are at most $(1 + \frac{2}{p})\mathrm{cr}(G - e) + m - \frac{n}{2}$ crossings.

The following theorem establishes Theorem 1 for all graphs with $n$ vertices of degree $d_1, \ldots, d_n$ such that $\mathrm{cr}(G) \geq \frac{7}{16}\sum_{i=1}^{n} d_i^2$. For graphs that do not satisfy this condition, we alter the proof in Sections 3 and 4.

**Theorem 2.** *For every $\epsilon$, $0 < \epsilon < 1$, there is a positive constant $n(\epsilon)$ such that for every $G$ with $n > n(\epsilon)$ vertices, with a degree sequence $d_1, \ldots, d_n$, and $m > n^{1+\epsilon}$ edges, there is a subgraph $G'$ of $G$ with at most $(1 - \frac{\epsilon}{8})m$ edges such that*

$$4\mathrm{cr}(G') \geq \mathrm{cr}(G) - \frac{3}{8}\sum_{i=1}^{n} d_i^2.$$

*Proof.* Erdős and Simonovits [ES82] proved that for every integer $r > 1$, there is a constant $c_r$ such that every graph $G$ with $n$ vertices and $m > c_r n^{1+\frac{1}{r}}$ edges contains a cycle of length $2r$. This implies that for $0 < \epsilon < 1$, there is a positive integer $r$ satisfying $\frac{1}{\epsilon} < r \leq \frac{2}{\epsilon}$ so that every sufficiently large graph $G$ with $m > n^{1+\epsilon}$ edges contains a family $\mathcal{C}$ of edge-disjoint cycles of length $2r$ that cover at least half of the edges of $G$. Let $G'$ be a subgraph of $G$ formed by deleting an arbitrary edge $e_j$ from each cycle $C_j \in \mathcal{C}$. The remaining edges of cycle $C_j$ form a path $P_j$. Hence, the number of edges of $G \setminus G'$ is at least $\frac{\epsilon}{8}m$. Let us denote the vertices of $G$ by $v_i$, $i = 1, 2, \ldots, n$, such that the degree of $v_i$ is $d_i$ in $G$ and $d_i'$ in $G'$. Let $h_i = d_i - d_i'$, which is the number of edges incident to $v_i$ in $G \setminus G'$.

Consider a drawing $D'$ of $G'$ in the plane with $\mathrm{cr}(G')$ crossings. We generate a drawing $D$ of $G$ based on $D'$ by applying the embedding method. In particular, for every edge $e_j$ of cycle $C_j \in \mathcal{C}$, we draw $e_j$ along the path $P_j$. Since the paths $P_j$ with $C_j \in \mathcal{C}$ are edge-disjoint, $D$ has at most 4 crossings at every crossing of $D'$. Therefore, the total number of crossings of $D'$ and first category crossings of $D$ is at most $4\mathrm{cr}(G')$.

Next we estimate the number of second category crossings. Each of the $h_i$ edges incident to $v_i$ in $G \setminus G'$ is drawn, in a neighborhood of $v_i$, close to one of the $d_i'$ edges incident to $v_i$ in $G'$. The vertex $v_i$ with degree $d_i'$ in $G'$ is an internal node of at most $\lfloor (d_i' - h_i)/2 \rfloor$ paths $P_j$. For every such path $P_j$, the edge $e_j$ is drawn along one of two possible routes, as depicted in Figure 1(c-d), with the minimum number of crossings with the edges of $G$ incident to the vertex $v_i$. Every edge $e_j \in G \setminus G'$ passing though a small neighborhood of $v_i$ has at most $\lfloor (d_i' + h_i - 1)/2 \rfloor$ second category crossings with edges of $G$ incident to $v_i$. Each pair of edges passing through a small neighborhood of $v_i$ cross at most once. So the total number of second category crossings at $v_i$ is at most

$$\left\lfloor \frac{d_i' - h_i}{2} \right\rfloor \cdot \left\lfloor \frac{d_i' + h_i - 1}{2} \right\rfloor + \binom{\lfloor d_i' - h_i/2 \rfloor}{2} < \frac{3}{8}d_i'^2 \leq \frac{3}{8}d_i^2.$$

Summing over all vertices, we have at most $\sum_{i=1}^{n} \frac{3}{8}d_i^2$ second category crossings.

Hence, we have
$$\mathrm{cr}(G) \leq 4\mathrm{cr}(G') + \frac{3}{8}\sum_{i=1}^{n} d_i^2. \qquad \square$$

## 3 The Sum of Degree Squares and the Crossing Number

The *bisection width*, denoted by $b(G)$, is defined for every simple graph $G$ with at least two vertices. $b(G)$ is the smallest nonnegative integer such that there is a partition of the vertex set $V = V_1 \cup^* V_2$ with $\frac{1}{3} \cdot |V| \leq V_i \leq \frac{2}{3} \cdot |V|$ for $i = 1, 2$, and $|E(V_1, V_2)| \leq b(G)$. Extending the Lipton-Tarjan separator theorem [LT79], Gazit and Miller [GM90] established an upper bound on the bisection width in terms of the sum of degree squares.

**Theorem 3 (Gazit and Miller).** *Let $G$ be a* planar *graph with $n$ vertices of degree $d_1, d_2, \ldots, d_n$. Then*

$$b^2(G) \leq \frac{5 + 2\sqrt{6}}{4} \cdot \sum_{i=1}^{n} d_i^2.$$

Pach, Shahrokhi, and Szegedy [PSS96] used Theorem 3 to relate the bisection width with the crossing number.

**Theorem 4 (Pach, Shahrokhi, and Szegedy).** *Let $G$ be a graph with $n$ vertices of degree $d_1, d_2, \ldots, d_n$. Then*

$$40\mathrm{cr}(G) \geq b^2(G) - \frac{5}{2} \cdot \sum_{i=1}^{n} d_i^2(G).$$

Pach, Spencer and Tóth [PST00] have further exploited the connection between the bisection width and the crossing number. They have established lower bounds on the crossing number of graphs with some monotone graph property in terms of the number of edges and vertices of the graph. A simplified version of their proof method yields the following bounds.

**Lemma 2.** *Let $G(V, E)$ be a graph with $n$ vertices of degree $d_1, d_2, \ldots, d_n$, and $m \geq 8n^{7/5} \log^{2/5} n$ edges. Then*

$$\mathrm{cr}(G) \geq \frac{1}{24} \sum_{i=1}^{n} d_i^2.$$

This bound is better than the classical lower bound (1) due to Ajtai et al. [ACNS82] and Leighton [L84] for graphs of irregular degree patterns and $m = O(n^{3/2})$ edges. Consider the complete bipartite graph $K_{a,b}$ with $n = a+b$ vertices and $m = ab$ edges, where $a \leq b$. For this graph, our Lemma 2 gives $\mathrm{cr}(G) = \Omega(ab^2)$, which is a tighter than the classical $\Omega(m^3/n^2) = \Omega(a^3 b)$ bound for $(8 + o(1))b^{2/5} \log^{2/5} b \leq a \leq \sqrt{b}$, where the $o(1)$ term goes to 0 as $b \to \infty$. Similar bounds have also been deduced by Pach, Solymosi, and Tardos [PST06].

*Proof of Lemma 2.* We decompose the graph $G$ by the following recursive algorithm into induced subgraphs such that every subgraph is either a singleton or its squared bisection width is at least five times the sum of its degree squares. In an induced subgraph $H \subseteq G$, we denote by $\deg_H(v)$ the degree of a vertex $v \in V(H)$.

```
1. Let $S_0 = \{G\}$ and $i = 0$.
2. Repeat until $|V(H)| = 1$ or $b^2(H) \geq 5\sum_{v\in H}\deg_H^2(v)$ for every $H \in S_i$.
       Set $i := i + 1$ and $S_{i+1} := \emptyset$. For every $H \in S_i$, do
       • If $b^2(H) \geq 5\sum_{v\in H}\deg_H^2(v)$ or $|V(H)| \leq (2/3)^i|V|$, then let $S_{i+1} := S_{i+1} \cup \{H\}$;
       • otherwise split $H$ into graphs $H_1$ and $H_2$ along an edge separator of size $b(H)$, and let $S_{i+1} := S_{i+1} \cup \{H_1, H_2\}$.
3. Return $S_i$.
```

First, we show that the algorithm is correct. In every round, every graph $H \in S_i$ that does not satisfy the end condition has at most $|V(H)| \leq (2/3)^i \cdot |V|$ vertices. The algorithm terminates in $t \leq \log_{(3/2)} n$ rounds, and it returns a set $S_t$ of induced subgraphs. By Theorem 4 and the end condition of the decomposition algorithm, for every $H \in S_t$ we have $40\text{cr}(H) \geq (5/2)\sum_{v\in H}\deg_H^2(v)$. So

$$40\text{cr}(G) \geq 40\sum_{H\in S_t}\text{cr}(H) \geq \frac{5}{2}\cdot\sum_{H\in S_t}\sum_{v\in H}\deg_H^2(v) \geq \frac{5}{2}\cdot\sum_{v\in V}\deg_{H(v,t)}^2(v), \quad (3)$$

where $H(v,i)$ denotes the graph $H \in S_i$ containing vertex $v \in V$.

Next, we count the number of edges deleted during the recursive decomposition. Following an argument of [PST00], we count separately the edges deleted in each step of the decomposition algorithm. Let $S_i' = \{H : H \in S_i, H \notin S_{i+1}\}$, that is, $S_i'$ consists of those subgraphs in $S_i$ that are decomposed at step $i$. Notice that $|S_i'| < (\frac{3}{2})^{i+1}$ since every subgraph of $S_i$ that splits has more than $(2/3)^{i+1}|V|$ vertices. Let $V_i = \{v : v \text{ is a vertex of a graph } H \in S_i'\}$.

In step $i$, when some of the subgraphs in $S_i$ are decomposed in $S_{i+1}$, the total number of deleted edges is at most

$$\sum_{H\in S_i'}\sqrt{5\sum_{v\in H}\deg_H^2(v)}.$$

Using the Cauchy-Schwartz inequality, we have

$$\sum_{H\in S_i'}\sqrt{5\sum_{v\in H}\deg_H^2(v)} \leq \sqrt{5|S_i'|}\sqrt{\sum_{v\in V_i}\deg_{H(v,i)}^2(v)} \leq \sqrt{5\left(\frac{3}{2}\right)^{i+1}}\sqrt{\sum_{v\in V_i}\deg_{H(v,i)}^2(v)}.$$

Since $|V(H)| \leq (\frac{2}{3})^i|V|$ for each subgraph $H \in S_i'$, we conclude that

$$\sqrt{5\left(\frac{3}{2}\right)^{i+1}}\sqrt{\sum_{v\in V_i}\deg_{H(v,i)}^2(v)} \leq \sqrt{5\left(\frac{3}{2}\right)^{i+1}}\sqrt{\max_{v\in V_i}\deg_{H(v,i)}(v)\cdot\sum_{v\in V_i}\deg_{H(v,i)}(v)}$$

$$\leq \sqrt{5\left(\frac{3}{2}\right)^{i+1}}\sqrt{\left(\frac{2}{3}\right)^{i}n(2m)} \leq \sqrt{15mn}.$$

Since the algorithm terminates in at most $\log n/\log(3/2)$ steps, the total number of edges deleted throughout the decomposition algorithm is at most

$$\frac{\sqrt{15}}{\log(3/2)}\sqrt{mn}\log n < 7\sqrt{mn}\log n.$$

If we increase the degree of a vertex by one, the degree square increases by at most $2n-1 < 2n$. By putting back the deleted edges, the sum of degree squares increases by less than $28m^{1/2}n^{3/2}\log n$. From Inequality (2), we have

$$8\mathrm{cr}(G) \geq 8 \cdot \frac{1024}{31827} \cdot \frac{m^3}{n^2} \geq 28m^{1/2}n^{3/2}\log n, \tag{4}$$

if $m \geq 8n^{7/5}\log^{2/5} n$. Summing Inequalities (3) and (4), we obtain

$$24\mathrm{cr}(G) \geq \sum_{v\in V} \deg^2_{H(v,t)}(v) + 88m^{1/2}n^{3/2}\log n \geq \sum_{i=1}^{n} d_i^2.$$

This completes the proof of Lemma 2. □

We are now ready prove Theorem 1 for the case that $m \geq 8n^{7/5}\log^{2/5} n$.

**Theorem 5.** *For every $\epsilon > 0$, there is a constant $n_0$ depending on $\epsilon$ such that if $G$ is a graph with $n > n_0$ vertices and $m > 8n^{7/5}\log^{2/5} n$ edges, then $G$ has a subgraph $G'$ formed by deleting at least $m/20$ edges from $G$ such that $\mathrm{cr}(G') \geq \frac{1}{13}\mathrm{cr}(G)$.*

*Proof.* Combining Theorem 2 and Lemma 2, we obtain

$$\mathrm{cr}(G) \leq 4\mathrm{cr}(G') + \frac{3}{8}\sum_{i=1}^{n} d_i^2 \leq 4\mathrm{cr}(G') + 9\mathrm{cr}(G') = 13\mathrm{cr}(G').$$

□

## 4 Proof of Theorem 1

Theorem 5 leaves us with the case that $n^{1+\epsilon} \leq m < 8n^{7/5}\log^{2/5} n$. Instead of Lemma 4, we employ the following bounds.

**Lemma 3.** *Let $G$ be a graph with $n$ vertices of degree $d_1, d_2, \ldots, d_n$, and $m$ edges. For any $\delta$, $0 < \delta < 1$, let $\Delta = \Delta(\delta)$ be the integer such that $\sum_{i=1}^{n}\min(d_i, \Delta) < 2\delta m$ but $\sum_{i=1}^{n}\min(d_i, \Delta+1) \geq 2\delta m$. The crossing number of $G$ is bounded by the sum of truncated degree squares. If $m \geq 45(1-\delta)^{-2}n\log^2 n$, then*

$$\mathrm{cr}(G) \geq \frac{1}{16}\sum_{i=1}^{n}(\min(d_i, \Delta))^2.$$

**Lemma 4.** *Let $G$ be a graph with $n$ vertices and $m$ edges, and let $d_1 \leq d_2 \leq \ldots \leq d_n$ denote the degree sequence sorted in monotone increasing order. Let $\ell$ be the integer such that $\sum_{i=1}^{\ell-1} d_i < 4m/3$ but $\sum_{i=1}^{\ell} d_i \geq 4m/3$. The crossing number of G is bounded by a prefix sum of the degree squares. If $m = \Omega(n \log^2 n)$, then*

$$\mathrm{cr}(G) \geq \left(\frac{1}{64} - o(1)\right) \sum_{i=1}^{\ell} d_i^2.$$

*Proof of Lemma 3.* Run the recursive decomposition algorithm described in the previous section on graph $G$. We have shown that during the algorithm at most $(\sqrt{15}/\log\frac{3}{2})$
$\sqrt{mn}\log n$ edges are deleted. This is less than $(1-\delta)m$ if $m \geq 45(1-\delta)^{-2} n \log^2 n$.

We are now ready to estimate $\sum_{v \in V} \deg^2_{H(v,t)}(v)$. Since the number of edges decreased by at most $(1-\delta)m$, the sum of degrees decreased by at most $(2-2\delta)m$. The sum of degree squares decreases maximally if the highest degrees are truncated to at most $\Delta$, and so we have

$$\sum_{v \in V} \deg^2_{H(v,t)}(v) \geq \sum_{i=1}^{n} \left(\min(d_i, \Delta)\right)^2 . \tag{5}$$

This completes the proof of Lemma 3. □

*Proof of Lemma 4.* We extend the argument of the previous proof with $\delta = \frac{5}{6}$. If $d_\ell \leq \Delta$, then the right hand side of (5) must clearly be at least $\sum_{i=1}^{\ell} d_i^2$ and our proof is complete. Let us assume that $\Delta < d_\ell$. Refer to Figure 2.

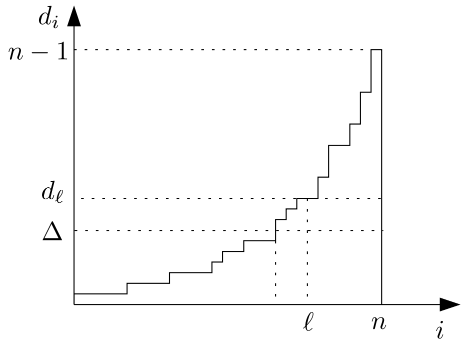


**Fig. 2.** The monotone increasing degree sequence of a graph $G$

Recall that $\sum_{i=1}^{n} d_i = 2m$. We have assumed that $\sum_{i=\ell+1}^{n} d_i \leq \frac{2m}{3} < \sum_{i=\ell}^{n} d_i$, and for $\delta = \frac{5}{6}$ we have $\sum_{i=1}^{n} \min(d_i, \Delta) < \frac{5m}{3} \leq \sum_{i=1}^{n} \min(d_i, \Delta+1)$. It follows that $(n-\ell+1)(\Delta+1) > \frac{m}{3}$. Since $\Delta < n$ and $n = o(m)$, we conclude that $(n-\ell)\Delta > (1-o(1))\frac{m}{3}$.

Observe that $(n-\ell)d_\ell \le \sum_{i=\ell+1}^n d_i \le \frac{2m}{3}$, and so $m \ge \frac{3}{2}(n-\ell)d_\ell$. Furthermore, observe that $\sum_{i=1}^{\ell} \max(0, d_i - \Delta) \le \sum_{i=1}^{n} \max(0, d_i - \Delta) \le n + \sum_{i=1}^{n} \max(0, d_i - (\Delta+1)) \le n + \frac{m}{3} = (1+o(1))\frac{m}{3}$. Putting these simple observations together, we obtain

$$\begin{aligned}\sum_{i=\ell+1}^{n} (\min(d_i,\Delta))^2 = (n-\ell)\Delta^2 > \left(\frac{1}{3}-o(1)\right) m\Delta \ge \left(\frac{1}{2}-o(1)\right)(n-\ell)d_\ell\Delta \\ \ge \left(\frac{1}{6}-o(1)\right) d_\ell m \ge \left(\frac{1}{2}-o(1)\right) d_\ell \sum_{i=1}^{\ell} \max(0, d_i-\Delta) \\ \ge \left(\frac{1}{2}-o(1)\right) \sum_{i=1}^{\ell} (\max(0,d_i-\Delta))^2.\end{aligned}$$

We can now estimate the right hand side of Inequality (5).

$$\begin{aligned}\sum_{i=1}^{n} (\min(d_i,\Delta))^2 &= \sum_{i=1}^{\ell} (\min(d_i,\Delta))^2 + \sum_{i=\ell+1}^{n} (\min(d_i,\Delta))^2 \\ &\ge \sum_{i=1}^{\ell} (\min(d_i,\Delta))^2 + \left(\frac{1}{2}-o(1)\right) \sum_{i=1}^{\ell} (\max(0,d_i-\Delta))^2 \\ &\ge \left(\frac{1}{2}-o(1)\right) \sum_{i=1}^{\ell} (\min(d_i,\Delta))^2 + (\max(0,d_i-\Delta))^2 \\ &\ge \left(\frac{1}{4}-o(1)\right) \sum_{i=1}^{\ell} d_i^2.\end{aligned}$$

Comparing the above inequality with Inequalities (3) and (5), we obtain $\mathrm{cr}(G) \ge \left(\frac{1}{64}-o(1)\right) \sum_{i=1}^{\ell} d_i^2$. □

We can now prove Theorem 1 in general. Order the vertices $v_1, v_2 \dots v_n$ of $G$ such that their degree sequence $d_1, d_2, \dots, d_n$ monotone increases. Let $\ell$ be the integer such that $\sum_{i=1}^{\ell-1} d_i < \frac{4m}{3}$ but $\sum_{i=1}^{\ell} d_i \ge \frac{4m}{3}$. Consider the graph $G_0$ induced by the vertices $v_1, v_2, \dots, v_\ell$. Notice that $G_0$ has at least $\frac{m}{3}$ edges. We choose a family $\mathcal{C}$ of edge-disjoint cycles of length at most $\frac{4}{\epsilon}$ from $G_0$ so that at least half of the edges of $G_0$ are covered by cycles of $\mathcal{C}$. Let $G'$ be a subgraph of $G$ formed by deleting an edge $e_j$ from each cycle $C_j \in \mathcal{C}$. We have deleted at least $\frac{1}{2} \cdot \frac{\epsilon}{4} \cdot \frac{m}{3} = \frac{\epsilon}{24} m$ edges. Let $m'$ be the number of edges of $G'$ and $d_i'$ be the degree of $v_i$ in $G'$. We have $d_i' \le d_i$ for $1 \le i \le \ell$ and $d_i' = d_i$ for $i > \ell$. It follows that $\sum_{i=1}^{\ell-1} d_i' < \frac{4m'}{3}$. By Lemma 4, we have $\mathrm{cr}(G') \ge \left(\frac{1}{64}-o(1)\right) \sum_{i=1}^{\ell} d_i'^2$. If we apply the embedding method to draw graph $G$ based on the drawing of $G'$ with $\mathrm{cr}(G')$ crossings and drawing each $e_j$ along $P_j$, we obtain

$$\mathrm{cr}(G) \le 4\mathrm{cr}(G') + \frac{3}{8} \sum_{i=1}^{\ell} d_i'^2.$$

Hence, we have $\mathrm{cr}(G) \le 4\mathrm{cr}(G') + \frac{3}{8}(64+o(1))\mathrm{cr}(G') = (28+o(1))\mathrm{cr}(G')$. □

## Acknowledgments

We would like to thank Daniel J. Kleitman, János Pach, Rados Radoičić, and Géza Tóth for helpful comments. Thanks to László Szegő for directing us to Nagamochi and Ibaraki's results.

## References

[ACNS82] M. Ajtai, V. Chvátal, M. Newborn, and E. Szemerédi: Crossing-free subgraphs, in *Theory and Practice of Combinatorics,* vol. 60 of Mathematical Studies, North-Holland, Amsterdam, 1982, pp. 9–12.

[ES82] P. Erdős and M. Simonovits: Compactness results in extremal graph theory, *Combinatorica* **2** (1982), 275–288.

[GM90] H. Gazit and G. L. Miller: Planar separators and the Euclidean norm, in *SIGAL International Symposium on Algorithms*, vol. 450 of LNCS, Springer-Verlag, Berlin, 1990, pp. 338–347.

[L83] T. Leighton: *Complexity Issues in VLSI,* MIT Press, Cambridge, MA, 1983,

[L84] T. Leighton: New lower bound techniques for VLSI, *Math. Systems Theory* **17** (1984), 47–70.

[LT79] R. J. Lipton and R. E. Tarjan: A separator theorem for planar graphs, *SIAM J. Appl. Math.* **36** (1979), 177–189.

[NI92] H. Nagamochi and T. Ibaraki: A linear-time algorithm for finding a sparse $k$-connected spanning subgraph of a $k$-connected graph, *Algorithmica* **7** (1992), 583–596.

[PRTT04] J. Pach, R. Radoičić, G. Tardos, and G. Tóth: Improving the Crossing Lemma by finding more crossings in sparse graphs, in *20th ACM Symposium on Computational Geometry*, ACM Press, New York, 2004, pp. 68–75

[PSS96] J. Pach, F. Shahrokhi, and M. Szegedy: Applications of the crossing number, *Algorithmica* **16** (1996), 111–117.

[PST06] J. Pach, J. Solymosi, and G. Tardos: Crossing numbers of imbalanced graphs, lecture presented at *SIAM Conf. Discrete Math. (Victoria, BC, 2006).*

[PST00] J. Pach, J. Spencer, and G. Tóth: New bounds on crossing numbers, *Discrete Comput. Geom.* **24** (2000), 623–644.

[PT00] J. Pach and G. Tóth: Thirteen problems on crossing numbers, *Geombinatorics* **9** (2000), 194–207.

[RT93] B. Richter and C. Thomassen: Minimal graphs with crossing number at least $k$, *J. Combin. Theory Ser. B* **58** (1993), 217–224.

[S00] G. Salazar: On a crossing number result of Richter and Thomassen, *J. Combin. Theory Ser. B* **79** (2000), 98–99.

[SSSV97] F. Shahrokhi, O. Sýkora, L. A. Székely, and I. Vrťo: Crossing numbers: bounds and applications, in *Intuitive geometry* (Budapest, 1995), 179–206, vol. 6 of Bolyai Soc. Math. Stud., János Bolyai Math. Soc., Budapest, 1997.

[S04] L. A. Székely: A successful concept for measuring non-planarity of graphs: the crossing number, *Discrete Math.* **276** (2004), 331–352.

[S04a] L. A. Székely: Short proof for a theorem of Pach, Spencer, and Tóth, in *Towards a theory of geometric graphs*, vol. 342 of Contemp. Math., AMS, Providence, RI, 2004, pp. 281–283.

# How Important Is the "Mental Map"? – An Empirical Investigation of a Dynamic Graph Layout Algorithm

Helen C. Purchase[1], Eve Hoggan[1], and Carsten Görg[2,⋆]

[1] Department of Computing Science, University of Glasgow, Glasgow, UK
[2] College of Computing, Georgia Institute of Technology, Atlanta, USA
{hcp,eve}@dcs.gla.ac.uk, goerg@cc.gatech.edu

**Abstract.** While some research has been performed on the human understanding of static graph layout algorithms, dynamic graph layout algorithms have only recently been developed sufficiently to enable similar investigations. This paper presents the first empirical analysis of a dynamic graph layout algorithm, focusing on the assumption that maintaining the "mental map" between time-slices assists with the comprehension of the evolving graph. The results confirm this assumption with respect to some categories of tasks.

## 1 Introduction

Research on algorithms for the effective layout of graphs has been active for many years - these algorithms have typically been valued for their computational efficiency and the extent to which they conform to pre-defined layout principles. Only recently any empirical work had been conducted to determine the effect of conformance to these principles on user understanding [7]. While static graphs are still applicable and very useful in a variety of situations, recent developments in graph layout research have concentrated on the layout of dynamic graphs, representing changing relational information over time. Examples of applications of such dynamic processes include software engineering visualizations where graphs depict the execution time behavior, changing Internet usage and changes in social network structures. Research on the effects of dynamic layout principles on user understanding is therefore timely.

### 1.1 Dynamic Graph Layout Systems

Empirical investigation of a dynamic graph layout system requires that the system have changeable parameters, so that comparative tests can be performed. Several dynamic graph layout systems are domain specific. For example, Brandes and Corman [1] present a method for visualizing network evolution in which

⋆ The author was supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD).

each modification is shown in a separate layer of 3D representation with edges common to two layers represented as columns connecting the layers. Gevol is a system that visualizes the evolution of software using a modification of a force-directed algorithm [3]. Gevol extracts information about a Java program stored within a CVS version control system and displays it using a temporal graph visualization method. These systems have all been built for a specific purpose or specific type of information visualization. Thus there is little room for the manipulation of layout parameters and the use of different types of algorithms or data.

GraphAEL is a more generic system for graph animation of evolving layouts which has been designed to provide the necessary structure and flexibility for force-directed graph drawing research [5]. The system contains several novel algorithms and visualization techniques, such as force-directed methods in hyperbolic and spherical spaces. However, the research prototype system made available did not have an obvious way of manipulating parameters.

We therefore chose GraphAnimation [4] as the first dynamic graph layout system for our experiments. GraphAnimation provides two layouts: one based on the spring algorithm, and one based on a hierarchical algorithm. It provides easy manipulation of important parameters.

### 1.2 The User's "Mental Map"

Dynamic graph drawing involves laying out graphs which evolve over time by the addition or deletion of edges and nodes at the end of each time period. In creating systems that produce dynamic graph animation, algorithm designers have needed to take into account an additional aesthetic criterion known as "preserving the mental map" [2]. The term mental map refers to the structural cognitive information a user creates internally by observing the layout of the graph [4]. This internal cognitive structure represents the user's underlying understanding of the information. It is important that this remains consistent throughout the dynamic graph animation, otherwise confusion may result. A good preservation of the mental map can help people to understand the application, but a display where it is difficult for the user to maintain a consistent mental map can be misleading.

In developing a dynamic graph layout algorithm, it is of course useful to take advantage of existing static graph layout algorithms. Given that each time-slice could be considered as a static graph, a static graph layout algorithm can be applied to the graph after each update. Using animation between time-slices to show how nodes and edges are moved to the new positions may assist in preserving the mental map over time.

There are therefore two important criteria to consider: the readability of the individual static layouts and the mental map preservation in the sequence of drawings. Thus, some interaction is required between the static layout applied at each time-slice, and the movement of nodes and edges between each time-slice. The readability of the individual graph drawings produced by a static layout algorithm depends on aesthetic criteria such as minimal number of bends, uniform edge lengths, and minimal number of crossings. Preservation of the mental map can be achieved by ensuring that nodes that appear in consecutive graphs in the sequence remain in the same positions, so that they can easily be identified as the same nodes over time.

These two criteria are often contradictory. If each graph is laid out individually, without regard to other graphs in the sequence, its readability may be optimized at the expense of mental map preservation. This may result in an animation where nodes change position radically from time-slice to time-slice. On the other hand, if the node positions are fixed in all graphs, mental map preservation is optimized but the individual layouts may not conform to static graph drawing aesthetics. This may result in an animation where the time-slices themselves are difficult to understand because of, for example, a high number of edge crossings, or several awkward edge bends.

The experiment reported in this paper addresses this contradiction in hierarchical layout of graphs, attempting to determine whether, from a user-understanding point of view, it is preferable to maintain the mental model, conform to static layout aesthetics at each time-slice, or make a compromise between the two.

## 2 Experimental Methodology

The experimental methodology used was based on former static graph layout experiments [7], using an online system to present the graphs, asking the participants to enter their answers to questions on the graphs, and collecting error and response time data. As before, tutorial and worked example material was given at the beginning to familiarize the participants with the experimental tasks, with a ranking and qualitative questionnaire at the end. A within-subjects methodology was used to reduce any subject variability, with the inclusion of practice tasks and randomization controlling for the learning effect. User-controlled rest breaks were included throughout the duration of the experiment, to address any problems of fatigue.

For the empirical analysis of dynamic graphs, additional experimental design decisions needed to be made with regard to the timing of the presentation of the graphs and questions. Through pre-pilot and pilots tests, we determined appropriate animation speed, time per time-slice, the number of times to show the complete animation, pause time before and after the animation, and an appropriate range for the feasible number of time-slices. We also determined when, and for how long, the question should be displayed. These pilots tests were essential, there being no other former experiments of this kind to inform these necessary details for our experimental design.

The experimental system used, DynaGUESS was implemented as a generic system for information visualization experiments. It facilitates easy preparation and customization of online experiments, enabling the experimenter to set parameters regarding timing, randomization, and rest breaks. The dependent variables are the accuracy of the question answers, and the response time.

## 2.1 The GraphAnimation Hierarchical Layout Algorithm

The algorithm evaluated was the hierarchical dynamic layout provided by GraphAnimation [4]. GraphAnimation provides a generic framework for a dynamic layout process. It offers the possibility to trade local layout quality for dynamic stability (preserving the user's mental map) by adjusting a parameter delta that limits the changes between two layouts of succeeding graphs. A large delta offers more freedom to support the optimization of the local layout quality whereas a small delta fortifies the preservation of the mental map.

The hierarchical dynamic layout process in GraphAnimation has many facets. The static part of the layout algorithm used at each time-slice maintains consistent vertical flow of directed edges; it attempts to produce a layout such that all edges point downwards. This is done for each graph of the sequence. The dynamic part of the layout algorithm tries first to keep nodes on the same hierarchical level as in the previous time-slice, and secondly tries to keep nodes in the same horizontal order. There is therefore a possible tension that could lead to a contradiction: the static part of the hierarchical layout algorithm could try to change the vertical level of a node to fulfill the flow constraint, while the dynamic part of the algorithm tries to keep its level and horizontal order. It is the choice of parameters that determines the result if there is a conflict.

GraphAnimation provides two delta parameters by which the extent to which the mental model is maintained between time-slices can be controlled. The two parameters affect the main phases of the hierarchical layout process – the layer assignment and the layer sorting:

– Delta on ranks: maximal number of changes in the layer assignment
– Delta on orders: maximal number of changes in the order of the nodes within one layer

These parameters were used to produce dynamic graph layout animations under three conditions, these were our independent variables. The values of the parameters were chosen after extensive visual analysis.

1. High Delta Condition (both deltas=40): priority is given to layout quality at each time-slice rather than preserving the mental map. This value was defined in pilot tests as the minimum value for all nodes to move freely.
2. Medium Delta Condition (both deltas=20): equal priority is given to layout quality at each time-slice and preserving the mental map.
3. Low Delta Condition (both deltas=0): priority is given to preserving the mental map rather than the layout quality at each time-slice. This is the best this algorithm can do to maintain the mental model.

The details of the hierarchical dynamic layout algorithm used in GraphAnimation are presented in [6].

## 2.2 Experimental Tasks

Three different evolving graphs were created, each with between 14 and 20 nodes, between 15 and 30 edges and 4 changes per time-slice, where a change is an insertion or deletion of a node or an edge. We aimed to keep the size and changes of these graphs as similar as possible, while keeping them distinctive. The graphs represented the changing structure of an imaginary web site over six time periods. Each graph animation was created using GAML, an XML-format based on GraphML which is required by GraphAnimation. Overall, each task had a time limit of 30 seconds. Each time-slice was displayed for 3 seconds with the animation between time-slices set at a speed of 1 second. There was a pause of 4 seconds at the beginning of each task in order to allow the user enough time to read the question before the animation began. A warning beep was sounded 3 seconds before the end of each task to ensure that participants entered their answer before running out of time.

For each of these graphs, three versions were created, one for each of our three experimental conditions. Thus, we had nine graph animations in total. These are here referred to by their graph number (1,2,3), followed by their condition (L, M, H). For example, the high delta version of the third graph is referred to as G3H. The appendix shows the diagrams of all the individual layouts of the three versions of graph 2. The animations of all nine evolving graphs used for the experiment are available at `http://www.cc.gatech.edu/~goerg/GD06`.

Each of these nine animations was presented four times, once for each of four questions, resulting in 36 total tasks. The order of the presentation of tasks was random. Before beginning the actual 36 tasks, the participants were presented with a practice set of 8 tasks.
The questions were:

1. How many new links were added to the site over the years?
2. Which page has been changed the most over the years? (meaning the node which has had the most changes in its incoming and outgoing edges).
3. In which year did the site reduce in size by a quarter? (meaning the year in which the number of nodes reduced by 25%)
4. In which year did [page name] become accessible through only one other page?

The first three questions therefore covered the addition and removal of both nodes and edges, while the fourth one considered the structure of the graph. Each question was multiple choice with the participants indicating one of four possible answers their answer with a check box. The incorrect options that were listed with the correct option were chosen randomly. The worked example and tutorial at the start of the experiment ensured that the participants knew the meaning of the questions.

### 2.3 Experimental Process

20 student participants were recruited; they were typically of a computing science background, although there were also some zoologists, musicians and law students. The experiments were held in 5 sessions over a period of 10 days. Each experiment, including time spent at the beginning on the tutorial and the worked example, and on the questionnaire at the end, took approximately 45

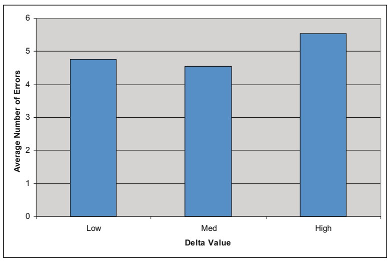
minutes. No problems were experienced during the experiments, and all participants appeared to engage in the task seriously. Participants were paid 5 for their participation.

## 3 Results and Analysis

### 3.1 Analysis by Delta Condition

Our hypothesis was that the extent to which the mental model was maintained between time-slices would affect performance. Intuitively, we felt that a low delta value (which results in an animation that attempts to maintain the mental model) would produce a better performance than a high delta value (which attempts to produce a 'good layout' at each time-slice).

**Performance Data.** The average number of errors and the average response time for the three delta conditions are shown in Figure 1.



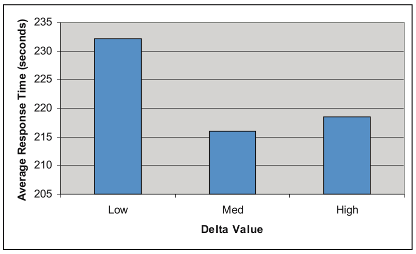


**Fig. 1.** Average Errors and Response Time according to delta. Low delta means few visual changes between time-slices, so a high conformance to maintaining the mental model.

To test the hypothesis, first the significance of the effects of each delta condition were investigated [1]. The statistical analysis used here is a standard two-tailed ANOVA analysis, based on the critical values of the F distribution, with $\alpha = 0.05$. In all cases, conservative readings of the critical values of the F distribution were used.

**Errors.** There are no significant differences between delta conditions in the error data, as $F = 1.366 < (F(2, 57) = 3.23)$.

**Response Time.** There are no significant differences between delta conditions in the response time data, as $F = 2.489 < (F(2, 57) = 3.23)$.

[1] Identifying a statistical significance between results collected from different conditions indicates that the difference between the results can be attributed to the differing nature of these conditions, rather than being due to mere chance. In these experiments, we test whether the probability of the difference being due to chance is less than 0.05.

## 3.2 Further Analysis

At first glance these results appear to disprove our hypothesis. However, further analysis was performed to identify any possible significant differences between delta conditions with respect to a particular question or graph. The analysis above aggregates the data from all four questions and all three graphs: it may be that delta conditions only affect particular questions or graphs.

We anticipated that there would be a difference in the difficulty of the questions, and that the delta conditions may have significant effects when considered independently with respect to the data from each of the different questions. However, having made an effort to keep the three evolving graphs comparable (similar size, similar number of changes per time-slice), we did not anticipate that there would be any difference in difficulty between the graphs. The further analysis took the form of first applying the ANOVA test to the data from the four questions, to see if there were differences in performance between them. If there were significant differences between the performances on the questions, we then analyzed each question separately according to delta condition, applying the Bonferroni correction as appropriate. A similar process was followed for the three graphs.

**Analysis with respect to Question.** There are no significant differences in the number of errors between questions ($F = 0.6540 < F(3,76) = 2.76$).

However, there are significant differences in the response time data between questions ($F = 35.52 > F(3,76)$). Tukey's pairwise analysis showed that the average response time for Q1 was significantly greater than the response times in Q2, Q3, and Q4. It also showed that the response time for Q2 was significantly less than the response time in Q4, and the response time for Q2 approaches significance when compared with Q3. There were no other pairwise differences (see Figure 2).

As expected, the questions were of different difficulty. Separate analyzes were performed on the effect of delta condition on the response times for each of the four different questions.

There were no significant differences in the response time data between delta conditions in Q1 or in Q2.

For Question 3, ("In which year did the site reduce in size by a quarter?"), there are significant differences in the response time data between delta conditions ($F = 5.832 > F(2,57)$). Tukey's pairwise analysis showed that the average response time for low delta value was significantly greater than the response times in both medium and high delta value conditions. There were no other pairwise differences (see Figure 3(a)).

For Question 4 ("In which year did [page name] become accessible through only one other page?"), there were also significant differences in the response time between conditions ($F = 6.903 > F(2,57)$). Tukey's pairwise analysis showed that the average response time for low delta values was significantly greater than the response times in medium and high delta values. There were no other significant pairwise differences (see Figure 3(b)).

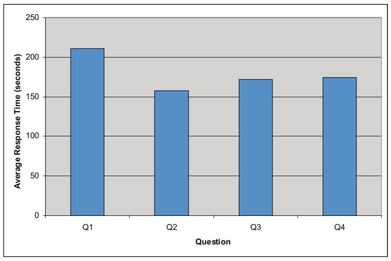
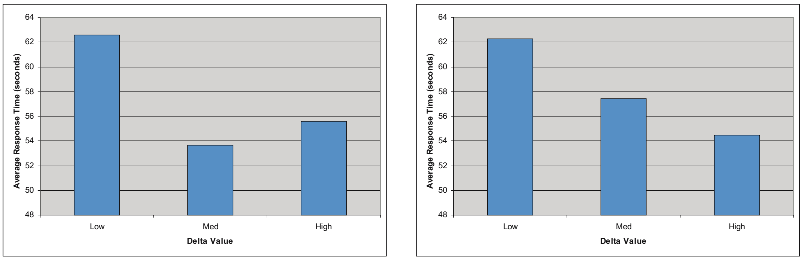


**Fig. 2.** Average Response Time according to Question



(a) Question 3 (b) Question 4

**Fig. 3.** Analysis by delta condition

**Analysis according to Graph.** Our assumption was that the three evolving graphs themselves were of comparable difficulty (in terms of size and number of changes per time-slice), and therefore it was appropriate to aggregate the data from all three of them when testing our initial hypothesis. Our next step was to test this assumption.

There were no significant differences in the number of errors between each graph. However, for response time, there were significant differences between the graphs ($F = 15.98 > F(2, 57)$). Tukey's pairwise analysis showed that the average response times for graphs 2 and 3 were significantly greater than the response times in graph 1. There were no other pairwise differences (see Figure 4).

This indicated that the graphs were unexpectedly of different difficulty. Separate analyzes were therefore performed on the effect of the delta condition on response times for each of the three different graphs. There were no significant differences in the response time data between delta conditions in graph 1, but there were differences between delta conditions in graphs 2 and 3.

On examining the three graphs, we could see no discernible differences between them that would explain the better performance on graph 1, apart from the fact that graph 1 had more levels and was narrower than the other two graphs. The fact that there was no variation in data for graph 1 indicates that it was so easy that a 'floor' effect resulted. We therefore removed graph 1 from our analysis, and repeated the delta condition tests, for each of the four questions.

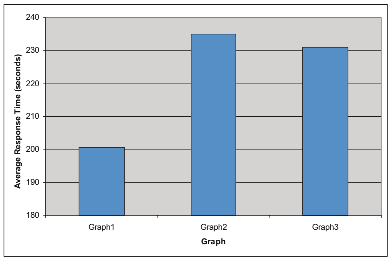


**Fig. 4.** Average Response Time according to graph

The results were more encouraging. There was no significance for question 1, probably due to a 'ceiling effect', as the question was particularly difficult[2]. Significance was found in all three other questions:

- For question 2, low delta produced better performance than high delta for errors ($F = 3.23 > F(2, 57)$)(see Figure 5).
- For question 3, low and medium delta produced better performance than high delta for errors ($F = 3.23 > F(2, 57)$); however, as there was also significance in time data (low and medium delta produced worse performance than high delta ($F = 3.23 > F(2, 57)$), this indicates a correlation between time and error data, so little weight can be attached to these results(see Figure 6).
- For question 4, low delta produced better performance than high delta for errors ($F = 3.23 > F(2, 57)$)(see Figure 7).

### 3.3 Analysis Summary

The significant results with respect to our independent variable, the mental model (delta) condition, were:

- Question 2 errors ("which page has been changed the most over the years"), over graphs 2 and 3: high mental model was important.
- Question 3 response time ("in which year did the site reduce in size by a quarter"), over all graphs: high mental model was not important.
- Question 3 errors ("in which year did the site reduce in size by a quarter"), over graphs 2 and 3: high mental model was not important.
- Question 3 response time ("in which year did the site reduce in size by a quarter"), over graphs 2 and 3: high mental model was important.
- Question 4 response time ("in which year did [page name] become accessible through only one other page?"), over all graphs: high mental model was not important.

[2] A floor/ceiling effect is when the task is so easy/difficult that the manipulation of experimental condition has no effect on the data variation.

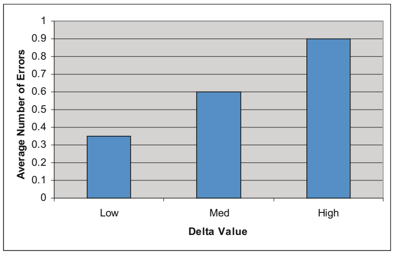


**Fig. 5.** Average Errors according to Q2, for graphs 2 and 3

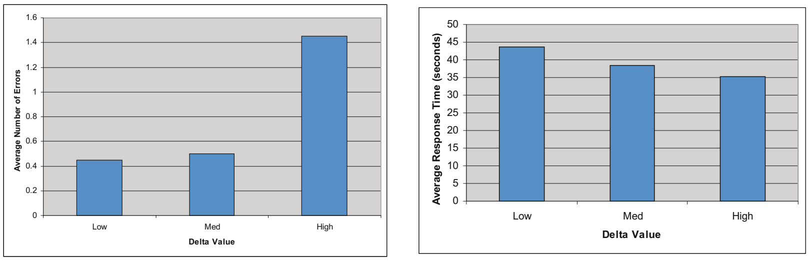


**Fig. 6.** Average Errors and Response Times according to Q3, for graphs 2 and 3

– Question 4 errors (“in which year did [page name] become accessible through only one other page?”), over graphs 2 and 3: high mental model was important.

So, in summary, we cannot make any claims about the usefulness of the mental model for question 1 (as there were no significant results), nor for question 3 (as the significant results are contradictory, indicating a correlation between the dependent variables). However, the data implies that when we remove graph 1 from our analysis (as it was found to be too easy), the mental model did assist with understanding for questions 2 and 4.

– Question 2: “Which page has been changed the most over the years?” In this case, the mental model would have kept the nodes in similar positions, so that the participants could focus on the differing density of edges in the diagram.
– Question 4: “In which year did [page name] become accessible through only one other page?” The mental model would have helped with this question, as the participants would have needed to follow a single node through all time-slices.

Although we cannot make any concrete claims about questions 1 and 3, on examining these questions, we suggest the following:

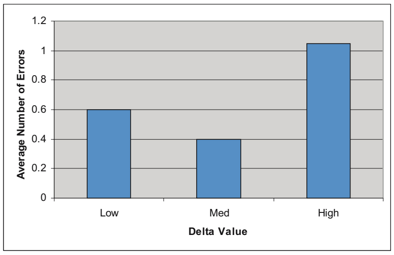


**Fig. 7.** Average Errors according to Q4, for graphs 2 and 3

– Correctly answering question 1 ("How many new links were added to the site over the years?") does not rely on node movement or identification, so the delta value is irrelevant. This was an extremely difficult question, even when multiple choice options are given.
– Similarly, correctly answering question 3 ("In which year did the site reduce in size by a quarter?") only focuses on the number of nodes, not their identification. Thus, the delta value is irrelevant.

We therefore suggest that maintaining the mental map is important for the comprehension of an evolving graph for tasks that require that nodes be identifiable by name, but that it is less important for tasks that focus on edges rather than nodes, or which do not require that nodes be nominally differentiated from each other.

## 4 Conclusions

These conclusions are, of course, limited by our choice of graph parameters: the dynamic algorithm used, the size and structure of the graph, the number of changes per time-slice, and animation speed, and any conclusions can be made only within the context of these choices. Choices that resulted in tasks being too easy (G1) or too difficult (Q1) further constrained the conclusions.

Devising an experimental methodology for dynamic graph layout proved more difficult than for static graph drawings. In particular, we could not rely on experimental tasks based on common graph theoretic questions (for example, "What is the shortest path between two nodes") as such questions to do take into account the temporal, changing nature of the information. It was also very difficult, as shown, to create three dynamic graphs of similar complexity, even when the quantitative descriptors of these graphs were comparable. Other practical experimental issues needed to be addressed carefully, for example, when to display the question, how many times to show the animation, how to present a worked example with explanation of correct answers, what pauses are necessary at the start and end. It became clear that the temporal aspect of a dynamic algorithm makes for a more complex experimental process.

These results are the first empirical evidence that the mental map factor is important in the layout of dynamic graphs and are important despite the restricted size and scope of this experiment. However, the contribution of this paper to graph drawing research is more than merely affirming assumptions made about the importance of maintaining the mental map in the context of two types of tasks. As the first experiments in this area, it represents a significant step forward in human empirical research on the efficacy of graph layout algorithms. There is, of course, substantial future work to be performed in this area - there are other dynamic algorithms to investigate (e.g. force directed, orthogonal), other visualization features that can change over time (for example, node sizes, color), other domains, different sizes and structures of graphs, and alternative tasks. This initial research opens up this wide, rich area for investigation.

The animations of the graphs used for this experiment are available at the following webpage `http://www.cc.gatech.edu/~goerg/GD06`.

## References

1. Ulrik Brandes and Steven R. Corman. Visual unrolling of network evolution and the analysis of dynamic discourse. In *INFOVIS '02: Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)*, page 145, Washington, DC, USA, 2002. IEEE Computer Society.
2. Michael K. Coleman and D. Stott Parker. Aesthetics-based graph layout for human consumption. *Softw. Pract. Exper.*, 26(12):1415–1438, 1996.
3. Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. A system for graph-based visualization of the evolution of software. In *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization*, pages 77–ff, New York, NY, USA, 2003. ACM Press.
4. Stephan Diehl and Carsten Görg. Graphs, they are changing – dynamic graph drawing for a sequence of graphs. In *Proceedings of Graph Drawing 2002*, pages 23–30, London, UK, 2002. Springer-Verlag.
5. Cesim Erten, Philip J. Harding, Stephen G. Kobourov, Kevin Wampler, and Gary V. Yee. Graphael: Graph animations with evolving layouts. In Giuseppe Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 98–110. Springer, 2003.
6. Carsten Görg, Peter Birke, Mathias Pohl, and Stephan Diehl. Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In János Pach, editor, *Graph Drawing, New York, 2004*, pages pp. 228–238. Springer, 2004.
7. Helen C. Purchase. Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, 13(2):147–162, 2000.

# Computing Geometric Minimum-Dilation Graphs Is NP-Hard

Rolf Klein[1,⋆] and Martin Kutz[2]

[1] Institute of Computer Science I
53117 Bonn, Germany
rolf.klein@uni-bonn.de
[2] Max-Planck-Institut für Informatik
66123 Saarbrücken, Germany
mkutz@mpi-inf.mpg.de

**Abstract.** Consider a geometric graph $G$, drawn with straight lines in the plane. For every pair $a, b$ of vertices of $G$, we compare the shortest-path distance between $a$ and $b$ in $G$ (with Euclidean edge lengths) to their actual Euclidean distance in the plane. The worst-case ratio of these two values, for all pairs of vertices, is called the vertex-to-vertex *dilation* of $G$.

We prove that computing a minimum-dilation graph that connects a given $n$-point set in the plane, using not more than a given number $m$ of edges, is an NP-hard problem, no matter if edge crossings are allowed or forbidden. In addition, we show that the minimum dilation tree over a given point set may in fact contain edge crossings.

**Keywords:** dilation, geometric network, plane graph, spanning ratio, stretch factor, NP-hardness.

## 1 Introduction

Given a set $P$ of $n$ points in $\mathbb{R}^2$, one of the basic problems is to find a geometric network $G = (P, E)$ that provides good connections between the points in $P$, at low cost.

Often, the quality of connections is measured as follows. For any two points, $a$ and $b$, of $P$, let $\pi_G(a, b)$ be a shortest path from $a$ to $b$ in $G$, where the length of a path is given by the sum of the Euclidean lengths $|p_i p_{i+1}|$ of its edges $e_i = \{p_i, p_{i+1}\}$. Then

$$\delta_G(a, b) := \frac{|\pi_G(a, b)|}{|ab|}$$

denotes the dilation of $a, b$ in $G$, and

$$\delta(G) := \max_{a \neq b \in P} \delta_G(a, b)$$

⋆ This work was partially supported by the German Research Foundation DFG, grant no. Kl 655/ 14-1/2/3.

is the *vertex-to-vertex dilation* of $G$. This value is also known as the stretch factor or the spanning ratio of $G$; it should not be confused with the geometric dilation that takes all points of the network into account, vertices and interior edge points. In this paper, the *cost* of a network $G$ will be measured by the number of its edges, $|E|$. Alternative cost measures would be the weight, i. e., the sum of all edge lengths, the diameter, the maximum degree, etc.

Clearly, the complete graph over $P$ has optimal dilation 1 but its number of edges is, in general, in $\Omega(n^2)$.[1] On the other hand, spanning trees realize the minimum edge number $n-1$, but they cannot offer good connections. In fact, each tree $T$ containing the vertex set of the regular $n$-gon has dilation $\delta(T) > 1.57$ for $n = 5$, while $\delta(T) \in \Omega(n)$ holds for larger $n$; see Ebbers-Baumann et al. [7] and Aronov et al. [1].

In the framework of spanners it has been shown that one can (almost) combine the merits of both solutions, and efficiently construct networks of dilation $1 + \epsilon$ that have only $O(\epsilon^{-2} \cdot n)$ many edges; for surveys, see the handbook chapter by Eppstein [8] or the forthcoming monograph by Narasimhan and Smid [13].

In this paper we are interested in the computational complexity of constructing good geometric spanners. More precisely, we study the following problems.

**Definition 1.** *Given a finite point set $P$ in the plane, a threshold $\delta > 1$ and a parameter $m \geq |P| - 1$,*

- *the decision problem* DilationGraph *asks, whether there exists a geometric graph with vertex set $P$, that has dilation at most $\delta$ and contains at most $m$ edges*
- *the decision problem* PlaneDilationGraph *asks if there exists a crossing-free geometric graph with the same properties.*

In this paper we prove that both DilationGraph and PlaneDilationGraph are NP-hard. It is interesting to observe that the number $|E|$ of edges we need, in order to prove NP-hardness, is only slightly larger than the minimum number $|P| - 1$. This fits nicely to a recent result by Aronov et al. [1], which also states, in a different way, that few extra edges matter a lot in constructing spanners. They proved that with $n-1+k$ edges, where $0 \leq k < n$, a dilation of $O(n/k + 1)$ can be achieved, which is optimal.

A lot of work has been done on the complexity of finding spanners of low dilation and weight in general graphs. Closely related to our work is a result by Brandes and Handke [2]. Building on previous work by Cai [3], they proved the following fact for weighted graphs. For each fixed rational number $\delta \geq 4$, it is an NP-complete problem to decide if a given graph $H$ contains a planar subgraph $G$, whose weight does not exceed a given bound $W$, such that for any two vertices $v, w$ of $H$ the relation $|\pi_H(v,w)| \leq \delta \cdot |\pi_G(v,w)|$ holds, where the length of a path is given by the sum of its edge weights.

Our paper extends this result to the (more restrictive) geometric case where $H$ is the complete graph over $n$ points and edge weights are Euclidean lengths.

[1] If $m$ points are collinear we need only $m - 1$ edges to build a connecting chain of dilation 1.

It implies one of the recent results by Gudmundsson and Smid [11] that it is NP-hard to find a $\delta$-spanner with $\leq m$ edges in a given geometric graph.

Cai [3] and Cai and Corneil [4] have studied the problem of finding tree spanners of dilation $\leq \delta$ in weighted graphs. They proved that the decision problem is NP-complete for any $\delta \geq 4$, but polynomially solvable for $\delta = 2$, while the case $\delta = 3$ seems to be open. The corresponding geometric problem appears to be rather complicated. This may be due to the surprising fact that the minimum dilation tree over $n$ points in the plane may contain edge crossings, as we shall prove in Section 4, thus solving an open problem stated by Eppstein in [8], p. 444.

While working on the revision of this paper we learned that this fact has also been observed by Cheong et al. [6]. They can show that constructing the minimum dilation tree is NP-hard, too. On the other hand, Eppstein and Wortman [9] showed how to compute, in expected time $O(n \log n)$, a star of minimum dilation for $n$ points.

The rest of this paper is organized as follows. In Section 2 we derive some technical results that will be needed in the reduction; Section 3, the main part, contains the reduction from Partition to DilationGraph; and Section 4 provides a point set whose minimum-dilation tree has a crossing. We close with some open problems in Section 5.

## 2 Technical Lemmata

Throughout this section, $P$ denotes a finite set of points in the plane.

**Definition 2.** *(i) A geometric network $G = (P, E)$ with dilation $\delta(G) \leq \delta$ will be called a $\delta$-*graph *for $P$. A $\delta$-graph $G = (P, E)$ with $|E| \leq m$ edges will be called a $(\delta, m)$-*graph *for $P$.*
*(ii) The $\delta$-*ellipse *of two points $a, b$ in the plane is the set of all points $x$ satisfying $|ax| + |bx| \leq \delta \cdot |ab|$.*

**Lemma 1.** *Each shortest path $\pi_G(a, b)$ in a $\delta$-graph $G$ for $P$ is contained in the $\delta$-ellipse of $a, b$.*

*Proof.* For each vertex $v$ on $\pi_G(a, b)$ the inequality

$$\delta \geq \frac{|\pi_G(a, b)|}{|ab|} \geq \frac{|av| + |vb|}{|ab|}$$

implies that $v$ is contained in the $\delta$-ellipse of $a, b$. □

Now we show how to enforce that certain edges are contained in minimum-weight $(\delta, m)$-graphs for $P$, using geometric properties.

**Lemma 2.** *Let $a, b$ be two points in $P$ such that all points of $P$ that are contained in the $\delta$-ellipse around $a, b$, lie on the line $L$ through $a$ and $b$, but not between $a$ and $b$. Then the edge $ab$ is contained in any $(\delta, m)$-graph for $P$ of minimum weight.*

*Proof.* Assume for contradiction that there is no edge between $a$ and $b$ in some minimum-weight $(\delta, m)$-graph $G$ for $P$. Let $\pi$ be a shortest path from $a$ to $b$ in $G$. By Lemma 1 and by assumption, all vertices of $\pi$ lie on $L$, but none of them between $a$ and $b$. Let $q_1, \ldots, q_t$ be the sequence of these vertices, sorted by their order on $L$ (which is not the order in which they occur on $\pi$; for example, $q_1$ need not be equal to $a$ or $b$!). Let $G'$ result from $G$ by replacing the edges of $\pi$ with the edges $(q_i, q_{i+1})$ for $1 \leq i \leq t-1$. This transformation does not increase the $G$-distance of any point pair in $P$ because for each edge of $\pi$ there exists a concatenation of collinear edges in $G'$. Thus, $G'$ is a $(\delta, m)$-graph for $P$. On the other hand, it is clearly of smaller weight than $G$—a contradiction. □

## 3 The Reduction

We shall prove the NP-hardness of DilationGraph and PlaneDilationGraph by a reduction from the Partition problem:

Given a set $S$ of $n$ positive integers with $\sum_{r \in S} r$ even, decide whether there exists a subset $T \subseteq S$ such that $\sum_{r \in T} r = \sum_{r \in S \setminus T} r$.

Presented with an instance of Partition involving $n$ integers, we are going to construct a planar point set $P$ of size $5919 \cdot n - 4214$. Roughly, this point set $P$ results from densely sampling a plane straight line drawing that consists of $O(n)$ segments, as shown in Figure 6. It takes $|P| - n - 2$ small edges to connect adjacent sample points on the long segments. If a partition exists for the given instance, we can carefully add $2n$ further edges, two in each of the $n$ bubbles depicted in Figure 6, to ensure that the resulting graph is of dilation $\leq 7$.

Conversely, suppose that the class of $(7, |P|+n-2)$-graphs for $P$ is not empty; then it contains a graph of minimum weight. By Lemma 2, $|P|-n-2$ of its edges are forced to form the long segments. Since the dilation is $\leq 7$, the remaining $2n$ edges must be placed inside the bubbles, and their positions must correspond to a partition of the integer set $S$. In particular, the graph must be plane. These properties will become evident below.

Our construction depends on $n$ and on the size of the maximum element $r_{\max}$ of $S$, and it uses some scaling factors that will be stated as negative powers of 10. Let $\lambda$ be the smallest integer greater than or equal to 8 for which

$$\max(10^5 r_{\max}, 2nr_{\max}^2) < 10^\lambda$$

holds. In particular, this ensures $r \cdot 10^{-\lambda} < 10^{-5}$ for all $r \in S$. Observe that exponent $\lambda$ depends linearly on the bit length of the partition instance (which is bigger than $n$ and the bit length of $r_{\max}$).

The basic idea behind our reduction is to arrange points along two long U-shaped paths like in Figure 1.

The vertical baselines of the two U's will be interrupted by horizonta gadgets—the bubbles depicted in Figure 6, one for each element $r \in S$. Each gadget will stretch horizontally over both U's. It can offer a short cut of $\approx r \cdot 10^{-\lambda}$ to either the left U, or to the right U—but not to both, since this would cause the inner part of the gadget to have a dilation $> 7$.

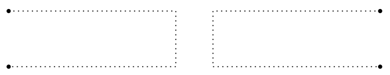


**Fig. 1.** A double U of intended dilation 7

Consequently, both U's can receive the same total short cut, approximately

$$P := \frac{1}{2} \sum_{r \in S} r \cdot 10^{-\lambda},$$

if and only if set $S$ admits a partition. After carefully adjusting the lengths of the horizontal edges of the U's, this will become equivalent to both $U's$ having a dilation $\leq 7$ at their respective endpoints.

## 3.1 The Choice Gadget

The core part of our reduction is the choice gadget, which realizes the selection of an element $r \in S$ for the subset $T$ from the given PARTITION instance. Basically, such a gadget consists simply of two horizontal densely sampled lines, with a larger gap in the upper row. Figure 2 shows the relative lengths and distances of the respective parts. The three segments $au, vc$, and $bd$ are sampled with a regular spacing of $10^{-2}$, giving a total of 1703 points.

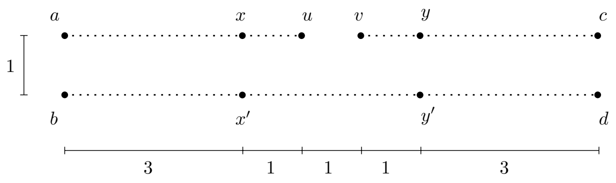


**Fig. 2.** The choice gadget (with lengths annotated)

Assume we want to connect this point set to a tree, i.e., with $|P| - 1$ edges so that the dilation is exactly 7—the same threshold as is intended on the global scale. By Lemma 2, we know that, because the three line segments are very densely sampled, in a minimum weight graph we must have an edge between any pair of direct neighbors on those segments. This leaves just two more edges for connecting the segments.

There must be at least one edge from top to bottom, so let's assume that there is such an edge, $e$, incident to a point on the line between $a$ and $u$. Then the second edge cannot touch segment $au$, too, because otherwise the resulting path from $c$ to $d$ would be more than 10 units long. Hence, there must also be an edge $f$ between the upper right segment $vc$ and the bottom segment $bd$.

Taking the dilation of the points $u$ and $v$ into consideration, too, we see that a dilation of 7 can only be achieved if the edges $e$ and $f$ connect the points $x, x'$

and $y, y'$, respectively. Shifting these links to the left or right would impair the dilation between one of the pairs $\{a, b\}$, $\{c, d\}$, or $\{u, v\}$. Tilting $e$ or $f$ would have a similar effect because slanted edges are longer than vertical ones.

However, allowing only one solution is not what we desire. For the configuration to work as a choice gadget, we apply two minor modifications to leave some very restricted room for the precise placement of the links $e$ and $f$. On the bottom line we introduce two extra points: a point $\hat{x}$ exactly $r \cdot 10^{-\lambda}$ to the left of $x'$, where $r$ is the given integer from the set $S$ that we want to encode into this gadget, and a point $\hat{y}$ located $r \cdot 10^{-\lambda}$ to the right of $y'$.[2]

Moreover, we shift the middle points $u$ and $v$ on the upper line slightly outwards, each by a distance of $r \cdot 10^{-\lambda-1}$, so that the width of the gap increases by $2r \cdot 10^{-\lambda-1}$. See Figure 3 for a close-up on the relevant parts of the modified point set.

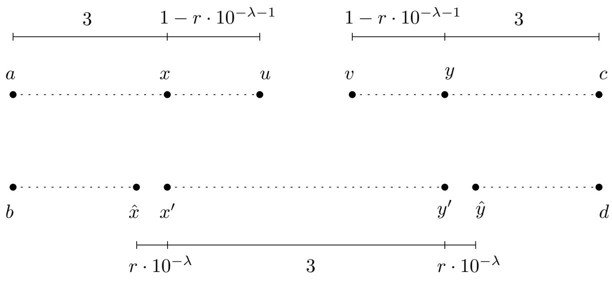


**Fig. 3.** A non-proportional drawing of the crucial parts of the final choice gadget

What is the effect of these modifications? First of all, it is easy to see that a dilation-7 tree on the new point set cannot deviate much from the optimum that we have determined for the original set above. There still have to be all edges between direct neighbors along the three segments and there have to be the two edges $e$ and $f$ somewhere around $x, x', y$, and $y'$. Increasing the distance from $u$ to $v$ by $2r \cdot 10^{-\lambda-1}$ does not give us enough room to fix $e$ or $f$ to any upper vertex other than the designated $x$ and $y$, respectively, since any shift of these edges would immediately increase one of the relevant distances by $10^{-2}$. Moving the upper endpoint of $e$ some $k$ points to the left while moving the lower endpoint $k$ steps to the right would not work either because it would increase the edge length to at least $\sqrt{1^2 + .02^2} \approx 1.0002$, which yields an increase in the distance that cannot be compensated by the comparatively small shift of $u$ and $v$.

It turns out that the only freedom for placing the connections $e$ and $f$ lies in choosing $x'$ or $\hat{x}$, respectively $y'$ or $\hat{y}$, as their lower endpoints. What if we connect $e$ to $\hat{x}$ but $f$ to $y'$, i.e., the left connection tilts slightly to the left, while the right one stays perfectly vertical? The resulting dilation of $u$ and $v$ would then be

[2] Just for the record: we have now used 1705 points per choice gadget.

$$\frac{7 - 2r \cdot 10^{-\lambda-1} + r \cdot 10^{-\lambda} + \sqrt{1 + (r \cdot 10^{-\lambda})^2} - 1}{1 + 2r \cdot 10^{-\lambda-1}}.$$

The square root, which is due to the slope of the edge $e$, minus 1 is smaller than the $2r \cdot 10^{-\lambda-1}$ term so that this expression is smaller than

$$\frac{7 + r \cdot 10^{-\lambda}}{1 + \frac{1}{5}r \cdot 10^{-\lambda}} = 7 \cdot \frac{7 + r \cdot 10^{-\lambda}}{7 + \frac{7}{5}r \cdot 10^{-\lambda}} < 7$$

for such $r$.

Thus we see that slanting one of the two edges $e$ and $f$ outward does not create a dilation of more than 7 in the gadget. (It is obvious that the vertices $u$ and $v$ form the dilation-critical pair in this configuration, all other pairs having better dilation.)

But if we slanted both edges outward, connecting them to $\hat{x}$ and $\hat{y}$, we would get a dilation of

$$\frac{7 - 2r \cdot 10^{-\lambda-1} + 2r \cdot 10^{-\lambda} + 2\sqrt{1 + (r \cdot 10^{-\lambda})^2} - 2}{1 + 2r \cdot 10^{-\lambda-1}},$$

which is lower bounded by 7, as straightforward calculation shows. Therefore it is not possible to slant both edges outward without raising the dilation above 7.

This concludes the construction of our choice gadget. For a given integer $r$, we built it in such a way that either the path from $a$ to $b$ or that from $c$ to $d$ can be reduced by

$$1 + r \cdot 10^{-\lambda} - \sqrt{1 + r^2 \cdot 10^{-2\lambda}} \;\geq\; r \cdot 10^{-\lambda} - r^2 \cdot 10^{-2\lambda}.$$

We will now insert such gadgets into the big picture of Figure 1 by connecting their left endpoints $a, b$ to the left U there and the right endpoints $c, d$ to the right U.

## 3.2 Linking the Choice Gadgets

For a Partition instance $S$ of size $n$, we have to build $n$ individual choice gadgets, one for each $r \in S$. We arrange all these gadgets vertically, one below the other, forcing their left and right endpoints to get connected by paths. Figure 4 shows two choice gadgets linked at their endpoints.

The vertical distance between two choice gadgets is three on each side, left and right, we bridge this gap by a column of points with a regular spacing of $10^{-2}$. The highest of these points is placed exactly 1/10 below the endpoint of the upper gadget and symmetrically, the lowest point sits 1/10 above the endpoint of the bottom gadget.

Since the internal spacing of such a link is by a factor of 10 smaller than the gaps to the endpoints, Lemma 2 applies and tells us that any dilation-7 graph

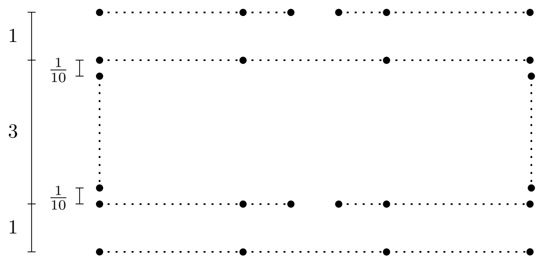


**Fig. 4.** Two choice gadgets linked at their endpoints

must connect these 281 points by the canonical 280 length-$10^{-2}$ edges. The only question that remains is, how such a vertical segment connects to the gadgets above and below. It clearly has to establish connections there to avoid extremely large dilation between the endpoint of the gadget and the endpoint of the vertical link, which are only 1/10 apart.

Globally, only a connection that minimizes the length of the resulting path along the vertical link and through the gadgets can lead to a dilation $\leq 7$ of the corresponding $U$. Such connections will look roughly like the one shown in Figure 5. The further away the slanted edge is from the endpoints $c$ and $w$, the better the short cut. However, the diagonal is also restricted by the dilation between $c$ and $w$.

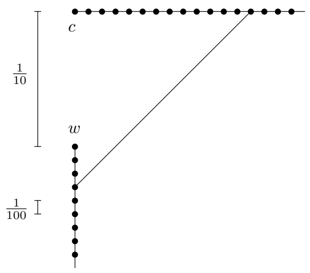


**Fig. 5.** Efficiently connecting around a corner

It is not hard to find out that an almost perfect 45°-connection yields the optimal tradeoff between short-cut effect and $c$-$w$ dilation. One verifies that connecting the 23th point to the right (counting $c$ as the first) with the 15th on the left (counting $w$ as the first) yields the best[3] short cut, giving a $c$-$w$ dilation of

$$\left(\frac{22}{100}+\frac{14}{100}+\sqrt{\left(\frac{22}{100}\right)^2+\left(\frac{14}{100}+\frac{1}{10}\right)^2}\right)\Big/\frac{1}{10} \approx 6.856 \;<\; 7.$$

[3] One could as well connect the 25th point to the 13th point and obtain the same values for dilation and shortcut; minimum dilation graphs need not be unique.

On path length we save at each corner

$$\left(\frac{22}{100}+\frac{14}{100}+\frac{1}{10}-\sqrt{\left(\frac{22}{100}\right)^2+\left(\frac{14}{100}+\frac{1}{10}\right)^2}\right) = \frac{23}{50}-\frac{\sqrt{265}}{50} \approx 0.134.$$

## 3.3 Putting Everything Together

We are now prepared to assemble all $n$ choice gadgets and the two big U's into one big point set, as shown in Figure 6.

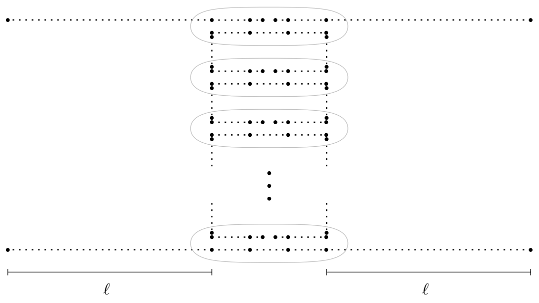


**Fig. 6.** The whole reduction in one picture; $n$ choice gadgets are marked with bubbles

Let us consider again how many edges we are going to allow for this point set, in order to enforce a dilation very close to 7. Each of the segments in Figure 6 shall form a long path. Taken alone, every choice gadget shall form a tree, with all its points connected but without any internal cycles. Cycles are only created by the links between gadgets. Precisely, every pair of link paths induces exactly one cycle. For a total number of $|P|$ points, we thus fix the number of edges to

$$m = |P| - 1 + (n-1) = |P| + n - 2,$$

where $n$ again denotes the number of gadgets.

It remains to calibrate the length $\ell$ of the horizontal segments of the two U's in such a way that only a fair split of the total "short cut potential"

$$P = \sum_{r \in S} r \cdot 10^{-\lambda},$$

can result in a dilation $\leq 7$ between the endpoints of each $U$.

First, let us assume that a partition $S = T \cup (S \setminus T)$ is possible such that the sum of the elements of $T$ equals the sum over the elements of $S \setminus T$. In the choice gadgets associated with $T$ the left edges are slanted, whereas in the other gadgets the right edges are slanted. By the results of Subsections 3.1 and 3.2, the path through the left $U$ has total length

$$\leq\ 2\,\ell + n7 - \sum_{r\in T}(r\cdot 10^{-\lambda} - r^2\cdot 10^{-2\lambda}) + (n-1)\Bigg(3 - 2(\,\frac{23}{50} - \frac{\sqrt{265}}{50})\Bigg) \quad (1)$$

$$\leq\ 2\,\ell + (n-1)\Bigg(10 - \frac{23-\sqrt{265}}{25}\Bigg) + 7 - \frac{1}{2}P + nr_{\max}^2 10^{-2\lambda} \quad (2)$$

Exactly the same upper bound applies to the length of the path through the right U. We want the value of (2) to be at most 7 times the Euclidean distance of the endpoints of a U, which equals $n\cdot 1 + (n-1)\cdot 3 = 4n-3$, by construction. This can be achieved by letting

$$\ell\ \leq (n-1)\Bigg(9 + \frac{23-\sqrt{265}}{50}\Bigg) + \frac{1}{4}P - \frac{1}{2}nr_{\max}^2 10^{-2\lambda}. \quad (3)$$

Now let us assume that no partition of $S$ is possible. Let $L$ denote the set of all gadgets whose left edges are slanted, and let $R$ be the set of choice gadgets with a slanted right edge. If a gadget lies in both, $L$ and $R$, it causes a dilation $> 7$. So assume that $L$ and $R$ are disjoint. For one of these sets—say: $L$— must $\sum_{r\in L} r < \frac{1}{2}\sum_{r\in S} r$ hold. Then the total length of the path through the left U is at least

$$2\,\ell + n7 - \sum_{r\in L}(r\cdot 10^{-\lambda} - \sqrt{1 + r^2\cdot 10^{-2\lambda}}) + (n-1)\Bigg(3 - \ \frac{23-\sqrt{265}}{25}\Bigg) \quad (4)$$

$$\geq 2\,\ell + n7 - \frac{1}{2}P + 1\cdot 10^{-\lambda} + (n-1)\Bigg(3 - \ \frac{23-\sqrt{265}}{25}\Bigg) \quad (5)$$

$$\geq 2\,\ell + (n-1)\Bigg(10 - \frac{23-\sqrt{265}}{25}\Bigg) + 7 - \frac{1}{2}P + 10^{-\lambda}. \quad (6)$$

The left U will give a dilation $> 7$ if the value of (6) exceeds 7 times the distance of its endpoints, that is, if

$$\ell\ > (n-1)\Bigg(9 + \frac{23-\sqrt{265}}{50}\Bigg) + \frac{1}{4}P - \frac{1}{2}10^{-\lambda}. \quad (7)$$

In order to fulfill conditions (3) and (7) we use Newton's method to approximate $\sqrt{265}$, the only irrational number involved, by a rational number $q$ to an error smaller than

$$10^{-(2\lambda+2)} < \frac{10^{-(\lambda+2)}}{2n} < \frac{10^{-\lambda} - nr_{\max}^2 10^{-2\lambda}}{100n};$$

these estimates hold due to the choice of $\lambda$. This takes a number of iterations logarithmic in $\lambda$. Then, we compute $\ell$ from (3), read as an equality, after substituting the root by $q$.

Observe that the coefficient of $n-1$ in (3) is $\approx 9.1344$, and that the additive terms are bounded. If we split $\ell$ into $913(n-1)$ equal pieces, each of them has a (rational) length close enough to $10^{-2}$ to make Lemma 2 work for the horizontal segments of the big U. This takes $4 \cdot 913 \cdot (n-1)$ additional points. Now the definition of point set $P$ is complete.

It is clear that that the description complexity of the constructed point set $P$ is polynomial in the size of the PARTITION instance $S$, and that all computations can be carried out by a Turing machine. Moreover, $S$ admits a partition if and only if there exists a graph of dilation $\leq 7$ over $P$ with $|P| + n - 2$ edges; and each such graph of minimum weight must be plane. Thus, we have shown the following.

**Theorem 1.** *The decision problems* DILATIONGRAPH *and* PLANEDILATION-GRAPH *are NP-hard.*

By counting the numbers of points and edges introduced in our construction, one verifies that even the following problem is NP-hard. Given a set of $k$ points, is there a plane graph of dilation $\leq 7$ over these points that contains at most $\frac{5920}{5919} \cdot k - \frac{7624}{5919}$ many edges?

## 4 Crossings in the Minimum Dilation Tree

It is well-known that a (Euclidian) minimum spanning tree on a point set in the plane cannot have any edge crossings. In [8, p. 444], Eppstein asks whether this is also the case for minimum-dilation trees.

We give a negative answer to this question. In fact, it is not too hard to verify that any spanning tree on the 7-point set in Figure 7 has a dilation of at least 2 and that the two trees that attain this value both contain an edge crossing.

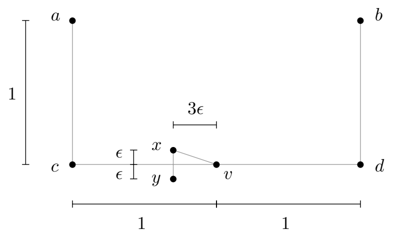


**Fig. 7.** A 7-point set whose minimum-dilation trees have a crossing

The reason for the inevitable crossing in our example lies in the overlay of two structures on different scales, that is, of a large U on *acvdb* and a tiny hook on *xyv*. However, we could easily draw a crossing-free tree on the points of Figure 7 if we were allowed to produce a slightly suboptimal dilation.

## 5 Open Problems

Is there a constant $c > 1$ such that for every point set $P$, there exists a crossing-free spanning tree on $P$ whose dilation is no more than $c$ times that of a minimum-dilation tree? How fast can such tree be computed?

In view of the recent result by Mulzer and Rote [12] on the minimum weight triangulation, is it also NP-hard to construct the minimum dilation triangulation of a given point set?

## Acknowledgement

The authors would like to thank three anonymous referees for their valuable comments.

## References

1. B. Aronov, M. de Berg, O. Cheong, J. Gudmundsson, H. Haverkort, and A. Vigneron. Sparse Geometric Graphs with Small Dilation. 16th International Symposium ISAAC 2005, Sanya. In X. Deng and D. Du (Eds.) Algorithms and Computation, Proceedings, pp. 50–59, LNCS 3827, Springer-Verlag, 2005.
2. U. Brandes and D. Handke. NP-Completeness Results for Minimum Planar Spanners. *Discrete Mathematics and Theoretical Computer Science* 3, pp. 1–10, 1998.
3. L. Cai. NP-Completeness of Minimum Spanner Problems. *Discrete Applied Math.* 48, pp. 187–194, 1994.
4. L. Cai and D. Corneil. Tree Spanners. *SIAM J. of Discrete Math.* 8, pp.359–387, 1995.
5. P. Callahan and S. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM* 42(1), pp.67–90, 1995.
6. O. Cheong, H. Haverkort, and M. Lee. Computing a Minimum-Dilation Spanning Tree is NP-hard, Manuscript, 2006
7. A. Ebbers-Baumann, A. Grüne, and R. Klein. On the Geometric Dilation of Finite Point Sets. 14th International Symposium ISAAC 2003, Kyoto. In T. Ibaraki, N. Katoh, and H. Ono (Eds.) Algorithms and Computation, Proceedings, pp. 250–259, LNCS 2906, Springer-Verlag, 2003. *Algorithmica* 44, pp. 137–149, 2006.
8. D. Eppstein. Spanning Trees and Spanners. In Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, editors, pp. 425-461. Elsevier, 1999.
9. D. Eppstein and K. A. Wortman. Minimum Dilation Stars. Proc. 21st ACM Symp. Comp. Geom. (SoCG), Pisa, 2005, pp. 321-326.
10. S. Fekete and J. Kremer. Tree Spanners in Planar Graphs. *Discrete Applied Mathematics* 108(1-2), pp. 85–103, 2001.
11. J. Gudmundsson and M. Smid. On Spanners of Geometric Graphs. 10th Scandinavian Workshop on Algorithmic Theory (SWAT'06). In L. Arge and R. Freivalds (Eds.), pp. 388–399, LNCS 4059, Springer-Verlag, 2006.
12. W. Mulzer and G. Rote. Minimum weight triangulation is NP-hard. Proc. 22nd Annual ACM Symp. Comp. Geom. (SoCG), Sedona, 2006 pp. 1–10.
13. G. Narasimhan and M. Smid. Geometric Spanner Networks. Cambridge University Press, to appear.

# Chordal Graphs as Intersection Graphs of Pseudosegments

Cornelia Dangelmayr[1] and Stefan Felsner[2]

[1] Freie Universität Berlin
Institut für Mathematik II,
Arnimallee 3,
14195 Berlin, Germany
dangel@math.fu-berlin.de
[2] Technische Universität Berlin
Institut für Mathematik, MA 6-1
Strasse des 17. Juni 136
10623 Berlin, Germany
felsner@math.tu-berlin.de

**Abstract.** We investigate which chordal graphs have a representation as intersection graphs of pseudosegments. The main contribution is a construction which shows that all chordal graphs which have a representation as intersection graph of subpaths on a tree are representable. A family of intersection graphs of substars of a star is used to show that not all chordal graphs are representable by pseudosegments.

## 1 Introduction

A set of Jordan arcs induces a graph $G$, the vertices of $G$ are the Jordan arcs and two Jordan arcs are adjacent if and only if they have non-empty intersection. Graphs representable in this model are called *string graphs*. String graphs are quite complicated, only recently it has been shown that membership in this class is at least decidable [7,11]. All planar graphs are string graphs, this follows e.g. from Koebe's circle representations.

If any two Jordan arcs in a set of nontagent Jordan arcs are either disjoint or have exactly one point of intersection, we call this as *set of pseudosegments* and the resulting class of graphs as *intersection graphs of pseudosegments*. We denote this class by PSI. Deciding whether a given graph is a PSI-graph is known to be NP-complete [8]. A main open problem is whether all planar graphs belong to PSI (see [1,2,3,4,5]). Actually, most of the cited references discuss the class of segment intersection graphs, the problem whether all planar graphs are representable is open with respect to this class as well.

There are some classes of graphs where PSI-representations are trivial (e.g. permutation graphs) or very easy to find (e.g. interval graphs). A large superclass of interval graphs is the class of chordal graphs. In this paper we investigate chordal graphs in view of their representability as intersection graphs of pseudosegments.

## 2 Basic Definitions and Results

**Definition 1.** *A graph $G = (V_G, E_G)$ is a* VPT-graph[1]*, if there exists a tree $T = (V_T, E_T)$ and a set $\mathcal{P}$ of paths in $T$ such that there is a mapping $v \to P_v \in \mathcal{P}$ with the property that $vw \in E_G$ iff $P_v \cap P_w \neq \emptyset$. Such a pair $(T, \mathcal{P})$ is said to be a VPT-representation of $G$ .*

These graphs have been introduced by Gavril [6] who gave a recognition algorithm and have been studied continuously since then. Monma and Wei [10] give some applications and many references. Our first result is:

**Theorem 1.** *Every VPT-graph has a PSI-representation.*

The proof of the theorem is given in Section 3. At this point we content ourselves with an indication that the result is not as trivial as it may seem at first glance. Let a VPT-representation $(T, \mathcal{P})$ of a graph $G$ be given. If we fix a plane embedding of the tree we obtain an embedding of each of the paths $P_v$ corresponding to $v \in V_G$, this embedded path $P_v$ is a Jordan arc. The first idea for converting a VPT-representation into a PSI-representation would be to slightly perturb $P_v$ into a pseudosegment $s_v$ and make sure that paths with common vertices intersect exactly once and are disjoint otherwise. Figure 1 gives an example of a set of subpaths which can't be perturbed such that they give a PSI-representation of the corresponding subgraph.

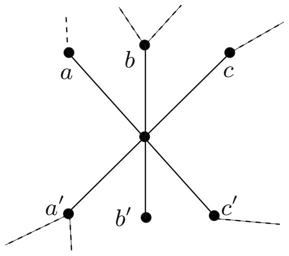


**Fig. 1.** The paths $P(a, a')$, $P(b, b')$ and $P(c, c')$ can not locally be perturbed into a PSI-representation

A superclass of VPT-graphs is the class of vertex intersection graphs of subtrees of a tree. Graphs with such a representation are exactly the chordal graphs.

Clearly, every cycle $C_n, n \in \mathbb{N}$ has a representation as intersection graph of pseudosegments, whereas $C_n$ is not chordal for $n \geq 4$. Hence the class PSI is not contained in the class of chordal graphs.

**Theorem 2.** *There are chordal graphs that are not in the class PSI.*

The proof of the theorem is given in Section 4. There we use geometric arguments to show that the graph $K_n^3$ defined below is not in PSI.

[1] VPT is mnemonic for *v*ertex intersection graph of *p*aths on a *t*ree.

$K_n^3$ has two groups of vertices, vertices $v_1, .., v_n$ that are a clique and a vertex $v_{i,j,k}$ for every triple $\{i, j, k\} \subset [n]$. These triple-vertices are adjacent only to the three corresponding vertices $v_i$, $v_j$ and $v_k$, hence form an independent set. The graph $K_n^3$ can be represented as intersection graph of subtrees of a star $S$ with $\binom{n}{3}$ leaves. The leaves correspond to the vertices $v_{i,j,k}$ and these vertices are represented by trivial trees with only one node. A vertex $v_i$ of the complete graph is represented by the star connecting to all leaves of triples containing $i$. This representation shows that $K_n^3$ is chordal.

The central node of the star $S$ has high degree. If we take a path of $\binom{n}{3}$ nodes and attach a leaf-node to each node of the path we obtain a tree $T$ of maximum degree three such that the graph $K_n^3$ can be represented as intersection graph of subtrees of $T$. Actually the tree $T$ and its subtrees are caterpillars of maximum degree three.

These remarks show that the positive result of Theorem 1 and the negative of Theorem 2 only leave a small gap for questions: If the subtrees in a tree representation of a chordal graph are paths we have a PSI-representation. If we allow the subtrees to be stars, or caterpillars of maximum degree three, there need not exist a PSI-representation. We don't know the answer if the subtrees in the representation only have a small constant number of leaves. When the number of leaves is allowed to get as big as $741 = \binom{39}{2}$ the graph $K_{39}^3$ again shows that the graphs are not in PSI. We think that one node of degree three in each subtree of a tree representation is sufficient to get graphs which are not PSI-representable. More precisely, let $S_n$ be the chordal graph whose vertices are represented by all substars with three leaves and all leaves on a star with $n$ leaves.

*Conjecture 1.* For $n$ large enough, $S_n$ is not a PSI-graph.

## 3 Proof of Theorem 1

### 3.1 Preliminaries

Let $P$ be a path in a tree $T$ with endpoints $a$ and $b$, this is denoted $P = P(a, b)$. If both endpoints of $P$ are leaves of $T$ we call $P$ a *leaf-path.*

**Lemma 1.** *Every VPT-graph has a representation $(T, \mathcal{P})$ such that all paths in $\mathcal{P}$ are leaf-paths and no two vertices are represented by the same leaf-path.*

*Proof.* Let an arbitrary VPT-representation $(T, \mathcal{P})$ of $G$ with $P_v = P(a_v, b_v)$ for all $v \in V_G$ be given. Now let $\overline{T}$ be the tree obtained from $T$ by attaching a new node $\overline{x}$ to every node $x$ of $T$. Representing the vertex $v$ by the path $\overline{P_v} = P(\overline{a_v}, \overline{b_v})$ in $\overline{T}$ yields a VPT-representation of $G$ using only leaf-paths.

The definition of VPT-graphs as intersection graphs immediately implies that every induced subgraph of a VPT-graph is a VPT-graph as well. The lemma together with this observation shows that Theorem 1 is implied by the following:

**Theorem 3.** *Given a tree $T$ we let $G$ be the VPT-graph whose vertices are in bijection to the set of all leaf-paths of $T$. The graph $G$ has a PSI-representation* *with pseudosegments $s_{i,j}$ corresponding to the paths $P_{i,j} = P(l_i, l_j)$ in $T$. In addition there is a collection of pairwise disjoint disks, one disk $R_i$ associated with each leaf $l_i$ of $T$, such that:*

(a) *The intersection $s_{i,j} \cap R_k \neq \emptyset$ if and only if $k = i$ or $k = j$. Furthermore the intersections $s_{i,j} \cap R_i$ and $s_{i,j} \cap R_j$ are Jordan curves.*

(b) *Any two pseudosegments intersecting $R_i$ cross in the interior of this disk.*

We will prove Theorem 3 by induction on the number of inner nodes of tree $T$. The construction will have multiple intersections, i.e., there are points where more than two pseudosegments intersect. By perturbing the pseudosegments participating in a multiple intersection locally the representation can easily be transformed into a representation without multiple intersections.

## 3.2 Theorem 3 Is True for Trees with One Inner Node

Let $T$ have one inner node $v$ and let $L = \{l_1, .., l_m\}$ be the set of leaves of $T$. The subgraph $H$ of $G$ induced by the set $\mathcal{P} = \{P(l_i, l_j) \mid l_i, l_j \in L, l_i \neq l_j\}$ of leaf-paths is a complete graph on $\binom{m}{2}$ vertices, this is because every path in $\mathcal{P}$ contains $v$.

Take a circle $\gamma$ and choose $m$ points $c_1, .., c_m$ on $\gamma$ such that the set of straight lines spanned by pairs of different points from $c_1, .., c_m$ contains no parallel lines. For each $i$ choose a small disk $R_i$ centered at $c_i$ such that these disks are disjoint and put them in one-to-one correspondence with the leaves of $T$. Let $s_{i,j}$ be the line connecting $c_i$ and $c_j$. If the disks $R_k$ are small enough we clearly have :

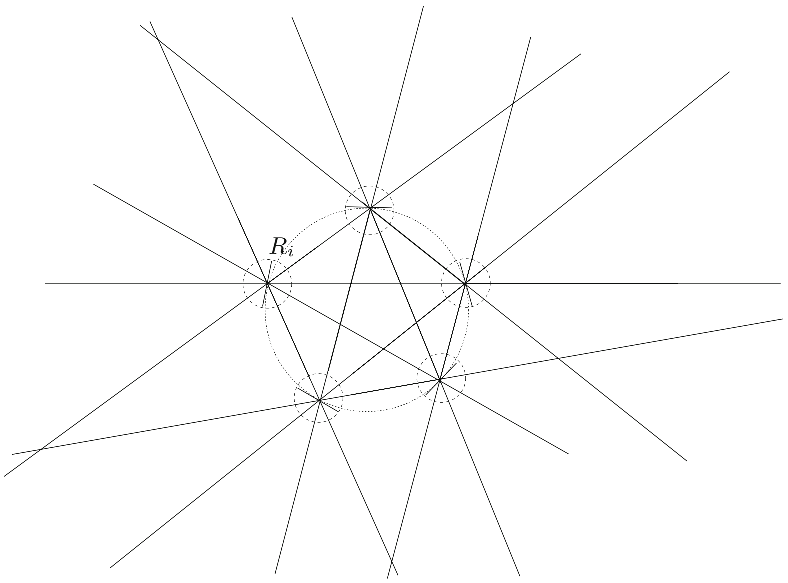


**Fig. 2.** The construction for the star with five leaves

(a) The line $s_{i,j}$ intersects $R_i$ and $R_j$ but no further disk $R_k$.

(b) Two lines $s_I$ and $s_J$ with $I \cap J = \{i\}$ contain corner $c_i$, hence $s_I$ and $s_J$ cross in the disk $R_i$.

Prune the lines such that the remaining part of each $s_{i,j}$ still contains its intersection with all the other lines and all segments have there endpoints on a circumscribing circle $C$. Every pair of segments stays intersecting, hence, we have a segment intersection representation of $H$.

Add a diameter $s_{i,i}$ to every disk $R_i$, this segment serves as representation for the leaf-path $P_{i,i}$. Altogether we have constructed a representation of $G$ obeying the required properties (a) and (b), see Figure 2 for an illustration.

### 3.3 Theorem 3 Is True for Trees with More Than One Inner Node

Now let $T$ be a tree with inner nodes $N = \{v_1, .., v_n\}$ and assume the Theorem has been proven for trees with at most $n-1$ inner nodes. Let $L = \{l_1, .., l_m\}$ be the set of leaves of $T$. With $L_i \subset L$ we denote the set of leaves attached to $v_i$. We have to produce a PSI-representation of the intersection graph $G$ of $\mathcal{P} = \{P_{i,j} \mid l_i, l_j \in L\}$, i.e., of the set of all leaf-paths of $T$. Let $v_1$ be the root of $T$, $h$ its resulting height and choose a vertex of $N$ with distance $h-1$ from the root, say $v_n$. We define two induced subtrees of $T$:

- The tree $T_n$ is the star with inner node $v_n$ and its leaves $L_n = \{l_k, .., l_m\}$.
- The tree $T'$ contains all nodes of $T$ except the leaves in $L_n$. The set of inner nodes of $T'$ is $N' = N \backslash \{v_n\}$, the set of leaves is $L' = L \backslash L_n \cup \{v_n\}$. For consistency we rename $l_0 := v_n$ in $T'$, hence $L' = \{l_0, l_1, .., l_{k-1}\}$.

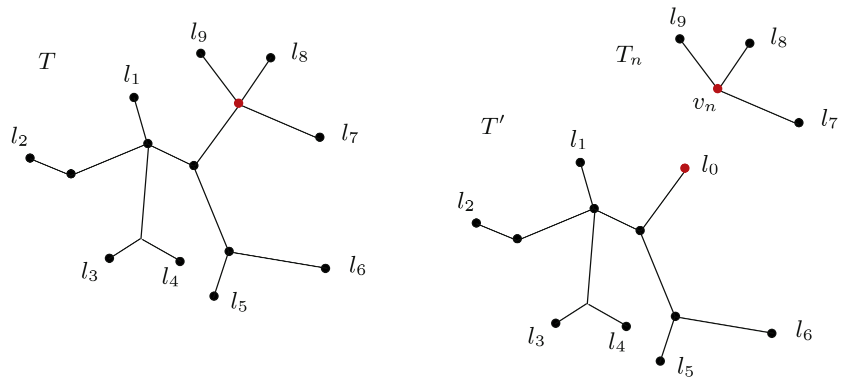


**Fig. 3.** A tree $T$ and the two induced subtrees $T'$ and $T_n$

Let $G_n$ and $G'$ be the VPT-graphs induced by all leaf-paths in $T_n$ and $T'$. Both these trees have fewer inner nodes than $T$. Therefore, by induction we can assume that we have PSI-representations $PS_n$ of $G_n$ and $PS'$ of $G'$ as claimed in Theorem 3. We will construct a PSI-representation of $G$ using $PS_n$ and $PS'$. The idea is as follows:

1. Replace every pseudosegment of $PS'$ representing a leaf-path ending in $l_0$ by a bundle of pseudosegments. This bundle stays within a narrow tube around the original pseudosegment.
2. Remove all pieces of pseudosegments from the interior of the disk $R_0$ and patch an appropriately transformed copy of $PS_n$ into $R_0$.
3. The crucial step is to connect the pseudosegments from the bundles through the interior of $R_0$ such that the induction invariants for the transformed disks of $R_r$ with $k \leq r \leq m$ are satisfied.

The set $\mathcal{P}$ of leaf-paths of $T$ can be partitioned into three parts. The subsets $\mathcal{P}'$ and $\mathcal{P}_n$ are leaf-paths of $T'$ or $T_n$ let the remaining subset be $\mathcal{P}^*$. The paths in $\mathcal{P}^*$ connect leaves $l_i$ and $l_r$ with $1 \leq i < k \leq r \leq m$, in other words they connect a leaf $l_i$ from $T'$ through $v_n$ with a leaf in $T_n$. We subdivide these paths into classes $\mathcal{P}^*_1, .., \mathcal{P}^*_{k-1}$ such that $\mathcal{P}^*_i$ consists of those paths from $\mathcal{P}^*$ which start in $l_i$. Each $\mathcal{P}^*_i$ consists of $|L_n|$ paths. In $T'$ we have the pseudosegment $s_{i,0}$ which leads from $l_i$ to $l_0$. Replace each such pseudosegment $s_{i,0}$ by a bundle of $|L_n|$ parallel pseudosegments routed in a narrow tube around $s_{i,0}$.

We come to the second step of the construction. Remove all pieces of pseudosegments from the interior of $R_0$. Recall that the representation $PS_n$ of $G_n$ from 3.2 has the property that all long pseudosegments have their endpoints on a circle $C$. Choose two arcs $A_b$ and $A_t$ on $C$ such that every segment spanned by a point in $A_b$ and a point in $A_t$ intersects each pseudosegment $s_{i,j}$ with $i \neq j$, this is possible by the choice of $C$. This partitions the circle into four arcs which will be called $A_b, A_l, A_t, A_r$ in clockwise order. The choice of $A_b$ and $A_t$ implies that each pseudosegment touching $C$ has one endpoint in $A_l$ and the other in $A_r$.

Map the interior of $C$ with an homeomorphism $h$ into a wide rectangular box $\Gamma$ such that $A_t$ and $A_b$ are mapped to the top and bottom sides of the box, $A_l$ is the left side and $A_r$ the right side. This makes the images of all long pseudosegments traverse the box from left to right. We may also require that the homomorphism maps the disks $R_r$ to disks and arranges them in a nice left to right order in the box, Figure 4 shows an example. The figure was generated by sweeping the representation from Figure 2 and converting the sweep into a wiring diagram (the diametrical segments $s_{r,r}$ have been re-attached horizontally).

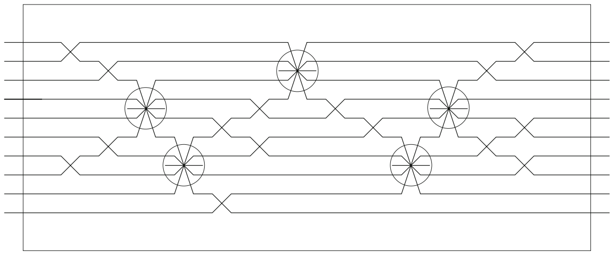


**Fig. 4.** A box containing a deformed copy of the representation from Figure 2

In the box we have a left to right order of the disks $R_r$, $l_r \in L_n$. By possibly relabeling the leaves of $L_n$ we can assume that the disks are ordered from left to right as $R_k, .., R_m$.

This step of the construction is completed by placing the box $\Gamma$ appropriately resized in the disk $R_0$ such that each of the segments $s_{i,0}$ from the representation of $G'$ traverses the box from bottom to top and the sides $A_l$ and $A_r$ are mapped to the boundary of $R_0$. The boundary of $R_0$ is thus partitioned into four arcs which are called $A_l, A'_t, A_r, A'_b$ in clockwise order. We assume that the segments $s_{i,0}$ touch the arc $A'_b$ in $PS'$ in counterclockwise order as $s_{1,0}, .., s_{k-1,0}$, this can be achieved by renaming the leaves appropriately.

Note that by removing everything from the interior of $R_0$ we have disconnected all the pseudosegments which have been inserted in bundles replacing the original pseudosegments $s_{i,0}$. Let $B_i^{in}$ be the half of the bundle of $s_{i,0}$ which touches $A'_b$ and let $B_i^{out}$ be the half which touches $A'_t$. By the above assumption the bundles $B_1^{in}, .., B_{k-1}^{in}$ touch $A'_b$ in counterclockwise order, consequently, $B_1^{out}, .., B_{k-1}^{out}$ touch $A'_t$ in counterclockwise order. Within a bundle $B_i^{in}$ we label the segments as $s_{i,k}^{in}, .., s_{i,m}^{in}$, again counterclockwise. The segment in $B_i^{out}$ which was connected to $s_{i,r}^{in}$ is labeled $s_{i,r}^{out}$ The pieces $s_{i,r}^{in}$ and $s_{i,r}^{out}$ will be part of the pseudosegment representing the path $P_{i,r}$.

To have property (b) for the pseudosegments of a bundle we twist whichever of the bundles $B_i^{in}$ or $B_i^{out}$ traverses $R_i$ within this disk $R_i$ thus creating a multiple intersection point. Note that they all cross $s_{i,i}$ as did $s_{i,0}$.

Also due to (b) the pseudosegments of paths $P_{i,r}$ for fixed $r \in \{k, .., m\}$ have to intersect in the disks $R_r$ inside of the box $\Gamma$. To prepare for this we take a narrow bundle of $k-1$ parallel vertical segments reaching from top to bottom of the box $\Gamma$ and intersecting the disk $R_r$. This bundle is twisted in the interior of $R_r$. Let $\check{a}_1^r, .., \check{a}_{k-1}^r$ be the bottom endpoints of this bundle from left to right and let $\hat{a}_1^r, .., \hat{a}_{k-1}^r$ be the top endpoints from right to left, due to the twist the endpoints $\check{a}_j^r$ and $\hat{a}_j^r$ belong to the same pseudosegment.

We are ready now to construct the pseudosegment $s_{i,r}$ that will represent the path $P_{i,r}$ in $T$ for $1 \leq i < k \leq r \leq m$. The first part of $s_{i,r}$ is $s_{i,r}^{in}$, this pseudosegment is part of the bundle $B_i^{in}$ and has an endpoint on $A'_b$. Connect this endpoint with a straight segment to $\check{a}_i^r$, from this point there is the connection up to $\hat{a}_i^r$. This point is again connected by a straight segment to the endpoint of $s_{i,r}^{out}$ on the arc $A'_t$. The last part of $s_{i,r}$ is the pseudosegment $s_{i,r}^{out}$ in the bundle $B_i^{out}$. The construction is illustrated in Figure 5.

It remains to prove that the construction indeed yields a representation of $G$ as intersection graph of pseudosegments and that this representation has the properties (a) and (b) from Theorem 3. The argument is split into a series of claims.

**Claim 1.** There is exactly one pseudosegment $s_{i,j}$ for every pair $l_i, l_j$ of leaves of $T$.

**Claim 2.** The pseudosegment $s_{i,j}$ traverses $R_i$ and $R_j$ but stays disjoint from every other of the disks.

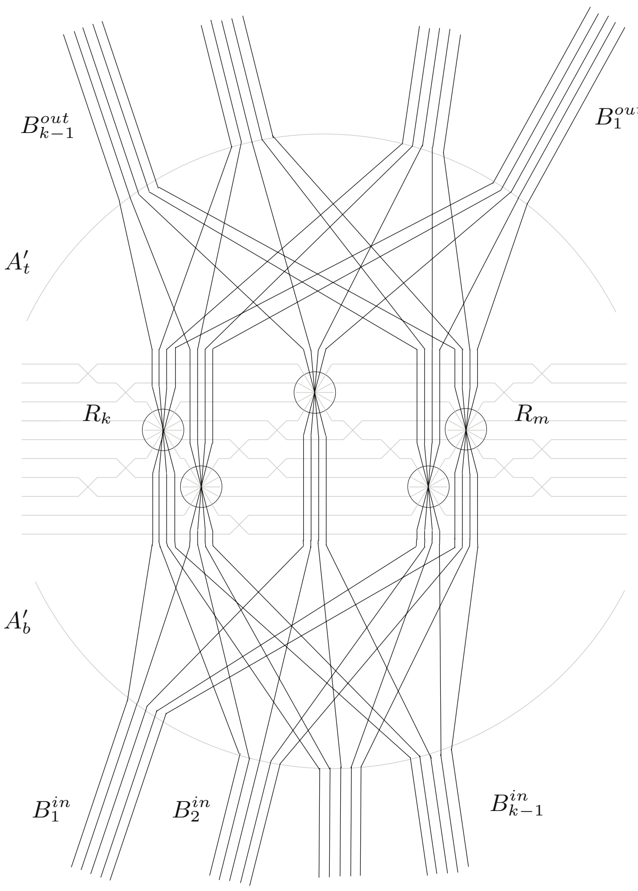


**Fig. 5.** The routing of pseudosegments in the disk $R_0$, an example

**Claim 3.** Any two pseudosegments intersecting the disk $R_i$ cross in $R_i$.

**Claim 4.** Two pseudosegments $s_{i,j}$ and $s_{i',j'}$ intersect at most once, i.e., to call them pseudosegments is justified.

**Claim 5.** Two pseudosegments $s_{i,j}$ and $s_{i',j'}$ intersect exactly if the corresponding paths $P_{i,j}$ and $P_{i',j'}$ intersect in $T$.

With the verification of these claims the proof of Theorem 3 is complete. In this extended abstract we dispense with the proof.

## 4 Proof of Theorem 2

Recall the definition of the graphs $K_n^3$ from the introduction: $K_n^3$ has two groups of vertices, the set $V_C = \{v_1, .., v_n\}$ induces a clique and there is an additional vertex $v_{i,j,k}$ for every triple $\{i, j, k\} \subset [n]$. These triple-vertices are adjacent only to the three corresponding vertices $v_i$, $v_j$ and $v_k$, hence form an independent set denoted $V_I$.

Assuming that there is a representation of $K_n^3$ as intersection graph of pseudosegments the set of pseudosegments can be divided into $PS_C$ and $PS_I$, i.e., the pseudosegments representing vertices from $V_C$ and $V_I$.

The pseudosegments of $PS_C$ form a set of pairwise crossing pseudosegments, we refer to the configuration of these pseudosegments as the arrangement $A_n$. The set $S = PS_I$ of 'small' pseudosegments has the following properties:

(i) Pseudosegments $t \neq t'$ from $S$ are disjoint,

(ii) Every pseudosegment $t \in S$ has nonempty intersection with exactly three pseudosegments from the arrangement $A_n$ and no two pseudosegments $t \neq t'$ intersect the same three pseudosegments from $A_n$.

The idea for the proof is to show that a set of pseudosegments with properties (i) and (ii) only has $O(n^2)$ elements. The theorem follows, since $|S| = \binom{n}{3} = \Omega(n^3)$.

## 4.1 Geometric Restriction

Every pseudosegment $p \in A_n$ is cut into $n$ pieces by the $n-1$ other pseudosegments of $A_n$. Let $W$ be the set of all the pieces obtained from pseudosegments from $A_n$, note that $|W| = n^2$. A triple-segment $t \in S$ intersects with exactly three pieces of three different pseudosegments of $A_n$. Hence it has a unique middle and two outer intersections. Let $S(w)$ be the set of triple-segments with middle intersection on the piece $w \in W$. The set $S$ is partitioned as $S = \dot{\bigcup}_{w \in W} S(w)$.

Define $G_p = (W, E_p)$ as the simple graph where two pieces $w, w'$ are adjacent if and only if there exists a triple-segment $t \in S$ such that $t$ has its middle intersection on $w$ and an outer intersection on $w'$.

**Lemma 2.** *$G_p = (W, E_p)$ is planar.*

*Proof.* A planar embedding of $G_p$ is induced by $A_n$ and $S$. Contract all pieces from pseudosegments in $A_n$, the contracted pieces represent the vertices of $G_p$.

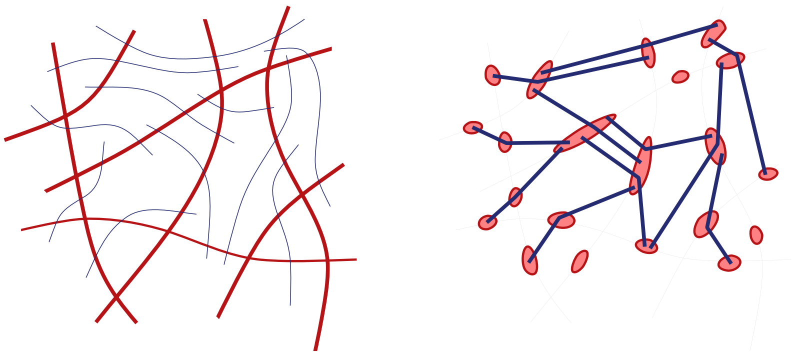


**Fig. 6.** A part of $A_n$ with some triple-segments and the edges induced by them of $G_p$

The pseudosegments in $S$ are pairwise non-crossing, this property is maintained during contraction of pieces, see Figure 6. If $t \in S$ has middle piece $w$ and outer pieces $w'$ and $w''$, then $t$ contributes the two edges $(w, w')$ and $(w, w'')$. Hence, the multigraph obtained through these contractions is planar and its underlying simple graph is indeed $G_p$. $\square$

Let $N(w)$ be the set of neighbors of $w \in W$ in $G_p$ and let $d_{G_p}(w) = |N(w)|$.

**Lemma 3.** *The size of a set $S(w)$ of triple-segments with middle piece $w$ is bounded by $d_{G_p}(w) - 1$ for every $w \in W$.*

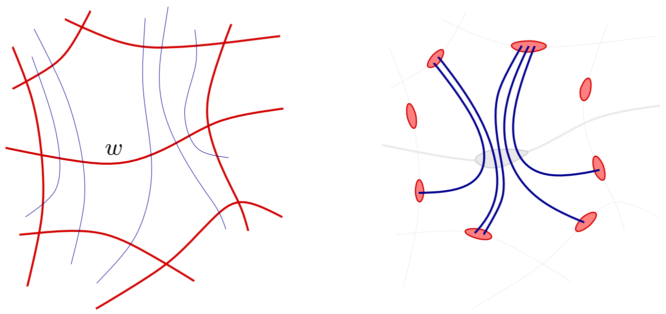


**Fig. 7.** The planar graph $NG_w$ induced by the triple-segments with middle intersection

*Proof.* Define the neighborhood graph $NG_w = (N(w), E_w)$ for every $w \in W$ where two vertices $u, u'$ are adjacent, if and only if there is a short pseudosegment $t \in S(w)$ with $u$ and $u'$ as outer pieces. The number of edges of $NG_w$ equals the number of triple-segments in $S(w)$. The idea is to contract just the pieces corresponding to elements of $N(w)$ to points. The triple-segments in $S(w)$ together with the vertices obtained by contraction form a planar graph, see Figure 7. Note that the resulting graph is not a multigraph, since a multiple edge would correspond to a pair of triple-segments intersecting the same three pieces of $A_n$. We will show that $NG_w$ is acyclic, hence a forest. This implies the statement of Lemma 3.

Assume there was a cycle $C$ in $NG_w$. Label its vertices $w_1, .., w_k \in N(w)$ such that $w_k w_1$ and $w_i w_{i+1}$ with $1 \leq i < k$ are the edges of $C$. Let $t_i$ be the triple-segment defining edge $w_i w_{i+1}$. Recall that $t_i$ intersects $w_i$, $w$ and $w_{i+1}$. A cycle in $NG_w$ corresponds to a simple closed curve in the PSI-representation as follows: Denote the part of $w_i$ connecting its crossings with $t_i$ and $t_{i-1}$ by $a_i$. As $w_i \in W$ is a piece and $t_j \in S(w) \subseteq S$, the set of pieces and triple-segments contributing to $C$ as vertices or edges does not induce more crossings than the pairs $(w_i, t_j)$ with $j \in \{i-1, i\}$. Ignoring the pending ends of $t_i$, the union of the $a_i$ and $t_i$, $i \in \{1, .., k\}$ corresponds to a simple closed curve $\Gamma$ within the PSI-representation. The curve $\Gamma$ is the concatenation of $a_i$ and $t_i$ in the order $a_1, t_1, a_2, .., t_{k-1}, a_k, t_k$, see Figure 8.

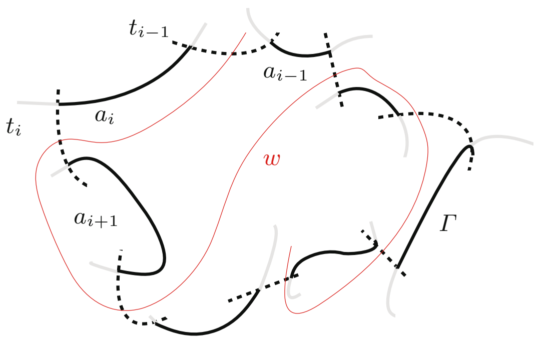


**Fig. 8.** A chain of segments corresponding to a cycle in $NG_w$

Note that $k \geq 3$, as $NG_w$ is not a multigraph. Choose three triple-segments $t_j, t_i, t_l$ from $\Gamma$ such that they intersect $w$ in this order and no other triple-segment of $C$ crosses $w$ between them. We will identify a simple closed curve $\gamma$ separating $w_i$ and $w_{i+1}$. Even more, the complete pseudosegments $p$ and $p'$ containing $w_i$ respectively $w_{i+1}$ will be separated by $\gamma$. This is a contradiction to the fact that they both belong to the set $A_n$ of pairwise intersecting pseudosegments.

Such a curve $\gamma$ can be described as follows: By possibly shifting the indices of $w_i$ along $C$, we can assume $i < j < l$. Denote the part of $w$ between its intersections with $t_j$ and $t_l$ by $w_{j,l}$. Be $P_{j+1,l}$ the subpath of $C \backslash w_i$ connecting $w_{j+1}$ and $w_l$, denote the corresponding part of $\Gamma$ by $\Gamma_{j+1,l}$. Connect $w_{j,l}$ to $\Gamma_{j+1,l}$ at $t_j$ and $t_l$. This gives a simple closed curve $\gamma$. The curve $\gamma$ consists of an arc of $\Gamma$, the arc $w_{j,l}$ of $w$ and parts of $t_j$ and $t_l$. It follows that $\gamma$ can not be crossed by a pseudosegment from $A_n$. Recall that by definition of PSI-representations an intersection of pseudosegments implies, that they cross. Applied to the intersection of $t_i$ and $w_{j,l} \subset w$ this has the consequence that $w_i$ and $w_{i+1}$ are on

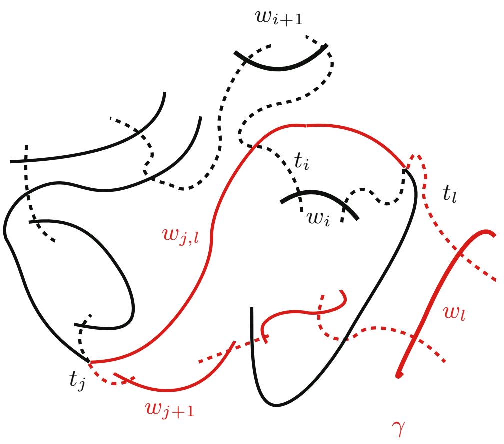


**Fig. 9.** Pieces $w_i$ and $w_{i+1}$ are separated by $\gamma$

different sides of $\gamma$, see Figure 9. As shown before this implies that the complete pseudosegments $p$ containing $w_i$ and $p'$ containing $w_{i+1}$ have to lie on different sides of $\gamma$, hence they can not intersect, a contradiction to the choice of $A_n$. Thus $NG_w$ is acyclic and $E_w$ is bounded by $d_{G_p}(w) - 1$. $\square$

In addition to the result of Lemma 3 we know

- $|W| = n^2$,
- $\sum_{w \in W} d_{G_p}(w) = 2|E_p| < 6|W|$.

This implies $|S| = \sum_{w \in W} |S(w)| \leq \sum_{w \in W} (d_{G_p}(w) - 1) < 6|W| = 6n^2$.

Since $\binom{n}{3} > 6n^2$ for all $n \geq 39$ we conclude that $K_n^3$ does not belong to PSI for $n \geq 39$. This completes the proof of Theorem 2.

## 5 Conclusions and Further Questions

We have investigated the containment relation between the classes of PSI-graphs and of chordal graphs. This may stimulate investigations concerning the relation between PSI-graphs and other classes. Of course the main open problem in the area remains the question whether all planar graphs are PSI-graphs.

## References

1. M.Bodirsky, C.Dangelmayr and J. Kára: Representing Series-parallel Graphs as Intersection Graphs of Line Segments in Three Directions, submitted.
2. N. de Castro, F. J. Cobos, J. C. Dana and A. Márquez: Triangle-Free Planar Graphs as Segment Intersection Graphs, *Journal of Graph Algorithms and Applications*, Volume 6, pp. 7–26, 2002.
3. I. Ben-Arroyo Hartman, I. Newman and R. Ziv: On grid intersection graphs, *Discrete Math.*, 87, pp. 41–52, 1991.
4. H. de Fraysseix, P.O. de Mendez and J. Pach: Representation of planar graphs by segments. *Colloquia Mathematica Societatis János Bolyai, Intuitive Geometry*, Szeged (Hungary), 1991.
5. H.de Fraysseix and P.O. de Mendez: Contact and Intersection Representations, GD 2004, *Lecture Notes in Computer Science*, Volume 3383, pp. 217–227, 2005.
6. F. Gavril, A recognition algorithm for the intersection graphs of paths in trees *Discrete Math.* Volume 23, pp. 211–227, 1978.
7. J. Kratochvil: String Graphs II: Recognizing String Graphs is NP-hard, *Journal of Comb. Theory*,Ser. B 52,No. 1, pp. 67–78 , 1991.
8. J. Kratochvil: A special planar satisfiability problem and a consequence of its NP-completeness, *Discrete Applied Mathematics*, 52, pp. 233–252 , 1994.
9. J. Kratochvil and J. Matoušek: Intersection graphs of Segments, *Journal of Comb. Theory*, Ser. B 62, pp. 289–315 , 1994.
10. C. Monma and V. K. Wei, Intersection graphs of paths in a tree, *Journal of Comb. Theory*, Ser. B 41, pp. 141–181 , 1986.
11. M. Schaefer, E. Sedgwick and D. Stefanovic: Recognizing string graphs in NP, *Journal of Comput. Syst.Sci.*, No. 2, pp. 365–380, 2003.
12. E. R. Scheinerman: Intersection classes and multiple intersection parameters of graphs, *PhD thesis*, Princeton University 1984.

# Parameterized *st*-Orientations of Graphs: Algorithms and Experiments

Charalampos Papamanthou[1] and Ioannis G. Tollis[2]

[1] Department of Computer Science, Brown University, P.O. Box 1910, Providence RI, U.S.A.
cpap@cs.brown.edu

[2] Department of Computer Science, University of Crete, P.O. Box 2208 & Institute of Computer Science, FORTH, P.O. Box 1385
Heraklion, Greece
tollis@ics.forth.gr

**Abstract.** *st*-orientations (*st*-numberings) or bipolar orientations of undirected graphs are central to many graph algorithms and applications. Several algorithms have been proposed in the past to compute an *st*-orientation of a biconnected graph. However, as indicated in [1], the computation of more than one *st*-orientation is very important for many applications in multiple research areas, such as this of Graph Drawing. In this paper we show how to compute such orientations with certain (parameterized) characteristics in the final *st*-oriented graph, such as the length of the longest path. Apart from Graph Drawing, this work applies in other areas such as Network Routing and in tackling difficult problems such as Graph Coloring and Longest Path. We present primary approaches to the problem of computing longest path parameterized *st*-orientations of graphs, an analytical presentation (together with proof of correctness) of a new $O(m \log^5 n)$ ($O(m \log n)$ for planar graphs) time algorithm that computes such orientations (and which was used in [1]) and extensive computational results that reveal the robustness of the algorithm.

## 1 Introduction

The problem of orienting an undirected graph such that it has one source, one sink, and no cycles (*st*-orientation) is central to many graph algorithms and applications, such as graph drawing [2,3,4,5,6], network routing [7,8] and graph partitioning [9].

*st*-numberings were first introduced in 1967 in [10], where it is proved that given any edge $\{s,t\}$ of a biconnected undirected graph $G$, we can define an *st*-numbering. The proof of a theorem in [10] gives a recursive algorithm that runs in time $O(nm)$. However, in 1976 Even and Tarjan proposed an algorithm that computes an *st*-numbering of an undirected biconnected graph in $O(n+m)$ time [11]. Ebert [12] presented a slightly simpler algorithm for the computation of such a numbering, which was further simplified by Tarjan [13]. The planar case has been extensively investigated in [14] where a linear time algorithm is presented which may reach any *st*-orientation of a planar graph. Additionally, in [15] a parallel algorithm is described (running in $O(\log n)$ time using $O(m)$ processors) and finally in [16] another linear time algorithm for the problem is presented. An overview of the work concerning bipolar orientations is presented in [17].

However, as indicated in [1], the computation of more than one *st*-orientation is very important for many applications in the area of Graph Drawing. Most algorithms use any algorithm that produces such an orientation, e.g., [11], without expecting any specific properties of the oriented graph.

In this paper we approach the problem of computing *st*-orientations of specific properties. We present and give proof of correctness of algorithms that are able to control the length of the longest path of the resulting directed acyclic graph. The algorithms run in $O(m \log^5 n)$ time ($O(m \log n)$ time for planar graphs). This provides significant flexibility to many graph algorithms and applications [2,3,7,8,9]. We also present extensive experimental results that reveal the robustness of the algorithm.

## 2 Preliminaries

### 2.1 The Very First Approach

The aim of this work has always been the computation of *st*-orientations of longest path length that can be efficiently controlled by an input. Towards this goal, we investigated the possibility of modifying the existing linear algorithms in order to produce longest path parameterized *st*-orientations. These algorithms, such as [11], proceed by choosing over a set a vertices. Thus in order to produce multiple *st*-orientations using these algorithms, one should try to consider different combinations of successive vertices. Some heuristics applied for the Tarjan-Even algorithm are described in [18], where after extensive computational results, we reached to the conclusion that exploiting the *freedom* of choice of successive vertices this algorithm gives us, can only lead to *st*-orientations that almost have **no** difference in the longest path length. After similar attempts on other existing algorithms, it became evident that linear time was not enough to produce both a correct *st*-orientation and to be able to discriminate between different longest path length *st*-orientations. As a result of this, we seeked for new algorithms that achieve this goal.

### 2.2 Exploiting Biconnectivity

The main idea behind the algorithm was the explosion of the biconnectivity structure of a graph. This finally seemed to be of great importance in a way that we could compute both a correct *st*-orientation of a graph and we could efficiently influence the length of the longest path of the computed *st*-orientation by using some information this structure can provide.

Following, we introduce some terminology and preliminary results. Throughout the paper, $N_G(v)$ denotes the set of neighbors of node $v$ in graph $G$, $s$ the source of the graph, $t$ the sink of the graph and $l(u)$ the length of the longest path of a node $u$ from the source $s$ of the graph. Let $G = (V, E)$ be a one-connected undirected graph, i.e., a graph that contains at least one vertex whose removal causes the initial graph to disconnect. The vertices that have that property are called *separation vertices*, *articulation points* or *cutpoints*. Each one-connected graph is composed of a set of blocks (biconnected components) and cutpoints that form a tree structure. This tree is called the block-cutpoint tree [19] of the graph and its nodes are the blocks and cutpoints of the graph. Suppose now that $G$ consists of a set of blocks $B$ and a set of cutpoints $C$. The respective block-cutpoint tree $T = (B \cup C, U)$ has $|B| + |C|$ nodes and $|B| + |C| - 1$ edges. The edges $(i, j) \in U$ of the block-cutpoint tree always connect pairs of blocks and cutpoints such that the cutpoint of a tree edge belongs to the vertex set of the corresponding block.

The block-cutpoint tree is a free tree, i.e., it has no distinguished root. In order to transform this free tree into a rooted tree, we define the $t$-rooted block-cutpoint tree with respect to a vertex $t$. Consequently, the root of the block-cutpoint tree is the block that contains $t$.

Finally, we define the leaf-blocks of the $t$-rooted block-cutpoint tree to be the blocks, except for the root, of the block-cutpoint tree that contain a single cutpoint. The block-cutpoint tree can be computed in $O(n + m)$ time with an algorithm similar to DFS [19]. Following, we give the description of the algorithm.

## 3 The Algorithm

The main idea of the algorithm is based on the successive removal of nodes and the simultaneous update of the $t$-rooted block-cutpoint tree. We call each removed node a *source*, because at the time of its removal it is effectively chosen to be a source of the remainder of the graph. We initially remove $s$, the first source, which is the source of the desired $st$-orientation and orient all its incident edges from $s$ to all its neighbors. After this removal, there exist three possibilities:

- The graph remains biconnected
- The graph is decomposed into several biconnected components but the number of leaf-blocks remains the same
- The graph is decomposed into several biconnected components and the number of leaf-blocks changes

This procedure continues until all nodes of the graph but one are removed. Finally, we encounter the desired sink, $t$, of the final $st$-orientation. The updated biconnectivity structure gives us information about the choice of our next source. Actually, the biconnectivity maintenance allows us to remove nodes and simultaneously maintain a "map" of possible vertices whose future removal may or may not cause dramatic changes to the structure of the tree.

As it will be clarified in the next sections, at every step of the algorithm there will be a set of potential sources to continue the execution. Our aim is to establish a connection between the current source choice and the length of the longest path of the resulting $st$-oriented graph.

**Lemma 1.** *Let $G = (V, E)$ be an undirected biconnected graph and $s$, $t$ be two of its nodes. Suppose we remove $s$ and all its incident edges. Then there is at least one neighbor of $s$ lying in each leaf-block of the $t$-rooted block-cutpoint tree of $G - \{s\}$. Moreover, this neighbor is not a cutpoint.*

*Proof.* If graph $G - \{s\}$ is still biconnencted, the proof is trivial, as the $t$-rooted block-cutpoint tree consists of a single node (the biconnected component $G - \{s\}$), which is both root and leaf-block of the $t$-rooted block-cutpoint tree. If graph $G - \{s\}$ is one-connected, suppose that there is a leaf-block $\ell$ of the $t$-rooted block-cutpoint tree defined by cutpoint $c$ such that $N(s) \cap \ell = \{\emptyset\}$. Then $c$, if removed, still disconnects $G$ and thus $G$ is not biconnected, which does not hold. The same occurs if $N(s) \cap \ell = \{c\}$. Hence there is always at least one neighbor of $s$ lying in each leaf-block of the $t$-rooted block-cutpoint tree, which is not a cutpoint. □

Let now $G = (V, E)$ be an undirected biconnected graph and $s$, $t$ two of its nodes. We will compute an $st$-orientation of $G$. Suppose we recursively produce the graphs $G_{i+1} = G_i - \{v_i\}$, where $v_1 = s$ and $G_1 = G$ for all $i = 1, \ldots, n - 1$ (note that the subscript $i$ of $v_i$ denotes the order with which the nodes are removed).

During the procedure (which we call STN) we always maintain a $t$-rooted block-cutpoint tree. Additionally, we maintain a structure $Q$ that plays a major role in the choice of the next source. $Q$ initially contains the desired source for the final orientation, $s$. Finally we maintain the leaf-blocks of the $t$-rooted block-cutpoint tree. During every iteration $i$ of the algorithm node $v_i$ is chosen so that

– it is a non-cutpoint node that is an element of $Q$
– it belongs to a leaf-block of the $t$-rooted block-cutpoint tree

Note that for $i = 1$ there is a single leaf-block (the initial biconnected graph) and the cutpoint that defines it is the desired sink of the orientation, $t$. When a source $v_i$ is removed from the graph, we have to update $Q$ in order to be able to choose our next source. $Q$ is then updated by removing $v_i$ and by inserting all nodes $u \in N_{G_i}(v_i)$ except for $t$.

Each time a node $v_i$ is removed we orient all its incident edges from $v_i$ to its neighbors. The procedure continues until $Q$ gets empty. Let $F = (V', E')$ be the directed graph computed by this procedure. We claim that $F = (V', E')$ is an $st$-oriented graph:

**Lemma 2.** *During STN, every node becomes a source exactly once. Additionally, after exactly $n - 1$ iterations (i.e., after all nodes but $t$ have been processed), $Q$ becomes empty.*

*Proof.* Let $v \neq t$ be a node that never becomes a source. This means that all incident edges $(u, v)$ have direction $u \to v$. As the algorithm gradually removes sources, by simultaneously assigning direction, one $u$ must be a cutpoint (as $v \neq t$ will become a biconnected component of a single node). But all nodes $u$ are chosen to be neighbors of prior sources. By Lemma 1, $u$ can never be a cutpoint, hence node $v \neq t$ will certainly become a source exactly once. Finally, $Q$ gets empty at the end of the algorithm as each time at least one node is added into $Q$ and exactly one node is removed from it. □

By Lemmas 1, 2, we see that at each iteration of the algorithm there will be at least one node to be chosen as a future source.

**Corollary 3.** *Suppose after a vertex $v$ is removed, $r$ different leaf-blocks are created. Then in each leaf-block of the $t$-rooted block-cutpoint tree there exists at least one non-cutpoint node that belongs to $Q$.* □

**Lemma 4.** *The directed graph $F = (V', E')$ has exactly one source $s$ and exactly one sink $t$.*

*Proof.* Node $v_1 = s$ is indeed a source, as all edges $(v_1, N(v_1))$ are assigned a direction from $v_1$ to its neighbors in the first step. Node $t$ is indeed a sink as it is never chosen to become a current source and all its incident edges are assigned a direction from its neighbors to it during prior iterations of STN. We have to prove that all other nodes have at least one incoming and one outgoing edge. As all nodes $v \neq t$ become sources exactly once, there will be at least one node $u$ such that $(v, u) \in E'$. Sources $v \neq t$ are actually nodes that have been inserted into $Q$ during a prior iteration of the algorithm. Before being chosen to become sources, all nodes $v \neq s \neq t$ are inserted into $Q$ as neighbors of prior sources and thus there is at least one $u$ such that $(u, v) \in E'$. Hence $F$ has exactly one source and one sink. □

**Lemma 5.** *The directed graph $F = (V', E')$ has no cycles.*

*Proof.* Suppose STN has ended and there is a directed cycle $v_j, v_{j+1}, \ldots, v_{j+l}, v_j$ in $F$. This means that $(v_j, v_{j+1}), (v_{j+1}, v_{j+2}), \ldots, (v_{j+l}, v_j) \in E'$. During STN, after an edge $(v_k, v_{k+1})$ is inserted into $E'$, $v_k$ is deleted from the graph and never processed again and $v_{k+1}$ is inserted into $Q$ so that it becomes a future source. In our case after edges $(v_j, v_{j+1}), (v_{j+1}, v_{j+2}), \ldots, (v_{j+l-1}, v_{j+l})$ will have been oriented, nodes $v_j, v_{j+1}, \ldots, v_{j+l-1}$ will have been deleted from the graph. To create a cycle, $v_j$ should be inserted into $Q$ as a neighbor of $v_{j+l}$, which does not hold as $v_j \notin N_{G_{j+l}}(v_{j+l})$ ($v_j$ has already been deleted from the graph). Thus $F$ has no cycles. □

By Lemmas 4, 5 we have:

**Theorem 6.** *The directed graph $F = (V', E')$ is st-oriented.* □

Alg.1 is a recursive algorithm for the *st*-orientation computation of a biconnected undirected graph $G$. During the execution of the algorithm we can also compute an *st*-numbering $f$ (line 9) of the initial graph. Actually, for each node $v_i$ that is

**Algorithm 1.** STN$(G, s, t)$

```
1: Initialize $F = (V', E')$;
2: Initialize $m(i) = 0$ for all nodes $i$ of the graph; (timestamp vector)
3: $j = 0$; $\{$Initialize a counter$\}$
4: $Q = \{s\}$; $\{$Insert $s$ into $Q\}$
5: STREC$(G, s)$; $\{$Call the recursive algorithm$\}$
6: ------------------------------------------------------------
7: function STREC$(G, v)$
8: $j = j + 1$;
9: $f(v) = j$;
10: $V = V - \{v\}$;   $\{$A source is removed from $G\}$
11: $V' = V' \cup \{v\}$;  $\{$and is added to $F\}$
12: for all edges $(v, i) \in E$ do
13:     $E = E - \{(v, i)\}$
14:     $E' = E' \cup \{(v, i)\}$
15: end for
16: $Q = Q \cup \{N(v) \sim t\} - \{v\}$; $\{$The set of possible next sources$\}$
17: $m(N(v)) = j$;
18: if $Q == \{\emptyset\}$ then
19:     $f(t) = n$;
20:     return;
21: else
22:     $T(t, B_j^1, B_j^2, \ldots, B_j^r)$=UpdateBlocks$(G)$; $\{$Update the block-cutpoint tree; $h_j^i$ is
        the cutpoint that defines the leaf-block $B_j^i\}$
23:     for all leaf-blocks $(B_j^i, h_j^i)$ do
24:         choose $v_\ell \in B_j^\ell \cap Q \sim \{h_j^\ell\}$
25:         STREC$(G, v_\ell)$;
26:     end for
27: end if
```

removed from the graph, the subscript $i$ is the final *st*-number of node $v_i$. The *st*-numbering can also be computed in linear time after the algorithm has ended, by executing a topological sorting on the computed *st*-oriented graph $F$.

Note that in the algorithm we use a vector $m(v)$ (line 17), where we store a timestamp for each node $v$ of the graph that is inserted into $Q$. These timestamps are of great importance during the choice of the next candidate source and will give us the opportunity to control the length of the longest path. Actually, they express the last time that a node $v$ became candidate for removal.

Note that the recursion is executed exactly $n - 1$ times. The running time of each recursive call is consumed by the procedure that updates the block-cutpoint tree, which is $O(n + m)$ [19]. Hence it is easy to conclude that STN runs in $O(nm)$ time. However, it can be made to run faster by a more efficient algorithm to maintain biconnectivity.

In fact, Holm, Lichtenberg and Thorup [20] investigated the problem of maintaining a biconnectivity structure without computing the block-cutpoint tree from scratch. They presented a fully dynamic algorithm that supports the insertion and deletion of edges and maintains biconnectivity in $O(\log^5 n)$ amortized time per edge insertion or deletion. In our case, only deletions of edges are done. If we use this algorithm in order to keep information about biconnectivity, we obtain the following:

**Theorem 7.** *Algorithm STN can be implemented to run in* $O(m \log^5 n)$ *time.* □

Finally, for planar graphs, we can compute biconnected components in $O(\log n)$ amortized time per edge deletion due to [21]. Hence, the algorithm can be implemented to run in $O(m \log n)$ time for planar graphs [1].

Finally, the *st*-orientation algorithm defines an *st-tree* $T_s$ (Figure 1). Its root is the source of our graph $s$ ($p(s) = -1$). It can be computed during the execution of the algorithm. When a node $v$ is removed, we simply set $p(u) = v$ for every neighbor $u$ of $v$, where $p(u)$ is a pointer to the father of each node $u$. The father of a vertex can be updated many times until the algorithm terminates. This tree is a directed tree that has two kinds of edges, the *tree edges*, which show the *last* father-ancestor assignment between two nodes made by the algorithm and the *non-tree* edges that include all the remaining edges. The non-tree edges never produce cycles. Finally, note that the sink $t$ is always a leaf of the *st*-tree $T_s$.

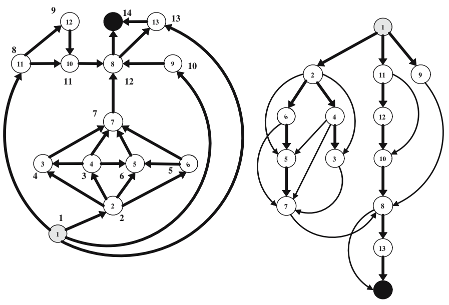


**Fig. 1.** Algorithm execution on a graph (left) and the respective *st*-tree (right). Beside each node of the graph the **STN** rank of visit is depicted.

As it happens with every *st*-oriented graph, there is a directed path from every node $v$ to $t$ and hence the maximum depth of the *st*-tree will be a lower bound for the length of the longest path, $l(t)$:

**Theorem 8.** *Let $G$ be an undirected biconnected graph and $s$, $t$ two of its nodes. Suppose we run STN on it and we produce the st-oriented graph $F$ and its st-tree $T_s$. If $d(T_s)$ denotes the maximum depth of the st-tree then $l(t) \geq d(T_s)$.* □

Additionally, there is a strong connection between the structure of the $t$-rooted block-cutpoint tree and the length of the longest path of the *st*-oriented graph that STN computes:

[1] We thank Philip Klein for this observation.

**Theorem 9.** *Suppose STN is run on an undirected st-Hamiltonian graph*[2] *$G$. Let $k_i$ denote the number of the leaf-blocks of the $t$-rooted block-cutpoint tree after the $i$-th removal of a node, for $i = 1, 2, \ldots, n-1$. Then $l(t) \leq n - 1 - \sum_{k_i > k_{i-1}} (k_i - k_{i-1})$.* □

## 4 Longest Path Parameterized Orientations

### 4.1 Maximal and Minimal Case

We have used two approaches in order to produce *st*-oriented graphs with long longest path and *st*-oriented graphs with small longest path. As presented in Section 3, during each iteration of the algorithm a timer $j$ (line 8 of Algorithm 1) is incremented and each vertex $x$ that is inserted into $Q$ gets a timestamp $m(x) = j$.

Our investigation has revealed that if vertices with high timestamp are chosen then long sequences of vertices are formed and thus there is higher probability to obtain a long longest path length. We call this way of choosing vertices MAX-STN. Actually, MAX-STN resembles a DFS traversal (it searches the graph at a *maximal* depth). Hence, during MAX-STN, the next source $v$ is arbitrarily chosen from the set $\{v \in Q' : m(v) = \max\{m(i) : i \in Q'\}\}$.

On the contrary, we have observed that if vertices with low timestamp are chosen, then the final *st*-oriented graph has relatively small longest path length. We call this way of choosing vertices MIN-STN, which in turn resembles a BFS traversal. Hence, during MIN-STN, the next source $v$ is arbitrarily chosen from the set $\{v \in Q' : m(v) = \min\{m(i) : i \in Q'\}\}$. Note that the choice of the new vertex with the minimum or maximum timestamp does not influence the running time of the algorithm (it can be done in $O(\log n)$ time) since we can implement $Q'$ as a priority queue.

### 4.2 Medium (Parameterized) Case

Instead of computing *st*-oriented graphs with either long or small length of longest path, it would be desirable to be able to produce medium (parameterized) longest path *st*-oriented graphs. So the question that arises is: Can we insert a parameter into our algorithm, for example a real constant $p \in [0, 1]$ so that our algorithm computes a directed acyclic graph of length of longest path that is a function of $p$?

It turns out that this is feasible if we modify STN. As the algorithm is executed exactly $n$ times ($n$ vertices are removed from the graph), we can execute MAX-STN for the first $pn$ iterations and MIN-STN for the remaining $(1-p)n$ iterations. We call this method PAR-STN($p$) and we say that it produces an *st*-oriented graph with length of longest path from $s$ to $t$ equal to $\Delta(p)$. Note that PAR-STN(0) is equivalent to MIN-STN and $\Delta(0) = \lambda(t)$, while PAR-STN(1)

[2] We say that a graph is *st*-Hamiltonian when it has at least one simple path from $s$ to $t$ that contains all the other nodes of the graph.

is equivalent to MAX-STN and $\Delta(1) = \ell(t)$. PAR-STN has been tested and it produces longest paths with $\Delta(p) \geq p(n-1)$, when applied to $st$-Hamiltonian graphs. Actually, $\Delta(p)$ is very close to $p(n-1)$. Additionally, it has been observed that if we switch the order of MAX-STN and MIN-STN execution, i.e., execute MIN-STN for the first $pn$ iterations and MAX-STN for the remaining $(1-p)n$ iterations, then we get longest paths with $\Delta(p) \leq p(n-1)$. These observations are fully clarified in the experimental results, where it is evident that the algorithm can compute many $st$-numberings within $[\lambda(t), \ell(t)]$. In this way, applications that use $st$-numberings can use this interval to compute an optimized solution.

### 4.3 Longest Path Timestamps and Weighted Graphs

If we apply a relaxation phase during STN, we can compute the longest path length $l(v)$ from $s$ to every node $v$ during the execution of STN. This can be achieved as follows: At the beginning, we initialize the longest path vector $l$ to be the zero vector, hence $l(v) = 0\ \forall v \in V$. Suppose that at a random iteration of the algorithm we remove a node $u$ and we orient all $u$'s incident edges $(u, i)$ away from $u$. For every oriented edge $(u, i) \in E'$ of weight $w_{ui}$ (in case of unweighted graphs it is $w_{ui} = 1$) we relax $l(i)$ as follows:

```
1: for all $(u,i) \in E'$ do
2:    if $l(i) < l(u) + w_{ui}$ then
3:       $l(i) = l(u) + w_{ui}$;
4:    end if
5: end for
```

Instead of now using the timestamps $m(u)$ to choose the next source of the algorithm, we can use the *online* computed longest paths $l(u)$.

This method mainly applies to the case of weighted graphs. By using the *relaxed* longest path length as a timestamp, we can produce long or short $st$-orientations of weighted graphs. Hence, the presented algorithm, implemented with the longest path timestamp method can be used to compute weighted numberings on the weighted $st$-oriented graph that is produced.

## 5 Some Applications

There are many areas where parameterized $st$-orientations can apply. For example, many Graph Drawing Algorithms [2,3,4,5,6] use an $st$-orientation as their first step. Actually, the length of the longest path of the used $st$-orientation determines the area bounds of the final drawing (the area can be reduced by a factor of $n$ for some classes of graphs [1]). More details can be found in [1].

Additionally, MAX-STN can be used as a heuristic for the longest path problem in undirected graphs. Actually, as we will see in the experimental results section, MAX-STN produces *near* optimal results for this problem.

MIN-STN can even be used to compute good colorings of graphs. By producing a good solution for the minimum $st$-orientation problem we can obtain a good solution for the graph coloring problem, based on the polynomial reduction between the two problems [22].:

**Theorem 10.** *( [22]) Let $G = (V, E)$ be a connected graph and let $G' = (V \cup \{s, t\}, \{E \cup \{s, i\} \cup \{t, i\} \forall i \in V\})$. $\chi$ is the chromatic number of $G$ if and only if $G'$ can be st-oriented with a minimum longest path length st-orientation with $l(t) = \chi + 1$.* □

Based on the above theorem (which actually justifies the $NP$-hardness of the minimum longest path length $st$-orientation problem[3]), we used PAR-STN(0) to $st$-orient $G'$ and derive a coloring for $G$. We give some indicative results for the graph coloring problem in the experimental results section.

## 6 Experimental Results

The first tests were conducted on $st$-Hamiltonian Graphs (Table 1). We used this class of graphs as they have an a priori known upper bound for the maximum longest path length equal to $n - 1$. In this way, we could see how effective the parameter $p$ is. In order to construct the graphs in random, we compute a random permutation $P$ of the vertices of the graph. Then we construct a cycle by adding the undirected edges $(P(1), P(2)), (P(2), P(3)), \ldots, (P(n-1), P(n)), (P(n), P(1))$ and we chose at random two adjacent nodes of the cycle to be the source $s$ and the sink $t$ of our graph. This guarantees the existence of a Hamiltonian path from $s$ to $t$ and a possible maximum longest path length of every $st$-oriented graph of length $n - 1$. Finally we add the remaining $nd - n$ edges, given that the density of the desired graph is $d$. We keep a list of edges that have not been inserted and make exactly $nd - n$ random choices on this list, by simultaneously inserting the chosen undirected edge into the graph and updating the list of the remaining undirected edges. During the execution of the algorithm, ties between the timestamps of the candidate sources are broken at random. We isolate the nodes that satisfy the current timestamp condition (i.e., the nodes with maximum timestamp in case of MAX-STN and the nodes with minimum timestamp in case of MIN-STN) and afterwards we choose a node from the isolated set at random.

The second series of experiments was conducted on low density (roughly equal to 1.5) planar graphs (Table 2). These graphs were constructed as follows: We build up a tree of $n$ nodes by randomly picking up a node and setting it to be the root of the tree. Then we connect the current tree (initially it only consists of the root) with a node that does not belong to the current tree and which is chosen at random. We execute the same procedure till all nodes are inserted into the tree. Then we connect the leaves of the tree following a preorder numbering so that all crossings are avoided.

Finally, the third series of experiments were conducted on weighted graphs (Figure 2). We used the algorithm described section 4.3 and make use of

[3] Note that the maximum longest path length $st$-orientation problem is $NP$-hard by reduction from the directed Hamilton Path problem.

**Table 1.** Results for density 6.5 *st*-Hamiltonian graphs

| n | p=0 | p=0.3 | p=0.5 | p=0.7 | p=1 |
|---|---|---|---|---|---|
| | $\frac{l(t)}{(n-1)}$ | $\frac{l(t)}{(n-1)}$ | $\frac{l(t)}{(n-1)}$ | $\frac{l(t)}{(n-1)}$ | $\frac{l(t)}{(n-1)}$ |
| 200 | 0.085 | 0.372 | 0.568 | 0.743 | 0.963 |
| 400 | 0.051 | 0.341 | 0.540 | 0.731 | 0.964 |
| 600 | 0.041 | 0.332 | 0.532 | 0.726 | 0.963 |
| 800 | 0.033 | 0.330 | 0.527 | 0.721 | 0.962 |
| 1000 | 0.027 | 0.325 | 0.521 | 0.716 | 0.967 |
| 1200 | 0.024 | 0.322 | 0.521 | 0.718 | 0.965 |
| 1400 | 0.024 | 0.318 | 0.515 | 0.714 | 0.964 |
| 1600 | 0.020 | 0.318 | 0.515 | 0.712 | 0.964 |
| 1800 | 0.020 | 0.315 | 0.514 | 0.710 | 0.966 |
| 2000 | 0.019 | 0.314 | 0.514 | 0.710 | 0.964 |

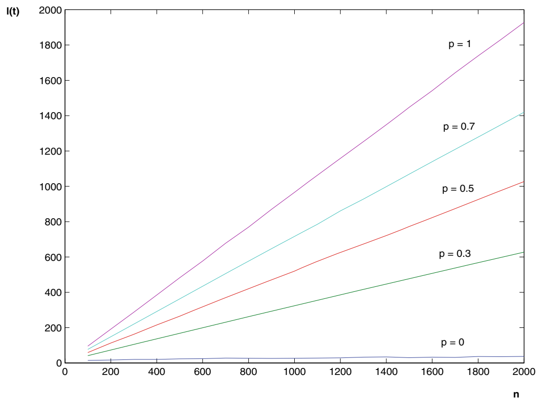
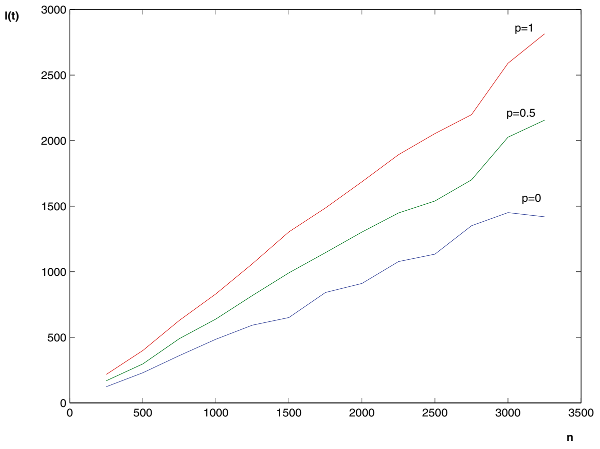


**Table 2.** Results for low density planar graphs

| n | p=0 | p=0.5 | p=1 |
|---|---|---|---|
| | $l(t)$ | $l(t)$ | $l(t)$ |
| 250 | 123.10 | 168.90 | 216.90 |
| 500 | 229.50 | 297.40 | 399.60 |
| 750 | 360.10 | 489.40 | 629.10 |
| 1000 | 485.20 | 639.60 | 831.40 |
| 1250 | 592.30 | 818.00 | 1060.70 |
| 1500 | 651.00 | 991.60 | 1304.10 |
| 1750 | 842.10 | 1145.70 | 1486.30 |
| 2000 | 910.30 | 1302.80 | 1686.10 |
| 2250 | 1077.20 | 1448.40 | 1892.60 |
| 2500 | 1134.10 | 1539.80 | 2053.50 |
| 2750 | 1350.70 | 1700.70 | 2198.10 |
| 3000 | 1451.30 | 2025.80 | 2590.20 |
| 3250 | 1418.80 | 2156.00 | 2814.40 |



the parameter $p$ in the same way as in the case of undirected graphs. The weighted graphs were constructed as follows. Firstly we construct a respective *st*-Hamiltonian unweighted graph.

Then we set a value $W$ to be an upper bound on the weights of the edges of the graph. We set the weights of the edges that lie on a hamiltonian path from $s$ to $t$ equal to $W$. Clearly, the maximum longest path length of an *st*-orientation that corresponds to such weighted graphs is $(n-1)W$. The weights of the remaining edges are uniformly distributed in $[1, W]$.

From Tables 1 and 2 we can see that, by using parameter $p$, we can influence the length of the longest path of the final *st*-oriented graph. It is remarkable that for the class of *st*-Hamiltonian graphs (Table 1) the computed length of longest path for a parameter $p$ is approximately $p(n-1)$. For the class of low density

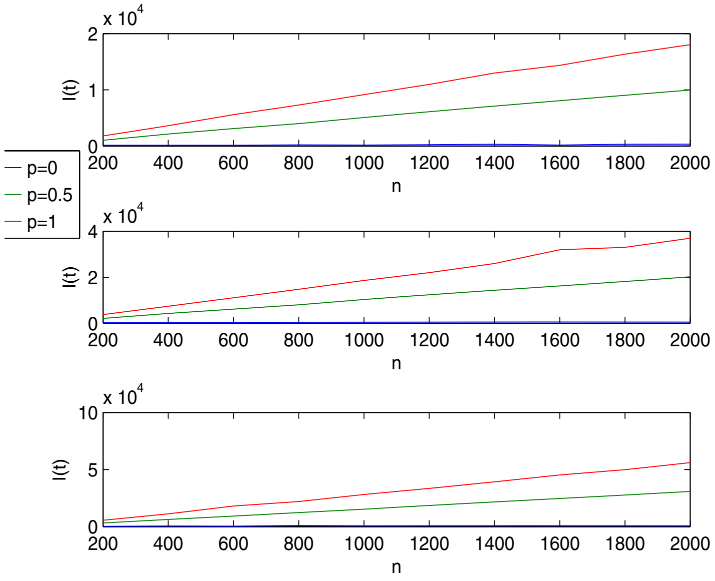


**Fig. 2.** Results for weighted graphs for $W = 10, 20, 30$ (up to bottom)

planar graphs (Table 2), the algorithm performs very well and indeed computes different *st*-orientations according to the parameter. Finally, for the class of the weighted graphs, note that the length of the longest path of the *st*-orientation is in absolute accordance with the value of the parameter $p$ and the value $W$. More resutls can be found at `http://www.csd.uoc.gr/~cpap/MSc_thesis.ps`.

As far as the applications of parameterized *st*-orientations in graph coloring are concerned, we have tested known benchmarks available at `http://mat.gsia.cmu.edu/COLOR/instances.html` (Table 3) and got good

**Table 3.** Some benchmark graphs for which MIN-STN has computed an optimal or near optimal coloring

| **file name** | **n** | **m** | **optimal coloring** | **MIN-STN (p=0) coloring** |
|---|---|---|---|---|
| games120.col | 120 | 368 | 9 | 9 |
| jean.col | 80 | 254 | 10 | 10 |
| huck.col | 74 | 301 | 11 | 11 |
| zeroin.i.1.col | 211 | 4100 | 49 | 49 |
| mulsol.i.3.col | 184 | 3916 | 31 | 31 |
| mulsol.i.1.col | 197 | 3925 | 49 | 49 |
| fpsol2.i.1.col | 496 | 11654 | 65 | 65 |
| miles250.col | 128 | 387 | 8 | 9 |
| anna.col | 138 | 493 | 11 | 12 |
| inithx.i.2.col | 645 | 13979 | 31 | 32 |

results for most of the tested graphs. Results are promising for other classes of graphs also.

## 7 Conclusions and Future Work

In this paper, we show that there is a very efficient way to control the length of the longest path of a produced *st*-orientation. In this way, we are able to produce more than one *st*-numberings of certain quality. This work is especially important from an applied point of view. The parameterized *st*-orientations can be used by various algorithms that use an *st*-numbering as their first step. In this way, better solutions to certain problems can be produced [1]. Concerning future work, some interesting questions that come out of this work are the following:(a) Can we prove that the presented algorithm may reach **any** possible *st*-orientation (note that the algorithm can also choose non-cutpoint nodes that belong to *some* block of the block-cutpoint tree)? and (b) Can we compute an *st*-orientation given a set of edges that have a predefined orientation (constrained *st*-orientation problem)? (c) Implementation of efficient data structures for the *t*-rooted block-cutpoint tree.

**Acknowledgements.** The authors would like to thank Franco P. Preparata (Brown University), Roberto Tamassia (Brown University) and Mihalis Yannakakis (Columbia University) for useful comments and discussions.

## References

1. C. Papamanthou and I.G. Tollis. Applications of parameterized *st*-orientations in graph drawing algorithms. In *LNCS Graph Drawing 2004*, volume 3843, pages 355–367, 2005.
2. R. Tamassia and I.G. Tollis. A unified approach to visibility representations of planar graphs. *Disc. and Comp. Geom.*, 1:321–341, 1986.
3. A. Papakostas and I.G. Tollis. Algorithms for area-efficient orthogonal drawings. *Computational Geometry: Theory and Applications*, 9:83–110, 1998.
4. G.D. Battista, P. Eades, R. Tamassia, and I.G. Tollis. Annotated bibliography on graph drawing algorithms. *Computational Geometry: Theory and Applications*, 4:235–282, 1994.
5. G.D. Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, 1999.
6. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-11(2):109–125, 1981.
7. F. Annexstein and K. Berman. Directional routing via generalized st-numberings. *SIAM J. Discrete Math.*, 13(2):268–279, 2000.
8. M. Mursalin Akon, S. Asaduzzaman, M.S. Rahman, and M. Matsumoto. Proposal for st-routing. *Tellecommunication Systems*, 25(3-4):287–298, 2004.
9. S. Nakano, M.S. Rahman, and T. Nishizeki. A linear-time algorithm for four-partitioning four-connected planar graphs. *Information Processing Letters*, 62:315–322, 1997.

10. A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *P. Rosenstiehl(ed.) Theory of Graphs: Tihany, Academic Press, New York*, pages 115–118, 1968.
11. S. Even and R. Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2:339–344, 1976.
12. J. Ebert. st-ordering the vertices of biconenected graphs. *Computing*, 30(1):19–33, 1983.
13. R. Tarjan. Two streamlined depth-first search algorithms. *Fundamentae Informatica*, 9:85–94, 1986.
14. P. Rosenstiehl and R. Tarjan. Rectilinear planar layout and bipolar orientation of planar graphs. *Discrete Comput. Geom.*, 1:343–353, 1986.
15. Y. Maon, B. Schieber, and U. Vishkin. Parallel ear decomposition search (eds) and st-numbering in graphs. *Theoret. Comput. Sci*, 47:277–298, 1986.
16. Ulrik Brandes. Eager st-ordering. In *LNCS ESA 2002*, volume 2461, pages 247–256, 2002.
17. H.D. Fraysseix, P.O. de Mendez, and P. Rosenstiehl. Bipolar orientations revisited. *Discrete Applied Mathematics*, 56:157–179, 1995.
18. C. Papamanthou and I.G. Tollis. *st*-numberings and longest paths, *manuscript, ICS-FORTH 2004*.
19. J. Hopcroft and R. Tarjan. Efficient algorithms for graph manipulation. *Comm. ACM*, 16:372–378, 1973.
20. J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge and biconnectivity. *J. ACM*, 48(4):723–760, 2001.
21. Tams Lukovszki and Willy-B. Strothmann. Decremental biconnectivity on planar graphs technical report, university of paderborn, tr-ri-97-186.
22. T. Gallai. On directed paths and circuits. In *P. Erdos(ed.) Theory of Graphs: International Symposium July 1966*, pages 215–232, 1967.

# Straight-Line Drawing of Quadrangulations

Éric Fusy

Algorithms Project (INRIA Rocquencourt) and LIX (École Polytechnique)
eric.fusy@inria.fr

**Abstract.** This article introduces a straight-line drawing algorithm for quadrangulations, in the family of the face-counting algorithms. It outputs in linear time a drawing on a regular $W \times H$ grid such that $W + H = n - 1 - \Delta$, where $n$ is the number of vertices and $\Delta$ is an explicit combinatorial parameter of the quadrangulation.

## 1 Introduction

A *plane graph* $G$ (also called planar map) is a graph embedded in the plane, defined up to continuous deformation. A major issue in graph drawing is to compute a *straight-line drawing* of $G$, i.e., to place vertices of $G$ at points of a regular $W \times H$ grid ($W$ being called the width and $H$ the height of the grid), so that the drawing obtained by linking adjacent vertices by segments is a planar drawing of $G$. An extensive literature is devoted to straight-line drawing of triangulations, i.e., plane graphs with faces of degree 3, and more generally of 3-connected plane graphs. Essentially there are two classes of linear time algorithms: a) purely iterative algorithms based on a specific order of treatment of vertices, the drawing being globally updated at each step [5,8,10]; b) one-shot algorithms, where the plane graph is first endowed with a combinatorial structure (e.g. Schnyder woods for triangulations), which is used to compute the coordinates of vertices, usually using some face-counting operations [11,2,6]. The algorithms of the second class have the advantage that the coordinates of vertices are computed independently, so that they are easier to implement and to perform on a piece of paper. In addition, the grid size can be expressed in terms of combinatorial parameters of the graph.

Surprisingly, little attention has been given to straight-line drawing algorithms dealing specifically with quadrangulations, i.e., plane graphs with all faces of degree 4 (these are also called maximal bipartite plane graphs). The only reference we found is an article by Biedl and Brandenburg [1], where it is shown that a quadrangulation $Q$ can be triangulated into a triangulation $T$ of the 4-gon with no separating triangle, so that the algorithm of Miura et al. [10] can be called to embed $T$ (hence, also $Q$) on a $(\lceil n/2 \rceil - 1) \times \lfloor n/2 \rfloor$ grid. This algorithm is linear and has small grid size, but it does not really use a combinatorial structure specific to quadrangulations. In contrast, the algorithm we introduce exploits a *quadri-partition* of the inner edges of a quadrangulation, presented in Section 3. This combinatorial structure is also investigated in [7] using another terminology (with labels) and is closely related to an edge bicoloration of a quadrangulation [4]. We use the quadri-partition to triangulate partially $Q$ into a plane graph $G$ endowed with a so-called *transversal structure*, i.e., a bicoloration and orientation of the inner edges of $G$ with specific local conditions. Transversal structures were originally only defined for triangulations of a 4-gon [6,9]. The definition is easily extended to *partially triangulated quadrangulations* in Section 2. Then, the transversal structure is used to draw $G$ (hence, also $Q$) with face-counting operations. To do this, we extend the algorithm of [6], which draws triangulations of the 4-gon, to partially triangulated quadrangulations. The obtained drawing of $Q$ verifies $W + H = n - 1 - \Delta$, where $n$ is the number of vertices and $\Delta$ is an explicit parameter of $Q$, see Theorem 1. Hence, the semi-perimeter is at most as large as in [1]. In addition, our algorithm has the advantage of being of the *one-shot* type.

## 2 Transversal Structures

Let $G$ be a plane graph with quadrangular outer face, the four outer vertices in cw order being denoted by $N$, $E$, $S$ and $W$ (like North, East, South, West). A *transversal structure* is an orientation and partition of the inner edges of $G$ into red and blue edges, satisfying the following conditions (see Figure 1(a)):

**C1: (Inner vertices)** Each inner vertex of $G$ is incident in cw order to: a non-empty interval of outgoing red edges, a non-empty interval of outgoing blue edges, a non-empty interval of ingoing red edges, and a non-empty interval of ingoing blue edges.

**C2: (Border vertices)** The inner edges incident to $N$, $E$, $S$, $W$ are ingoing red, ingoing blue, outgoing red, outgoing blue, respectively.

Transversal structures have been defined in [6,9] in the case where all inner faces are triangles. However, as we will see, other plane graphs can be endowed with a

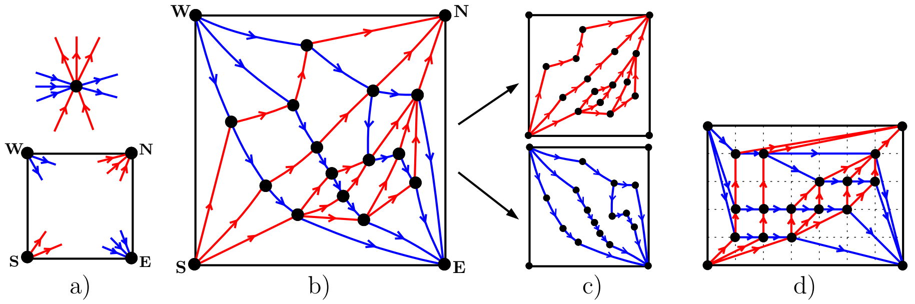


**Fig. 1.** The local conditions C1 and C2 for transversal structures (Fig. (a)), a partially triangulated quadrangulation $G$ endowed with a transversal structure (Fig. (b)), the associated red map and blue map (Fig. (c)), and the straight-line drawing of $G$ using TRANSVERSALDRAW(G) (Fig. (d))

transversal structure. Even if we do not need a precise condition of existence, let us mention that a necessary condition is that all inner faces have degree 3 or 4. Such a plane graph is called a *partially triangulated quadrangulation*. Let $G$ be a partially triangulated quadrangulation endowed with a transversal structure. Then, it is easily shown (adapting the proof of [6, Prop.1]) that the red edges of $G$ form a bipolar orientation with source $S$ and sink $N$; and the blue edges form a bipolar orientation with source $W$ and sink $E$. (We recall that a bipolar orientation is an acyclic orientation with a unique vertex having only outgoing edges, called *source*, and a unique vertex having only ingoing edges, called *sink*.) As developed in [6] for triangulated graphs, this property gives rise to a straight-line drawing algorithm TRANSVERSALDRAW(G), where the red edges are used to give abscissas and the blue edges are used to give ordinates. The *red map* (blue map) of $G$ is the plane graph $G_{\mathrm{r}}$ ($G_{\mathrm{b}}$) obtained by deleting all blue edges (red edges, respectively) of $G$, see Figure 1(c). Given an inner vertex $v$ of $G$, the *rightmost ingoing red path* of $v$ is the path $P_S(v) = (v_0 = v, v_1, \ldots, v_k = S)$ from $v$ to $S$ where, for each $i \in [0..k-1]$, $(v_i, v_{i+1})$ is the rightmost ingoing red edge of $v_i$, i.e., the unique ingoing edge of $v_i$ whose ccw consecutive edge around $v_i$ is outgoing. The *leftmost outgoing red path* of $v$ is the path $P_N(v) = (v_0 = v, v_1, \ldots, v_l = N\mathcal{P})$ where, for $i \in [0..l-1]$, $(v_i, v_{i+1})$ is the leftmost outgoing red edge of $v_i$. The *separating red path* $\mathcal{P}_{\mathrm{r}}(v)$ of $v$ is the concatenation of $P_N(v)$ and $P_S(v)$. Hence, $\mathcal{P}_{\mathrm{r}}(v)$ is a path from $S$ to $N$. We define similarly the *separating blue path* $\mathcal{P}_{\mathrm{b}}(v)$ of $v$, which goes from $W$ to $E$.

TRANSVERSALDRAW(G):

1. Take a regular grid of size $W \times H$, where $W$ ($H$) is the number of inner faces of the red map $G_{\mathrm{r}}$ (of the blue map $G_{\mathrm{b}}$, respectively).
2. Place the outer vertices $S$, $W$, $N$, $E$ at the grid corners $(0,0)$, $(0,H)$, $(W,H)$ and $(W,0)$, respectively.
3. For each inner vertex $v$ of $G$, let $x$ be the number of inner faces of $G_{\mathrm{r}}$ on the left of $\mathcal{P}_{\mathrm{r}}(v)$ and let $y$ be the number of inner faces of $G_{\mathrm{b}}$ on the right of $\mathcal{P}_{\mathrm{b}}(v)$. Place $v$ at the grid point of coordinates $(x,y)$.
4. Link each pair of adjacent vertices by a segment.

**Proposition 1.** *Given a partially triangulated plane graph $G$ endowed with a transversal structure,* TRANSVERSALDRAW(G) *outputs a straight-line drawing of $G$ in linear time. The semi-perimeter satisfies $W + H = n - 1 - \Delta$, where $n$ is the number of vertices and $\Delta$ is the number of quadrangular faces of $G$.*

*Proof.* The correctness proof of TRANSVERSALDRAW, given in [6, Theo.3] for the case of a triangulation of a 4-gon, adapts straightforwardly. The equality $W + H = n - 1 - \Delta$ is an easy consequence of Euler's relation. □

## 3 Edge Partition of a Quadrangulation

Let $Q$ be a quadrangulation, the four outer vertices in cw order denoted by $N$, $E$, $S$, and $W$. As all faces of $Q$ have even degree, the vertices of $Q$ can be greedily bicolored in black or white so that adjacent vertices have different colors. The bicoloration is unique up to the choice of the first vertex, hence there is a unique bicoloration such that $N$ and $S$ are black. Given $Q$ endowed with this bicoloration, the *primal map* of $Q$ is the plane graph $M$ whose vertices are the black vertices of $Q$, two black vertices being adjacent in $M$ if they are incident to the same face of $Q$. Hence, there is an edge of $M$ for each face of $Q$. Conversely, $Q$ is called the *angular map* of $M$ because each edge of $Q$ is associated with an angle of $M$. Observe also that each black vertex of $Q$ corresponds to a vertex of $M$ and each white vertex of $Q$ corresponds to a face of $M$, see Figure 2(b). As we consider quadrangulations with no double edge, it is well known that the primal map is 2-connected, i.e., the deletion of one vertex does not disconnect $M$. As detailed in [4], for any edge $(s, t)$ of $M$, there exists a bipolar orientation of $M$ with source $s$ and sink $t$, and there is a simple *sweeping algorithm* to compute such a bipolar orientation in linear time [3].

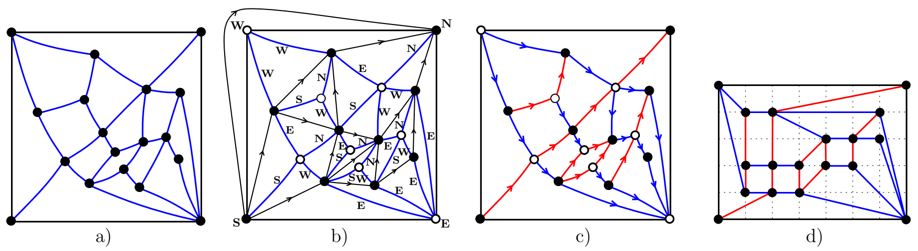


**Fig. 2.** A quadrangulation $Q$ (Fig. (a)), endowed with an angular partition of the inner edges (Fig. (b)), the associated uncomplete transversal structure (Fig. (c)), and the straight-line drawing of $Q$ (Fig. (d))

Given a bipolar orientation $X$ of $M$ with source $S$ and sink $N$, an angle $(e, e')$ of two edges of $M$ around a black vertex $v$ —with $e'$ following $e$ in cw order around $v$— is called an angle *of type* $N$, $S$, $W$, $E$ if $(e, e')$ are (ingoing, ingoing), (outgoing, outgoing), (ingoing, outgoing), (outgoing, ingoing), respectively. Accordingly, the inner edges of $Q$ are partitioned into $N$-edges, $S$-edges, $W$-edges and $E$-edges, depending on the type of their associated angle. This partition is called the *angular partition* of $Q$ associated with $X$, see Figure 2(b). An important property of a plane bipolar orientation is that the edges incident to an inner vertex are partitioned into a non-empty interval of ingoing edges and a non-empty interval of outgoing edges; and, dually, each face $f$ of $M$ has two particular vertices $S_f$ and $N_f$ such that the contour of $f$ consists of two non-empty oriented paths both going from $S_f$ to $N_f$, called *left lateral path* and *right lateral path* of $f$, respectively. Hence, each inner black vertex $v$ of $Q$ is incident to one $W$-edge $e_W$ and one $E$-edge $e_E$, which are separated by a possibly empty interval of $N$-edges in the cw sector between $e_E$ and $e_W$ and a possibly empty interval of $S$-edges in the cw sector between $e_W$ and $e_E$. Dually, each white vertex of $Q$, corresponding to a face $f$ of $M$, is incident to one $N$-edge $e_N$ (connected to $N_f$)

and one $S$-edge $e_S$ (connected to $S_f$), which are separated by a possibly empty interval of $W$-edges in the cw sector between $e_N$ and $e_S$ and a possibly empty interval of $E$-edges in the cw sector between $e_S$ and $e_N$. Observe also that all inner edges of $Q$ incident to $N$, $E$, $S$, and $W$ are $N$-edges, $E$-edges, $S$-edges, and $W$-edges respectively, see Figure 2(b).

## 4 The Algorithm

Let $Q$ be a quadrangulation endowed with an angular partition of the inner edges, associated with a bipolar orientation $X$ of the primal map $M$. The *uncomplete transversal structure* is the orientation and bicoloration of the inner edges of $Q$ where the $N$-edges and $S$-edges are colored red, the $W$-edges and $E$-edges are colored blue, the $S$-edges and $E$-edges are oriented from their black to their white vertex, and the $N$-edges and $W$-edges are oriented from their white to their black vertex. Clearly, the conditions of a transversal structure are satisfied, except that some of the four intervals of edges around an inner vertex may be empty. Precisely, if a black vertex $v$ of $M$ has $i \geq 1$ ingoing edges and $j \geq 1$ outgoing edges, then $v$ is incident in cw order to an outgoing blue edge (the $E$-edge $e_E$), an interval of $(i-1)$ ingoing red edges, an ingoing blue edge (the $W$-edge $e_W$), and an interval of $(j-1)$ outgoing red edges. Dually, if a face of $M$ has a left lateral path of length $i \geq 1$ and a right lateral path of length $j \geq 1$, then the associated white vertex of $Q$ is incident in cw order to an outgoing red edge (the $N$-edge $e_N$), an interval of $(j-1)$ outgoing blue edges, an ingoing red edge (the $S$-edge $e_S$), and an interval of $(i-1)$ ingoing blue edges.

We now describe an algorithm PartTriang(Q), which adds colored oriented edges to $Q$ so as to obtain a partially triangulated plane graph $G$ endowed with a transversal structure. An edge of $M$ is said to be *undeletable* if it is the unique outgoing edge of its origin or the unique ingoing edge of its extremity (or both). An edge of $M$ different from the edge $(S,N)$ is said to be *transitive* if it connects the two poles $N_f$ and $S_f$ of a face of $M$. Notice that an edge of $M$ can not be undeletable and transitive. The plane graph $G = \text{PartTriang}(Q)$ is obtained by adding to $Q$ the undeletable edges of $M$, colored red and oriented as in $X$; and by adding the edges dual to the transitive edges, colored blue and oriented from left to right, i.e., for each face of $Q$ associated with a transitive edge $e$ of $M$, a blue edge is added connecting the two white vertices of the face and oriented from the left of $e$ to the right of $e$. It is easily checked that the edge bicoloration and orientation of $G$ is a transversal structure, see the transition between Figure 2(b)–(c) and Figure 1(b).

**Theorem 1.** *Let $Q$ be a quadrangulation endowed with an angular partition of its inner edges. The algorithm that computes* G=PartTriang(Q)*, then calls* TransversalDraw(G)*, and finally deletes the edges added from $Q$ to $G$, is a straight-line drawing algorithm for quadrangulations with linear time complexity. All horizontal and vertical lines of the grid are occupied by at least one vertex. The semi-perimeter $W+H$ of the grid satisfies*

$$W + H = n - 1 - \Delta,$$

*where $n$ is the number of vertices and $\Delta$ is the number of alternating faces of $Q$, i.e., faces whose contour consists of two red edges and two blue edges that alternate. Alternating faces are strictly convex in the drawing.*

*Proof.* The result follows from Proposition 1 and from the fact that the faces of $Q$ that are not split into two triangles are the alternating faces. These faces are strictly convex because of the property (stated in [6] and also holding here) that red edges of TransversalDraw(G) are geometrically directed from bottom to top and weakly directed from left to right, while blue edges are geometrically directed from left to right and weakly directed from top to bottom. Finally, it is easily proved —adapting [6, Prop.12]— that a vertical (horizontal) line not occupied by any vertex corresponds to an edge $e = (v, v')$ of the red map (blue map, respectively) which is neither the rightmost ingoing edge of $v'$ nor the leftmost outgoing edge of $v$. Clearly, such edges do not exist in $Q$ and do not appear during the partial triangulation of $Q$. □

## References

1. Therese Biedl and Franz J. Brandenburg. Drawing planar bipartite graphs with small area. In *Proceedings of CCCG, Windsor*, pages 105–108, 2005.
2. N. Bonichon, S. Felsner, and M. Mosbah. Convex drawings of 3-connected plane graphs. In *Proceedings of Graph Drawing*, pages 287–299. Springer-Verlag, 2004.
3. H. de Fraysseix, P. Ossona de Mendez, and J. Pach. A left-first search algorithm for planar graphs. *Discrete Comput. Geom.*, 13:459–468, 1995.
4. H. de Fraysseix, P. Ossona de Mendez, and P. Rosenstiehl. Bipolar orientations revisited. *Discrete Appl. Math.*, 56(2-3):157–179, 1995.
5. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.
6. E. Fusy. Transversal structures on triangulations, with application to straight-line drawing. In *Proceedings of Graph Drawing, LNCS 3843*, pages pp. 177–188, 2005. Full paper with proofs available at `http://arxiv.org/abs/math.CO/0602163`.
7. C. Huemer and S. Kappes. A binary labelling for plane laman graphs and quadrangulations. In *Proceedings of EWCG, Delphi*, pages 83–86, 2006.
8. G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996.
9. G. Kant and Xin He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theoretical Computer Science*, 172(1-2):175–193, 1997.
10. K. Miura, S. Nakano, and T. Nishizeki. Grid drawings of four-connected plane graphs. *Disc. Comput. Geometry*, 26(2):73–87, 2001.
11. W. Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM Symposium on Discrete Algorithms*, pages 138–148, 1990.

# Visualizing Large and Clustered Networks

Katharina A. Lehmann and Stephan Kottler

University of Tübingen, Wilhelm-Schickard-Institute, Sand 14, 72076 Tübingen, Germany

**Abstract.** The need to visualize large and complex networks has strongly increased in the last decade. Although networks with more than 1000 vertices seem to be prohibitive for a comprehensive layout, real-world networks exhibit a very inhomogenous edge density that can be harnessed to derive an aesthetic and structured layout. Here, we will present a heuristic that finds a spanning tree with a very low average spanner property for the non-tree edges, the so-called *backbone* of a network. This backbone can then be used to apply a modified tree-layout algorithm to draw the whole graph in a way that highlights dense parts of the graph, so-called clusters, and their inter-connections.

## 1 Introduction

At first glance it seems prohibitive to visualize large and complex networks. The idea to represent these networks by suitable spanning trees and draw these trees instead of the whole graph, is a well-known approach, found, e.g., in [3,9,5]. In most of these cases it was assumed that the spanning tree was either given by the user or that the graph to draw was hierarchically organized and thus a spanning tree could be easily and more or less unambiguously derived. Here we will show that also the visualization of non-hierarchical networks is feasible with a spanning tree approach if the networks are clustered instead. A network is clustered if it can be decomposed into dense subgraphs that are only sparsely interconnected. In the past, this property has been used in various approaches, e.g., to analyse protein-protein-interaction networks or various social networks to find semantically connected subsets of vertices [4,8,13,12], to name but a few. We will show here, that this property can also be used to find a clear and computationally feasible layout for clustered graphs with more than 1,000 vertices and more than 10,000 edges. Actually computing a good partition can be computationally prohibitive, so our motivation is to decompose the graph into a set of *local* edges that are likely to be within clusters and a set of *global* edges that are likely to be between clusters. The decomposition into these sets has already been proven useful for drawing power-law graphs where the decomposition is derived by solving a network flow problem [1]. Our decomposition technique is based on finding a spanning tree that minimizes the distances between any two vertices connected by a non-tree edge, a so-called *backbone* of the graph. An edge whose endpoints have a large distance in the tree will be considered a global edge. We could show that finding the minimal spanning tree with this respect is

NP-hard by a reduction from *exact 3 cover*[1] [7]. Here, we will thus present two heuristics that yield very good initial backbones and an optional optimization step that can be used to improve the result.

A simple idea to draw a large graph is to take such a backbone and draw it with a tree layout algorithm, ignoring all non-tree edges. In most cases, this does not result in satisfying drawings. In our approach, non-tree edges will influence the order in which the children of a tree node are sorted, depending on the length of these edges in the backbone. This approach results in aesthetic drawings that reveal the large scale structure of the graph.

The paper is organized as follows: In Sec. 2 the needed definitions are given, the description of suitable backbones is given in Sec. 3. In Sec. 4 we then present a novel approach to draw large graphs based on backbones. We finish with a summary in Sec. 5.

## 2 Definitions

A graph is a pair $(V, E)$ with $V$ the set of vertices and $E \subseteq V \times V$ the set of edges, with $n := |V|$ the number of nodes, and $m := |E|$ the number of edges. We will assume that all graphs are free of self-loops, single-edged, undirected, and connected. The *neighborhood* $N(v)$ of a vertex $v$ is given by $N(v) := \{w | (v, w) \in E\}$ and its *degree* $deg(v)$ by the cardinality $|N(v)|$ of its neighborhood. A *path* $P(s, t)$ between vertices $s$ and $t$ is a set of edges $\{e_1, e_2, \ldots, e_k\} \subseteq E$ such that $e_1 = (s, v_1)$, $e_k = (v_{k-1}, t)$, and for all $1 < i < k$: $e_i = (v_{i-1}, v_i) \in E$. The *path length* of a path $P(s, t)$ in an unweighted graph is given by the number of edges $k$ in it. The *distance* $d(s, t)$ between two vertices $s, t$ is given by the minimal length of any path between them if existent and $\infty$ otherwise. $d_{E'}(s, t)$ denotes the distance of two vertices using only the edges in $E' \subseteq E$. A graph is a *tree* if there is exactly one path between any pair of vertices. A *spanning tree* $T$ of $G$ is here defined as a subset of edges that constitutes a tree on $V$.

In the following, we will often use the term *cluster*. We use this term as an abbreviation for *dense subgraph*. In this work we will not give a proper distinction when a subgraph will be called a cluster and when not but rather speak of subgraphs that are *more clustered* than others.

## 3 The Backbone of Complex Networks

To harness the clustered structure of a large graph for computing a layout, we will use an approach that is based on finding a good spanning tree of the graph: Let $T$ be a spanning tree of $G$ that defines weights $\omega_T(e)$ for all edges $e = (v, w) \in E(G)$ in the following way:

$$\omega_T((v, w)) = d_T(v, w) \tag{1}$$

$d_T(e)$ will also be called the *tree distance* of edge $e$. The quality $Q(T)$ of a spanning tree will be measured by the sum of the weights it assigns to the edges:

[1] Unpublished result by KAL and M. Kaufmann.

$$Q(T) = \sum_{e \in E(G) \setminus T} \omega(e) \tag{2}$$

The motivation behind this quality measure is that, given a dense cluster of the graph, $Q(T)$ will in most cases be smallest, if all vertices of this cluster are in a small and contiguous subtree of $T$. Otherwise, all edges between these vertices would have high tree distance values. In other words, the lower $Q(T)$ is, the more non-tree edges are 'local' edges between vertices that are not far away in the tree. Thus, a spanning tree with a low $Q(T)$ can be called a *backbone* of the graph since it represents clusters in a concentrated way. Since trees are planar, the hope is that most 'local' edges will also span short distances if they are added to a drawing of the backbone.

A trivial lower bound for $Q(T)$ is given by $2(m - n + 1)$. This lower bound is for example met by a clique if the spanning tree consists of one vertex and all incident edges. The following procedure computes a non-trivial lower bound that depends on the structure of the given graph: For every edge $e = (v, w)$ the distance $d_{E\setminus\{e\}}(v, w)$ is computed. Let $\Sigma(G)$ denote the sum of the $m - (n - 1)$ lowest values of $d_{E\setminus\{e\}}(v, w)$.

**Lemma 1.** *$\Sigma(G)$ is a lower bound for $Q(T)$ for any spanning tree $T$ in $G$.*

*Proof.* Let $T^*$ denote an arbitrary spanning tree with minimal $Q(T^*)$. Let $e$ be one of the $n - 1$ edges in $T^*$, then its weight does not contribute to $Q(T^*)$. If $e = (v, w)$ is not in $T$, $d_T((v, w))$ cannot be smaller than $d_{E\setminus\{e\}}(v, w)$. Since we do not know which edges will be in $T^*$, we disregard the $n - 1$ highest values of $d_{E\setminus\{e\}}(v, w)$ and thus, $\Sigma(G)$ is a lower bound for $Q(T^*)$. □

The quality of spanning trees with respect to $Q(T)$ can be very different, a fact that is shown in Fig. 1. Since finding the spanning tree with minimal $Q(T)$ is NP-hard as stated above we will now show greedy algorithms that compute reasonable initial backbones that can subsequently be improved by a local optimization heuristic.

## 3.1 Computing an Initial Backbone

To construct a backbone, the most simple idea is to choose one vertex at random and start a breadth first search and to mark the edge by which a vertex is first explored as tree edge. The quality $Q(T)$ of the resulting backbone is reasonably good compared to the above proposed quality measure $\Sigma_G$ and the tree can be computed in $O(m)$. We will introduce two other methods that are computationally more involved but yield much better backbones in practice. Both heuristics grow a spanning tree $S$ incrementally by first choosing the next vertex $v$ to append to $S$ and then choosing the best edge to hook $v$ into $S$. Both start with one vertex chosen at random. With $S$ the set of vertices already in the tree, let $R$ denote the set of vertices $v \in G \setminus S$ directly connected to at least one node in $S$. The vertex to append next is the vertex with maximal degree of $R$, where ties are broken in favor of the vertex with maximal number of neighbors in $S$; remaining ties are then broken at random. The intuition behind this heuristic

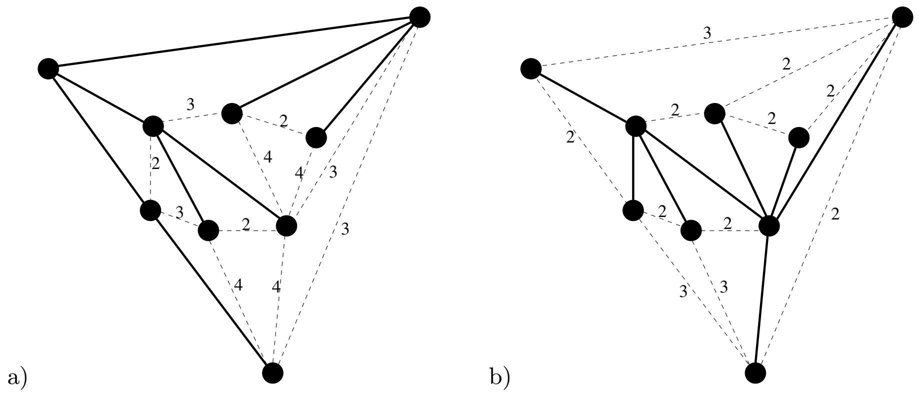


**Fig. 1.** The thick lines denote two different spanning trees $T$ for the given graph. Numbers next to a (dotted) non-tree edge denote the tree distance of this edge. The spanning tree in a) has a quality $Q(T)$ of 34 and the spanning tree in b) has a $Q(T)$ of 25.

is that vertices that are appended early to the growing backbone will influence the backbone's structure most. Since a vertex with a high degree will contribute a large sum of backbone distances to $Q(T)$, these nodes should have a large influence and thus be appended as early as possible. A trivial implementation searches for the vertex to append in $O(n)$ in every step, yielding a runtime of $O(n^2)$ for all steps. A more sophisticated data structure that keeps vertices in $R$ sorted in a kind of two-dimensional array of lists, can reduce this runtime to $O(n\ deg^*)$, where $deg^*$ is the maximal degree in the graph. For very large real-world networks this is in most cases a significant improvement.

In general, the chosen vertex $v$ will have more than one neighbor in $S$ and its tree edge will connect it to one of them. These neighbors are the possible *hooks* of $v$. Note that by choosing one of the edges to some hook to be $v$'s tree edge, the tree distances of all the possible tree edges of $v$ are determined. Thus, the first variant, the *minimized inner distance tree*, will choose that hook that minimizes the tree distances of all the other possible tree edges:

*Minimized Inner Distance Tree.* Let $S(v)$ denote the neighbors of the chosen vertex $v$ in $S$, i.e., the hooks of $v$. Since only one of the edges incident to a hook can be a tree edge without inducing a cycle in $T$, it is necessary to choose the one hook $h^*$ that minimizes the tree distances of all the other edges to hooks. Thus, for every hook the distance to all other hooks is summed up and the edge to the hook with the minimal distance to all other hooks is chosen as new tree edge (Fig. 2 **a**).

By holding an array $D(T)$ of size $n^2$ that keeps the distance $d_T(s,t)$ for all vertices $s, t$ in $S$, this computation can be done in $O((deg^*)^2)$. After the best hook $h^*$ has been chosen, this data structure has to be updated by adding the distances $d_T(v,w)$ between the newly added vertex $v$ and all other vertices $w$ in $S$ to $D(T)$. Since $d_T(v,w) = d_T(h^*,w) + 1$ for all $w \in S$, this can be done in

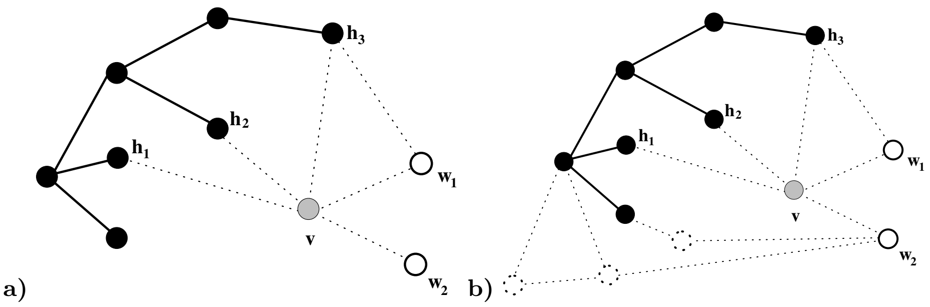


**Fig. 2. a)** Minimized Inner Distance Tree: Entering node $v$ has three hooks $h_1, h_2, h_3$. $h_2$ minimizes the sum of the tree distances of $v$'s edges to $h_1, h_3$ with a sum of 8, and thus $h_2$ is $h^*$. **b)** The tree distance of $v$'s edges to $w_1, w_2$ can be estimated by determining the distance of the hooks to these neighbors. It follows that $h_3$ has the best sum of distance to all others: $|P(h_3, h_1)| = 4$, $P(h_3, h_2)| = 3$, $|P(h_3, w_1)| = 5$, $|P(h_3, w_2)| = 1$.

$O(n)$. Thus, the entire runtime to construct a minimized inner distance tree is given by $O(n(deg^*)^2 + n^2)$.

**Lemma 2.** *A minimized inner distance tree for some randomly chosen root node can be computed in* $O(n(deg^*)^2 + n^2)$.

While this tree only regards those (inner) edges to other vertices in $S$, the next one tries to estimate the tree distance of the other edges of $v$ as well:

*Minimized Entire Distance Tree.* Let again $S(v)$ denote the neighbors of the chosen vertex $v$ in $S$, and $N(v)$ denote the full neighborhood of $v$ in $G$. For those edges of $v$ that do not lead directly to vertices in $S$, it is hard to estimate their tree distance: It could be that they will later choose $v$ as their hook to the growing tree and in this case an edge will not contribute to $Q(T)$. Since it is unlikely that all of them will use the edge to $v$ as their tree edge, it would be good to choose a hook $h^*$ such that all neighbors $w$ of $v$ have a short alternative path $P'(h^*, w)$ to $v$: A path $P'(h, w)$ is considered as an alternative if it can be split into two paths, the first using only vertices of $S$, the second -if necessary- only vertices of $V \backslash S$. In this way, the currently known structure of the tree is used as much as possible and the edges that are not yet known to be in the tree are only used for the last bit to reach $w$ (Fig. 2). With this intuition, we will choose the hook $h^* \in S(v)$ that minimizes the following sum:

$$\sum_{w \in N(v)} |P'(h, w)| \tag{3}$$

Note that the sum in Equ. 3 contains also the sum of the inner distances and thus the name of the tree is justified. A summary of the attachment procedure *hookIntoTree(E, T, E_S, v)* is given in [10].

This computation can be done by computing the distance of all vertices to every hook of $v$ which can be accomplished in $O(m\ deg^*)$. It follows that a minimized entire distance tree can be computed in $O(nm\ deg^*)$.

**Lemma 3.** *A minimized entire distance tree for some randomly chosen root can be computed in $O(n\ deg^* m)$.*

Table 1 shows a comparison of all three trees for some real-world networks. It is clearly visible that the higher computational effort for minimized inner distance and minimized entire distance trees results in much better backbones than the simple BFS tree and come near to the lower bound given by $\Sigma(G)$. However, even a good initial backbone can still be improved by the following optimization heuristic.

## 3.2 Optimization of the Backbone

The following steps allow for a local optimization of the initially computed backbone $T$. The main idea is that any edge $e$ that is not in $T$ would induce a cycle if it was added to $T$. By removing any other edge $f$ of this cycle, a new spanning tree $T'(e,f) := (T \cup e) \backslash f$ results. If no ambiguity is given we will reduce $T'(e,f)$ to $T'$ in the following. If $Q(T')$ is smaller than $Q(T)$, than $e$ should replace $f$ in $T$. We will call $e$ the *entering edge* and $f$ the *leaving edge*. To analyze whether $Q(T')$ is smaller than $Q(T)$, the following definitions are helpful: Let $e$ be any non-tree edge, then $P_T(e)$ denotes the path in $T$ that connects the end vertices of $e$, the so-called *tree path* of $e$.

**Proposition 4.** *For all non-tree edges $i$ with $f \notin P_T(i)$, $d_T(i)$ will not be changed.*

*Proof.* Since all edges of $P_T(i)$ are still in $T$, $d_T(i)$ cannot be increased. Let's assume that $d_T(i)$ is decreased by the insertion of $e$. This means that there is a second path connecting the end vertices of $i$, violating the tree property of $T$. □

Let $i$ denote some non-tree edge whose tree path contains at least one of the edges of $P_T(e)$, and let $C_T(i,e)$ denote the set of shared edges:

$$C_T(i,e) := P_T(i) \cap P_T(e) \tag{4}$$

If the leaving edge $f$ is in this set, the tree path of $i$ will be altered. To describe the change, the following definitions are needed (Fig. 3 **a**): Let $C_T(e)$ denote all edges in the cycle that is introduced by adding $e$ to $T$. Note that $C_T(e)$ is given by $P_T(e) \cup \{e\}$. Let $\overline{C_T(i,e)}$ denote the complement of $C_T(i,e)$ in cycle $C_T(e)$. The new tree path $P_{T'}(i)$ is then given by

$$P_{T'}(i) = P_T(i) \cup \overline{C_T(i,e)} \setminus C_T(i,e). \tag{5}$$

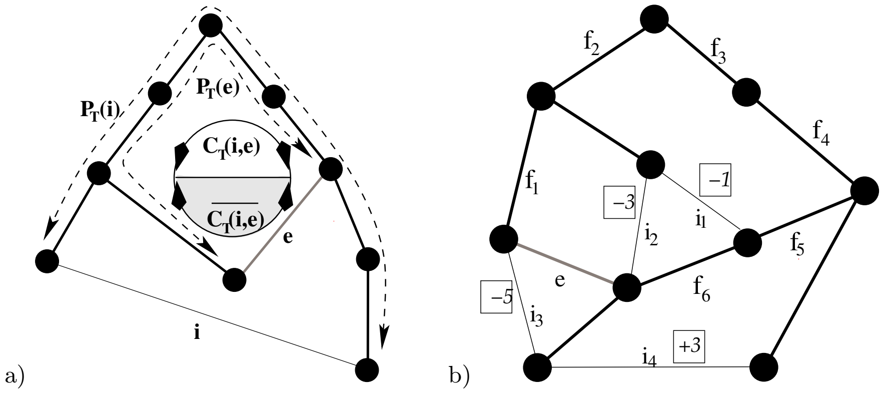


**Fig. 3. a)** $e$ is the entering edge, the tree paths $P_T(e)$ and $P_T(i)$ of some other non-tree edge $i$ are indicated by the dotted arrows. Every non-tree edge $i$ with $C_T(i,e) \neq \emptyset$ will have to change its tree path if the leaving edge is element of $C_T(i,e)$. The new tree path is built by removing from the old tree path all edges from $C_T(i,e)$ and adding the complement of the circle, i.e., $\overline{C_T(i,e)}$, to it.**b)** Again, $e$ is the entering edge, $i_j$ are edges that could be affected by choosing some of the possible leaving edges $f_i$. The boxed numbers give the difference between the new and old tree distance. It follows that for entering edge $e$, $f_2, f_3$, or $f_4$ would yield the best optimization with a value of $\Delta Q(T,e,f)$ of $-9$.

Note that this new tree path is always the same for any fixed non-tree edge $i$, independent of the identity of the leaving edge $f$ as long as $f \in C_T(i,e)$ (s. Fig. 3 **b**). Thus, $\Delta d_T(i,e) := d_{T'}(i) - d_T(i)$ is given by:

$$\Delta d_T(i,e) = |\overline{C_T(i,e)}| - |C_T(i,e)| \tag{6}$$
$$= |C_T(e)| - 2|C_T(i,e)| \tag{7}$$

With $I_e(f)$ denoting the set of non-tree edges $i$ with $f \in C_T(i,e)$, we can now state the following lemma:

**Lemma 5.** *For fixed entering edge e and leaving edge f, the difference in $Q(T)$ denoted by $\Delta Q(T,e,f)$ can be computed by:*

$$\Delta Q(T,e,f) = \sum_{i \in I_e(f)} \Delta d_T(i,e,f) \tag{8}$$

$\Delta(Q(T,e,f))$ can be computed efficiently by first determining the set $I(e) = \cup_{f \in P_T(e)} I_e(f)$ of all edges $i$ that are depending on at least one edge of $C_T(e)$ in their tree path. This can be done very efficiently if every tree edge $f$ stores $I_e(f)$ in a bit map. A bit map allows space and time efficient set operations, e.g., conjunctions and disjunctions. With at most $n$ sets $I_e(f)$, the set $I(e)$ can be computed in $O(nm)$. The tree path $P_T(i)$ of every non-tree edge $i$ is also stored as bits in a bit map. By simple $OR-, XOR-$, and $AND$-Operations all required sets $C_T(i,e)$, $\overline{C_T(i,e)}$, and $\Delta d_T(i,e)$ can be computed in $O(m)$ for a single non-tree edge $i$ and in $O(m^2)$ for all of them. The leaving edge is the edge $f$ with minimal $\Delta Q(T,e,f)$, which can be computed in $O(nm)$ where ties are broken at random. If there is no leaving edge because all resulting trees $T'$ would be worse, nothing will happen and the next entering edge $e$ is chosen at random. A summary of this algorithm is given in [10]. After $e$ and $f$ have been chosen in this way, some updates have to be done that are also computed very efficiently by operations on the bit maps. These updates can then be computed in $O(m^2)$.

**Lemma 6.** *A single local optimization step can be computed in* $O(m^2)$.

Table 1 shows that the optimization is able to decrease the already good $Q(T)$ of an *minimized entire distance tree* significantly towards the lower bound. The table also gives the time spent on the optimization, showing that there is a trade-off between the wanted quality of the backbone and the time spent on its computation.

## 4 Using the Backbone for Computing a Layout

As indicated above, a good backbone will try to concentrate the vertices of any cluster on a small, connected subtree. By doing so, the tree also indicates that edges with a high tree distance are more likely to be inter-cluster edges. These properties of the backbone can be used for computing a layout that co-locates the vertices that are supposedly in a dense part of the graph and simultaneously highlights the inter-connections between these dense parts.

To harness the backbone, our layout approach is based on a tree layout that is adapted towards the needs of a full graph. The layout of the graph can be computed by a variation of the balloon tree layout [3], resulting in a drawing which we will call a *backbone balloon drawing.* In the original balloon drawing of a tree, every subtree is enclosed entirely in a circle that is positioned in a wedge whose end-point is the parent node of this subtree. The radius of each circle is proportional to the number of vertices in the subtree.

To adapt this tree layout towards the needs of a full graph, the basic idea is to use the backbone and compute a balloon drawing for it and re-insert all non-tree edges as straight lines. To make this drawing a good drawing for the whole graph, the only parameter to change is the order of the children of any vertex in the tree. Since all direct neighbors of any vertex in the tree are positioned in a circle, the order of these children can be determined by a variation of the algorithm for crossing reduction in circular layouts [2]. The original algorithm is composed of two phases: In the first phase an initial ordering is heuristically determined. This is optimized by subsequent rounds of local sifting, where each vertex can try to improve the number of crossings by changing its position in the order computed so far. The application of this algorithm in a backbone balloon drawing requires the following two modifications:

1. Every edge between the children of a vertex in the tree can not only cross with each other, but also with the spokes, i.e., the edges from the father to its children. This changes the computation of the resulting number of crossings slightly.
2. Let $T(v)$ and $T(w)$ denote the subtrees rooted at $v$ and $w$, respectively, and let $v$ and $w$ be children of the same vertex. If the number of edges between these subtrees is large, then $v$ and $w$ should be close in the resulting order which is of course not regarded in the original algorithm.

The second point can be dealt with by introducing additional edges between any two children $v, w$ whose subtrees are connected by edges. Additionally, all edges will be assigned weights that present the number of edges between $T(v)$ and $T(w)$. The weight of a crossing between two edges is now given by the sum of the weights of the crossing edges, and the optimization goal is to minimize the sum of the weights of all crossings and not to minimize the number of crossings. The weights of all the edges between any two children can be computed in $O(nm)$. Every round of local sifting in a given circle with at most $deg^*$ vertices can be computed in $O((deg^*)^2)$ as shown in [2]. Since there are at most $n$ circles in the drawing, this sums up to $O(n(deg^*)^2)$ which is the largest factor in computing the backbone balloon drawing.

## 4.1 Experiments

We have applied the above presented variant of the balloon layout algorithm on different types of networks, shown in detail in [10]. Here, we show exemplary one network, a so-called *Amazon recommendation* network. To derive it, we start at some book that is offered by the Internet bookshop www.amazon.com and follow the links presented under the title "customers who bought this book also bought". By recursively following these links, very large and complex networks can be created. By construction, the outdegree of every vertex in the network is bounded by 6. The network shown here starts at [11] (Fig. 5). The balloon tree drawing shows discernible clusters connected by long-range edges, that are even more pronounced in the drawing that is based on an optimized backbone with minimized entire distance. This visual impression is supported by the fact that the force-directed drawing has the highest (normalized) total edge length of 434813, the one based on the unoptimized backbone has a total edge length of 321292 and the one based on the optimized backbone has a total edge length of 220857.

To show the quality of the different backbone heuristics and the optional optimization step, we have conducted experiments on this Amazon recommendation network and two other networks, shown in Table 1. For the creation of the Live Journal network a crawl was started at some participant of www.livejournal.com, following the links to designated friends unto depth 3. The co-authorship network is described in [14]. Fig. 4 gives a showcase for the improvements of $Q(T)$ by the optimization heuristic. It is clearly visible that the time spent in this step is worth the effort.

**Table 1.** For every network, 10 instances of every kind of spanning tree were computed. Displayed is the average $Q(T)$, its deviation, and the average time and its deviation to compute the tree. Note that the best unoptimized spanning trees already have a quality that is close to the lower bound given by $\Sigma(G)$ that can nonetheless be further reduced by the optimization. Furthermore, every of those 10 instances started at another, randomly chosen start vertex. The low deviation in $Q(T)$ shows that the method gives a stable $Q(T)$, independent of the choice of the start vertex. The experiments were conducted on a Pentium 4 with 3.2 GHz and 2GB RAM.

| **Graph** | **BFS** | **Minimized Inner Distance** | **Minimized Entire Distance** | **Optimized** | $\Sigma(G)$ |
|---|---|---|---|---|---|
| Amazon recommendation network $n = 3437$ $m = 9671$ | $31342 \pm 316$ $73 \pm 11$ [ms] | $21819 \pm 57$ $358 \pm 34$ [ms] | $20654 \pm 24$ $70 \pm 0.3$[s] | $17596 \pm 28$ 20min14s | 12468 |
| Live Journal $n = 3763$ $m = 11149$ | $29615 \pm 1332$ $65 \pm 11$[ms] | $23156 \pm 71$ $284 \pm 19$[ms] | $22058 \pm 38$ $100 \pm 0.4$[s] | $19588 \pm 3$ 22min12s | 14774 |
| Co-Authorship Network $n = 12357$ $m = 19448$ | $52896 \pm 1447$ $131 \pm 18$ [ms] | $52463 \pm 222$ $480 \pm 34$[ms] | $49951 \pm 98$ $337 \pm 0.6$[s] | $34287 \pm 47$ 49min2s | 14184 |

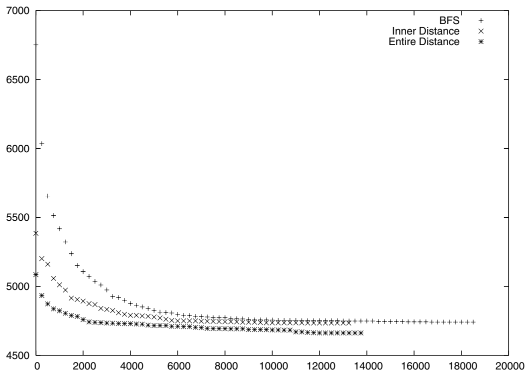


**Fig. 4.** For a smaller Amazon recommendation network with $n = 852$ and $m = 4220$, one BFS, one minimized inner distance and one minimized entire distance tree was computed and improved by the optimization heuristic until no further improvements could be found, i.e., a local minimum is reached. The trees start and end with the following $Q(T)$'s: 6572/4641 (BFS), 5385/4734 (Inner), and 5085/4662 (Entire), respectively. Note that $\Sigma(G)$ is 3822.

## 5 Summary

In this paper we presented a new quality measure $Q(T)$ for a spanning tree that helps to visualize large and clustered networks. We have shown that spanning

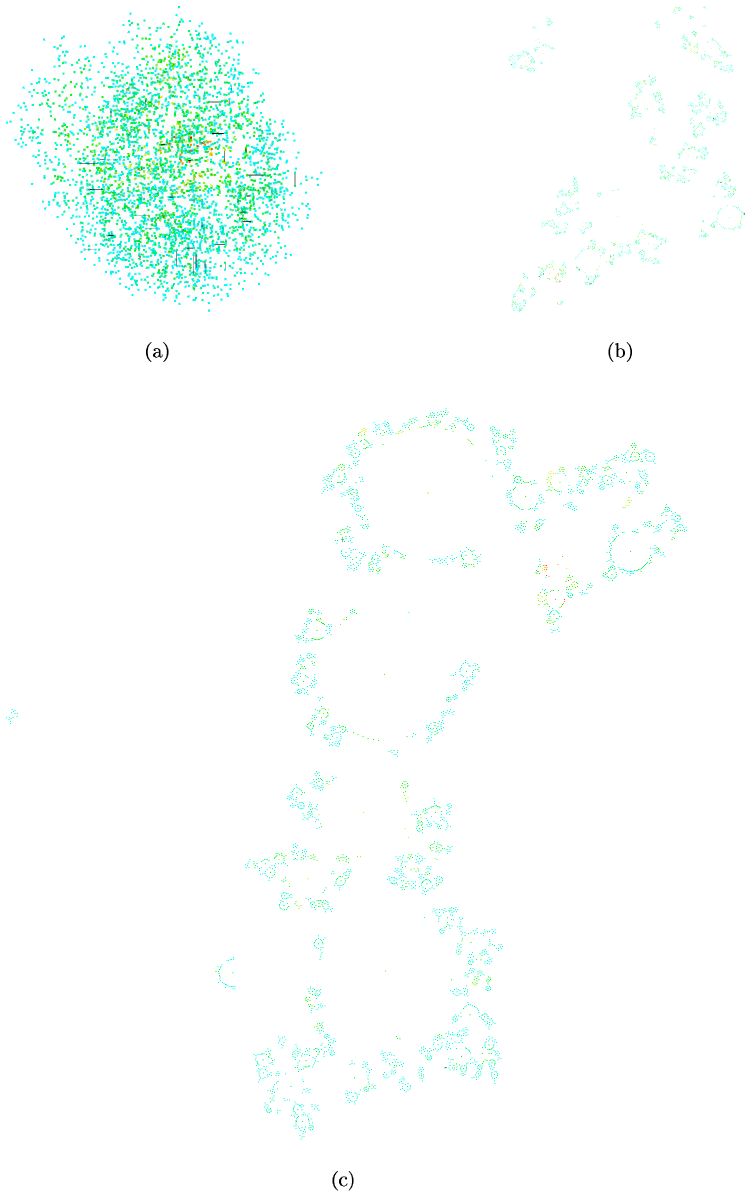


**Fig. 5. a)** A layout based on a force-directed approach, implemented by [15]. The normalized total edge length of this drawing is 434813. **b)** A balloon layout drawing based on a simple, unoptimized backbone with minimized entire distance. The normalized total edge length of this drawing is 321292. **c)** A balloon layout drawing based on an optimized backbone with minimized entire distance. The normalized total edge length of this drawing is 220857. The time to compute this layout was on average around 20 minutes (averaged over 5 drawings).

trees with a low $Q(T)$ can be computed in reasonable time and that these can be improved further by a local optimization heuristic. These trees or backbones can then be used to derive variations of classic layouts that are suitable for clustered graphs. Looking at the resulting drawings, a further application is to use these drawings as a basis for a geometric clustering method. First experimental evidence shows that a new, geometric variation of the Girvan-Newman-Clustering [8] applied to the drawings yields partition with a high modularity value [10] while being much faster computed than in the original approach. Further work will have to show whether backbones can also be used to adapt other drawings, such as the hierarchical Sugiyama drawing, or maybe build the basis for new approximation algorithms.

## References

1. Reid Andersen, Fan Chung, and Linyuan Lu. Drawing power law graphs. In *Proceedings of the 12th Symposium on Graph Drawing (GD'04)*, 2004.
2. Michael Baur and Ulrik Brandes. Crossing reduction in circular layouts. In *Proceedings of the 30th Workshop on Graph-Theoretic Concepts in Computer Science (WG'04)*, 2004.
3. J. Carriére and R. Kazman. Interacting with huge hierarchies: Beyond cone trees. In *Proceedings of the ACM conference on Information Visualization 1995*, pages 74–81, 1995.
4. I. Derényi, G. Palla, and T. Vicsek. Clique percolation in random networks. *Phys. Rev. Lett.*, 94:160202, 2005.
5. Jean-Daniel Fekete, David Wang, Niem Dang, Aleks Aris, and Catherine Plaisant. Overlaying graph links on treemaps. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'03)*, 2003.
6. T.M.J. Fruchtermann and E.M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.
7. Michael R. Garey and David S. Johnson. *Computers and intractability.* W.H. Freeman and Company, New York, 1979.
8. Michelle Girvan and M.E.J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99:7821–7826, 2002.
9. Ivan Herman, Guy Melançon, Maurice M. de Ruiter, and Maylis Delest. *Lecture Notes in Computer Science*, chapter Latour - A tree visualization system, page 392ff. Springer Verlag, Berlin, 2000.
10. Katharina A. Lehmann and Stephan Kottler *Visualizing Large and Clustered Networks.* Technical Report of the Wilhelm-Schickard-Institut, WSI-2006-06, ISSN 0946-3852, September 2006.
11. Mark Newman, Albert-Laszlo Barabasi, and Duncan J. Watts. *The structure and dynamics of networks.* Princeton University Press, 2006.
12. Andreas Noack. An energy model for visual graph clustering. In *Proceedings of the 11th International Symposium on Graph Drawing (GD'03)*, 2004.
13. Andreas Noack. Energy-based clustering of graphs with nonuniform degrees. In *Proceedings of the 13th International Symposium on Graph Drawing (GD'05)*, 2005.
14. G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814, 2005.
15. OrganicLayouter in the yFiles library *Sebastian Mueller.* www.yworks.com, Version 2.2.

# Partitioned Drawings*

Martin Siebenhaller

Universität Tübingen, WSI für Informatik, Sand 13,
72076 Tübingen, Germany
siebenha@informatik.uni-tuebingen.de

**Abstract.** In this paper we consider the problem of creating partitioned drawings of graphs. In a partitioned drawing each vertex is placed inside a given partition cell of a rectangular partition of the drawing area. This problem has several applications in practice, e.g. for UML activity diagrams or wiring schematics. We first formalize the problem and analyze its complexity. Then we give a heuristic approach which is based on the topology-shape-metrics approach and produces partitioned drawings in time $O((|V|+c)^2 \log(|V|+c))$, where $c$ denotes the number of crossings.

## 1 Introduction

In the area of graph drawing there are several approaches for drawing clustered graphs [2]. In a cluster drawing all vertices of a cluster are placed inside the same closed region (usually a rectangle). Regions could be nested, their positions are not given as input. In this paper, we consider the problem of placing each vertex inside a predetermined partition cell of a rectangular partitioned drawing area. Partition cells have fixed positions and do not overlap.

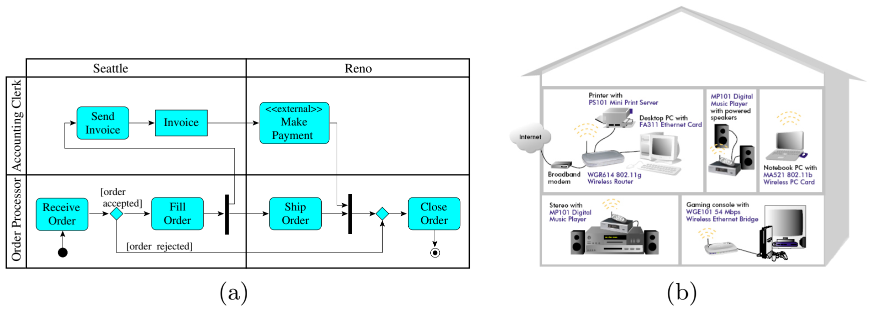

Seattle
Reno
Accounting Clerk
Order Processor
Send Invoice
Invoice
<<external>> Make Payment
Receive Order
[order accepted]
Fill Order
Ship Order
Close Order
[order rejected]
(a)
Internet
Printer with PS101 Mini Print Server
Desktop PC with FA311 Ethernet Card
Broadband modem
WGR614 802.11g Wireless Router
MP101 Digital Music Player with powered speakers
Notebook PC with MA521 802.11b Wireless PC Card
Stereo with MP101 Digital Music Player
Gaming console with WGE101 54 Mbps Wireless Ethernet Bridge
(b)

**Fig. 1.** Partitioned drawings: (a) shows an UML activity diagram taken from the UML 2.0 specification and (b) a wiring schematic taken from www.netgear.de

Simple partitions use only vertical or horizontal swim-lanes (stripes) to subdivide the drawing area. They are especially useful when we have to emphasize

* This work has been supported by DFG-grant Ka812/8-2.

a logical flow or time flow in a drawing. In UML activity diagrams, partitions are often used to divide a diagram into logical areas, e.g. organizational units in a business model. Activity diagrams also offer more complex, grid like partitions which are a combination of horizontal and vertical swim-lanes (see Figure 1(a)). Irregular partitions (see Figure 1(b)) are often used to indicate geometric information or positions.

The paper is organized as follows: In Section 2 we give a formal definition of the partitioned drawing problem followed by some theoretical results. A heuristic approach is presented in Section 3. We conclude the paper with a short discussion.

## 2 Definitions

In the following, we assume that the reader is familiar with the concept of planarity, the topology-shape-metrics approach for orthogonal graph drawings and Sugiyama's approach for layered graph drawings, see e.g. [2].

Let $A_R$ denote the (rectangular) drawing area. A *(rectangular) partition $P_R$* of $A_R$ is a partition of $A_R$ into a set $R = r_1, .., r_k$ of non-overlapping rectangles (=partition cells). Figure 2(a) gives an example. The corresponding *partition grid graph $P_G$* is constructed by placing a vertex on each point where a horizontal segment touches or intersects a vertical segment. $P_G$'s underlying structure is a regular grid graph, which enables us to assign grid coordinates to the vertices as shown in Figure 2(b). The largest vertical (horizontal) coordinate is denoted by $Y_{\max}$ ($X_{\max}$). For each partition cell $r \in R$ let $r^t$, $r^b$, $r^l$ and $r^r$ represent the grid coordinate of the top, bottom, left and right border respectively (e.g. for partition cell $r_4$ in Figure 2 is $r_4^t = 2$, $r_4^b = 3$, $r_4^l = 3$ and $r_4^r = 4$). In our approach those coordinates do not indicate distances. Only the topology and shape of $P_G$ has to be preserved, the size of the partition cells is not fixed.

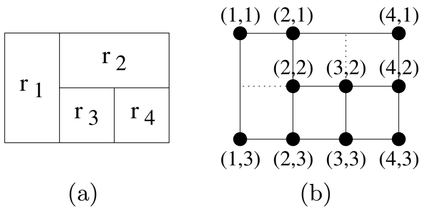


**Fig. 2.** A rectangular partition (a) and the corresponding partition grid graph (b)

Let $G = (V, E)$ denote an undirected graph, $P_R$ a rectangular partition and $p: V \to R$ a function that maps each vertex to a partition cell.

**Definition 1.** *A drawing of $G$ is called* partitioned drawing*, if each vertex $v \in V$ is drawn inside $p(v)$. $G$ is called* p-planar *if it has a partitioned drawing without edge crossings.*

**Theorem 1.** *A graph $G$ is p-planar if and only if it is planar. A (polygonal) p-planar embedding of a p-planar graph can be constructed in time $O(|V|^2)$.*

*Proof.* A p-planar graph is planar by definition. Let us assume that each vertex $v \in V$ is assigned to an arbitrarily distinct location inside $p(v)$. Pach and Wenger [8] showed that every planar graph admits a planar embedding which maps each vertex to an arbitrarily prescribed distinct location and each edge to a polygonal curve with $O(|V|)$ bends. This embedding can be found in $O(|V|^2)$ time [8] and implies that each planar graph is p-planar. □

An example for such an embedding is given in Figure 3(b). This result has several consequences: Since testing a graph for planarity can be done in linear time [7], the same is true for testing p-planarity. The problem of finding a planarization of a non-planar graph with a minimum number of crossings is NP-hard [6]. Thus the same holds for finding a p-planarization for those graphs.

For an orthogonal partitioned drawing of a graph $G = (V, E)$ we can state the following: if we do not prescribe an embedding, $G$ can always be drawn with less or equal than one bend per edge (omitting self-loops) and thus $\leq |E|$ bends at all. Note, that this bound is tight. A drawing with one bend per edge can simply be realized by placing each vertex $v \in V$ inside $p(v)$ such that there is no pair of vertices having the same x-coordinate (y-coordinate). Then the edges can be routed as in Figure 3(c). This strategy produces only few bends but it does not observe the number of crossings and thus often produce unsatisfying results. For a fixed embedding, the orthogonalization would often produce a lot of bends and strange edge routes.

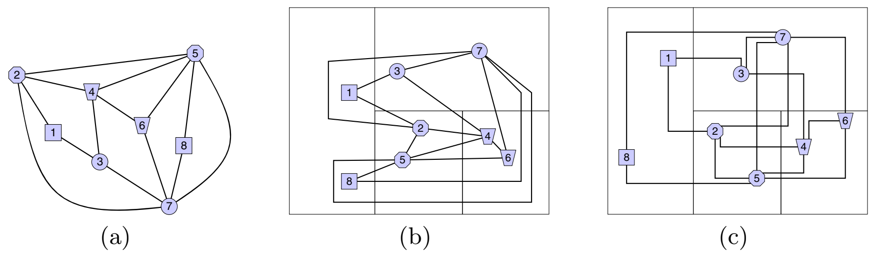


**Fig. 3.** (a) shows a planar graph and (b) the corresponding p-planar embedding using the partition of Figure 2(a). (c) shows a partitioned drawing with one bend per edge.

In practice we need to find a compromise between minimizing the number of crossings and minimizing the number of bends. Furthermore, we have to incorporate other requirements like vertices of prescribed size or edge labeling.

## 3 Algorithmic Approach

In this section we present a topology-shape-metrics approach for the automatic layout of partitioned diagrams. More precisely, we sketch the modifications necessary to include partitions into its planarization and orthogonalization phase.

### 3.1 Planarization

Our planarization strategy is based on the following three steps:

1. **Creating an initial partitioned drawing:**
   We use Sugiyama's approach to create an initial partitioned drawing. It is highly adaptable and especially suited for our purpose even if the input graph is undirected. The modifications for its different phases are quite simple:
   For the layering phase we insert $Y_{\max}$ dummy vertices $d_j^y$ into $G$ which represent the vertical grid coordinates of the partition grid graph $P_G$. These vertices are connected by directed edges $(d_j^y, d_{j+1}^y)$, $1 \leq j \leq Y_{\max} - 1$. For each vertex $v \in V$ with $p(v) = r$ we insert two directed edges $(v, d_{r^b}^y)$ and $(d_{r^t}^y, v)$. Thus, each feasible layering has the property that a vertex $v$ is placed between the top and bottom border of partition cell $p(v)$.
   The crossing minimization is modified such that the vertex order inside a layer is consistent with the fixed order of the corresponding partition cells.
   For the horizontal coordinate assignment we insert $X_{\max}$ dummy vertices $d_j^x$ into the compaction graph which represent the horizontal grid coordinates of $P_G$. These vertices are connected by directed edges $(d_j^x, d_{j+1}^x)$, $1 \leq j \leq X_{\max} - 1$. For each vertex $v$ with $p(v) = r$ we insert two directed edges $(v, d_{r^r}^x)$ and $(d_{r^l}^x, v)$ into the compaction graph to guarantee that $v$ is placed between the left and right border of partition cell $p(v)$.
2. **Construction of a p-planar embedding:**
   We "materialize" the partition $P_R$ by inserting the corresponding partition grid graph $P_G$ into the drawing constructed in the above step. Each vertex $v \in P_G$ is placed on its related coordinate $(d_i^x, d_j^y)$. To create a p-planar embedding, we detect crossings with a sweep-line algorithm and replace them by dummy vertices. This can be done in time $O(|S| \log |S| + c)$ where $c$ denotes the number of crossings and $S$ the set of segments. In our approach the number of segments is $|S| = O(|V| + |E|)$. Now, we determine the cyclic order of the edges around each vertex.
3. **Rerouting of edges:**
   The layered drawing produced by Sugiyama's approach is too restrictive because undirected edges can be routed non-monotonically. Thus, we perform a rerouting step to further reduce the number of crossings. The rerouting is based on shortest-path computations in the dual graph [2]. Since the size of the planarized graph is $O(|V| + c)$, the runtime of the rerouting is $O((|V| + c)|E|)$.

### 3.2 Orthogonalization

The orthogonalization phase has to preserve the shape of the partition grid graph $P_G$. The phase is divided into two steps:

1. First, we fix the angles between consecutive edges around each vertex of $P_G$. The angles have to comply with the partition structure.
2. In the second step, we calculate the shape of the edges. The fixed angles assigned in the first step are not allowed to change. Furthermore, the edge segments of $P_G$ are not allowed to bend. Hence, we assign high bend costs to those segments. The shape calculation of the edges is done with the network flow approach described in [5] applying the modifications of [1,3] that guarantee the conservation of prescribed angles and bends. We use a heuristic network solver that produces satisfying results in practice and has running time $O((|V|+c)^2 \log(|V|+c))$. There is always a valid drawing in which the fixed angles and shapes are observed [3].

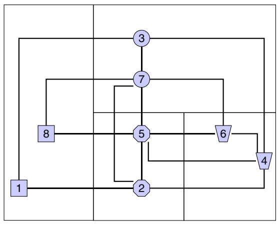
## 4 Discussion and Conclusion

First we discuss the running time of our new approach. We assume that the number of partition cells is $O(|V|)$. With the efficient implementation of Sugiyama's algorithm described in [4], we can create the initial partitioned drawing in time $O((|V| + |E|) \log |E|)$. The calculation of the planar embedding has time $O((|V| + |E|) \log(|V| + |E|) + c)$ and the rerouting $O((|V| + c)|E|)$ ($c$ denotes the number of crossings). Since the planarized graph has $|V| + c$ vertices the number of edges is $|E| = O(|V| + c)$. The orthogonalization step takes time $O((|V| + c)^2 \log(|V| + c))$. For the compaction we use the fast constructive algorithm described in [3] with a flow-based post-processing step and quadratic runtime. We sum up with the following theorem:

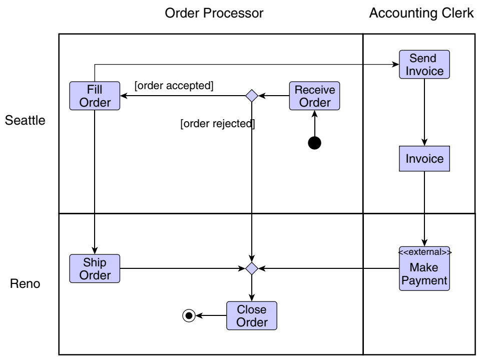
**Theorem 2.** *Given an undirected graph $G = (V, E)$, a rectangular partition $P_R$ of the drawing area and a mapping $p$ of the vertices to partition cells. Our approach creates a partitioned drawing of $G$ in time $O((|V| + c)^2 \log(|V| + c))$ where $c$ denotes the number of crossings in the planarized graph.*

We implemented our approach in Java using the yFiles library [10]. Two example layouts are given in Figure 4. More examples can be found at `http://www-pr.informatik.uni-tuebingen.de/partitioned-drawings/`. For diagrams with about 100 vertices and 150 edges we typically need a runtime of less than three seconds on a 3 GHz Pentium IV System with 1 GB RAM.

In this work we introduced the problem of finding a partitioned drawing of a graph $G = (V, E)$. We presented a heuristic approach that produces pleasing results in time $O((|V| + c)^2 \log(|V| + c))$. The approach can easily be extended to include edge labels or edges that should be drawn upward [9].



(a)



(b)

**Fig. 4.** Examples drawn with our new approach: (a) shows our layout for the example of Figure 3 and (b) the UML activity diagram of Figure 1(a)

## References

1. U. Brandes, M. Eiglsperger, M. Kaufmann, and D. Wagner. Sketch-driven orthogonal graph drawing. In *Proceedings of the 10th International Symposium on Graph Drawing (GD'02)*, volume 2528 of *LNCS*, pages 1–12. Springer, 2002.
2. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
3. M. Eiglsperger. *Automatic Layout of UML Class Diagrams: A Topology-Shape-Metrics Approach*. PhD thesis, Universität Tübingen, 2003.
4. M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An efficient implementation of Sugiyama's algorithm for layered graph drawing. In *Proceedings of the 12th Symposium on Graph Drawing (GD'04)*, volume 3383 of *LNCS*, pages 155–166. Springer, 2005.
5. U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In *Proceedings of the 3rd International Symposium on Graph Drawing (GD'95)*, volume 1027 of *LNCS*, pages 254–266. Springer, 1996.
6. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4:312–316, 1983.
7. J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.
8. J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. In *Proceedings of the 6th Symposium on Graph Drawing (GD'98)*, volume 1547 of *LNCS*, pages 263–274. Springer, 1998.
9. M. Siebenhaller and M. Kaufmann. Mixed upward planarization - fast and robust. In *Proceedings of the 13th Symposium on Graph Drawing (GD'05)*, volume 3843 of *LNCS*, pages 522–523. Springer, 2006.
10. yWorks. yFiles - a java graph layout and visualization library (WWW document). `http://www.yworks.com` (accessed September 2006).

# Path Simplification for Metro Map Layout

Damian Merrick[1,2] and Joachim Gudmundsson[2]

[1] School of Information Technologies, University of Sydney, Australia
dmerrick@it.usyd.edu.au
[2] National ICT Australia*, Sydney, Australia
joachim.gudmundsson@nicta.com.au

**Abstract.** We investigate the problem of creating simplified representations of polygonal paths. Specifically, we look at a path simplification problem in which line segments of a simplification are required to conform with a restricted set of directions $\mathcal{C}$. An algorithm is given to compute such simplified paths in $\mathcal{O}(|\mathcal{C}|^3 n^2)$ time, where $n$ is the number of vertices in the original path. This result is extended to produce an algorithm for graphs induced by multiple intersecting paths. The algorithm is applied to construct schematised representations of real world railway networks, in the style of metro maps.

## 1 Introduction

Metro maps have been used to effectively illustrate transportation networks for many decades. They incorporate a carefully balanced trade-off between geographical accuracy and readability of the diagram. Traditionally, these diagrams are drawn manually, and it is a significant challenge to create computer algorithms that produce high-quality metro maps. A common feature of metro maps is the logical division of the network into a set of intersecting paths, or train lines. One may consider the simpler problem of drawing these paths individually, instead of the entire network at once.

In the field of cartography, and particularly in geographical information systems (GIS), it is an important problem to represent detailed geographical features on a map in a simple, easily understandable form. Consequently, efficient algorithms for simplifying lines or paths in a geographical data set are desirable.

The problem of computing a simplification of a given polygonal path where the vertices of the simplification are required to be a subset of the input points has been studied extensively. Imai and Iri [19–21] formulated the problem as a graph problem. They constructed an unweighted directed acyclic graph and then used breadth-first search to compute a shortest path in this graph. The same approach has been used by many algorithms devoted to this problem [3, 4, 6, 7]. A widely used heuristic for path-simplification is the Douglas-Peucker algorithm [10]. If the path is given in the plane then it can be implemented to run in $\mathcal{O}(n \log^* n)$ time [17], but it does not guarantee an optimal solution.

* National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

Numerous criteria have been proposed for simplifying polygonal paths. In [4, 8, 19, 21, 23] the so-called *tolerance zone* criterion was used. Other measurements are the *infinite beam* criterion [7, 11, 29], the *uniform measure* criterion [2, 13], *distance preserving* criterion [15] and the *area preserving* criterion [3].

In this paper we address a related problem. As input, a polygonal *path* $P$ is given, consisting of a sequence of points $\langle p_1, \ldots, p_n \rangle$, with closed line segments called *links* joining each pair of consecutive points. Also given is a finite set of directions $\mathcal{C}$. The problem we address is to produce a simplification of $P$, subject to some constraints, where every link $l_i$ in the simplification is parallel to some orientation $c \in \mathcal{C}$. A path conforming to this restriction is called a $\mathcal{C}$-directed path. Requiring a $\mathcal{C}$-directed path as output results in a "schematised" approximation of the original path. Such an approximation can greatly improve the readability of diagrams in which several paths must be drawn. The vertices of the simplification are not restricted to be a subset of the input points.

In the case when the links are restricted to a given set of orientations Neyer [26] proposed an algorithm that minimises the number of links in the output path. This algorithm runs in time $\mathcal{O}(nk^2 \log n)$, where $k$ is the number of links in the output. The output is a path which remains within a given distance of the original path, according to the Fréchet distance metric. Utilising Fréchet distance in this respect, however, can produce undesirable "zig-zags" in some paths when the set of allowed orientations is small.

The problem considered in this paper uses a more relaxed restriction on distance. For each point $p$ in the input path $P$, consider its *$\varepsilon$-circle* $E(p)$; that is, the closed disc of radius $\varepsilon$ centred at $p$. Our requirement is that a simplification must intersect $E(p)$ for every $p \in P$, subject to a restriction on the order of intersections. This is equivalent to enforcing a maximal Haussdorf distance of the path simplification from the set of input points. Compared to the Fréchet metric, more freedom is given to the simplification at bends.

Guibas et al. [16] considered a version of this problem where the links are not restricted to a certain set of directions. They presented a dynamic programming algorithm that generates an optimal solution in $\mathcal{O}(n^2 \log^2 n)$ time, and a 2-approximation with running time $\mathcal{O}(n \log n)$. In this paper we show that the restricted direction path simplification problem can be solved in $\mathcal{O}(|\mathcal{C}|^3 n^2)$ time.

In addition to the simplification of individual lines, we enter into the problem of simplifying multiple intersecting paths in a network. This is of particular interest in metro map layout. Hong et al. [18] present some force-directed graph drawing approaches to metro map layout, which are also used to produce metro map layouts of non-geographical networks. Slower but more geographically accurate optimisation-based methods are detailed by Stott and Rodgers [28], and more recently by Nöllenburg and Wolff [27]. One of the main benefits of the approach we present in this paper is its fast running time. This gives the potential to handle much larger instances than those solvable by slower methods. In addition, faster methods bring about the possibility of real-time interactivity. For example, in a network design application or diagramming tool, a designer may want to modify a network and immediately visualise the effect on the diagram.

A second problem related to metro map layout is the schematisation of networks. Cabello et al. [5] give an $\mathcal{O}(n \log n)$ time algorithm which creates a schematised map of a given network, where intersection points remain in fixed locations and paths between intersection points are drawn using two or three links. This approach ensures that the topology of the network is preserved. Other recent work investigates problems in drawing schematised layouts of trees [14].

We restrict our attention to networks induced by multiple intersecting paths, as in metro maps. We propose an efficient method for simplifying such networks given a restricted set of directions and an error threshold. Preservation of topology is not guaranteed in this approach.

Section 2 formalises the path simplification problem we address, and proposes an algorithm to solve it. The extension of the path simplification algorithm to metro map layout is presented in Section 3, and implementation results are given in Section 3.1. Section 4 offers some concluding remarks and acknowledgements.

Due to space constraints most details are omitted and can be found in [25].

## 2 $\mathcal{C}$-Directed Path Simplification

The path simplification problem we address in this paper is defined as follows:

*Problem 1. The $\mathcal{C}$-Directed Path Simplification Problem*

*Input*: A path $P$, a set of directions $\mathcal{C}$ and a distance threshold $\varepsilon$.

*Output*: A ($\mathcal{C}$, $\varepsilon$)-simplification $P'$ of $P$ with the minimum possible number of links, such that the $\varepsilon$-circles of all points in $P$ are stabbed in order by $P'$.

To make the problem definition clear we also need to define "order".

**Definition 1.** *A directed line segment $\ell$ that intersects the $\varepsilon$-circles of a sequence of points $\mathcal{S} = \langle p_1, \ldots, p_n \rangle$ is said to follow the* order *of $\mathcal{S}$ if and only if for every pair of points $p_i, p_j \in \mathcal{S}, i < j$ there exist points $q_i, q_j$ on $\ell$ within the $\varepsilon$-circles of $p_i$ and $p_j$ respectively, for which it holds that $q_i$ is encountered before or at the same position as $q_j$ along $\ell$.*

To guarantee that a solution exists, we assume that for any direction $c \in \mathcal{C}$, the opposite direction $\bar{c}$ is also in $\mathcal{C}$.

The algorithm we propose to solve the $\mathcal{C}$-directed path simplification problem is composed of two parts. First, given a path to simplify, it constructs a boundary path (to be defined) which determines a set of minimum-link $(\mathcal{C}, \varepsilon)$-simplifications of the given path, see Fig. 1(a). In the second part, a single minimum-link $(\mathcal{C}, \varepsilon)$-simplification of the path is extracted from the boundary path, see Fig. 1(b). We start with some necessary definitions.

Given a pair of directions $(c_1, c_2)$ and a set of points $\mathcal{S} = \{p_1, \ldots, p_n\}$ let $\tau_i$ be the $c_1$-most tangent of the $\varepsilon$-circle of $p_i$ with direction $c_2$, and let $\alpha_i$ be the point (if any) on $\tau_i$ with the smallest $c_2$-coordinate for which it holds that a line through $\alpha_i$ with direction $c_1$ stabs $\{p_1, \ldots, p_i\}$ in order as shown in Fig. 2(a–b). Let $\iota$ be the smallest value for which there is no such point $\alpha_\iota$, and let $\mathcal{S}' = \{p_1, \ldots, p_{\iota-1}\}$. For every $1 \leq j < \iota$ let $\ell_j$ be the ray with direction $c_1$ emanating from $\alpha_j$.

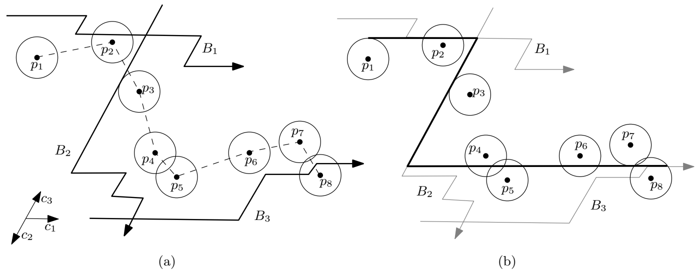


**Fig. 1.** (a) An example showing a sequence of eight $\varepsilon$-circles and three link boundaries $B_1, B_2$ and $B_3$ forming a boundary path. (b) Extracting a ($\mathcal{C}$, $\varepsilon$)-simplification from the boundary path.

**Definition 2.** *The* link boundary *$L$ of $\mathcal{S}$ with direction pair ($c_1$, $c_2$) is the polyline described by the upper envelope in direction $c_2$ of the set $\ell_1, \ldots, \ell_j$. The link boundary $L$ has* size *$j$ and is said to* stab *$\mathcal{S}'$ (in order).*

If the stabbing order is ignored, a single maximum link boundary $B$ can easily be computed in linear time with respect to the number of points stabbed by $B$. In Section 2.2 we discuss in more detail how to do it in the ordered case.

**Definition 3.** *A* boundary path *is a sequence of link boundaries that stabs a sequence of points $\mathcal{S}$. A size $k$ maximal boundary path of $\mathcal{S}$ is a boundary path containing $k$ link boundaries that stabs the maximal start sequence of $\mathcal{S}$.*

## 2.1 A High-Level Description

In this section we give an overview of the algorithm. The idea is to construct a boundary path $B$ where each link boundary, denoted $B_1, \ldots, B_k$, along $B$ corresponds to a segment in the final path simplification $P'$ of $P$, and then extract a path from $B$. The algorithm is given as pseudocode below.

**Building the boundary path.** The boundary path is built by incrementally adding a link boundary. Let $\beta_k(c_i, c_j)$ denote a maximal boundary path of size $k$ whose last link boundary has direction pair $(c_i, c_j)$. In each iteration of the algorithm, $\mathcal{O}(|\mathcal{C}|^2)$ active candidate boundary paths are maintained together with a counter $k$ of the number of iterations, i.e., the size of all the boundary paths. Initially, $\beta_k(c_i, c_j)$ is the maximal single link boundary path of $\mathcal{S}$ with direction pair $(c_i, c_j)$, and $k = 1$. If any of the $\mathcal{O}(|\mathcal{C}|^2)$ boundary paths stabs the entire sequence $\mathcal{S}$ then we are done, otherwise continue iteratively as follows.

For every triple of directions $(c_i, c_j, c_\ell)$ in $\mathcal{C}$ extend $\beta_k(c_i, c_j)$ with a link boundary of direction pair $(c_j, c_\ell)$, excluding cases where $c_i$ is parallel to $c_j$ or $c_j$ is parallel to $c_\ell$. Consider all the maximal boundary paths with $k+1$ links whose last link boundary has direction pair $(c_j, c_\ell)$, and save the boundary path that stabs the longest sequence of $\mathcal{S}$ in $\beta_{k+1}(c_j, c_\ell)$. Finally, increment the value of $k$. The process ends when a boundary path stabs $\mathcal{S}$. The first boundary path that stabs $\mathcal{S}$ is denoted $\Delta(\mathcal{S})$ and has size $k$.

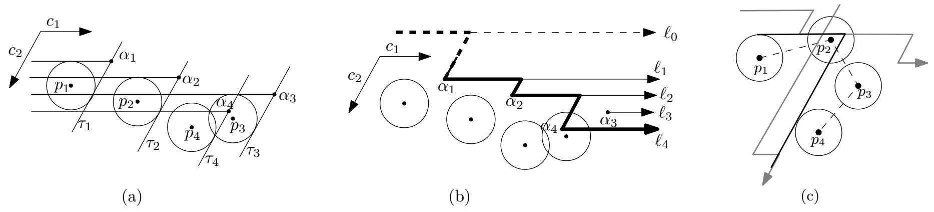


**Fig. 2.** (a) $\tau_i$ is the $c_1$-most tangent of the $\varepsilon$-circle of $p_i$ with direction $c_2$. (b) The link boundary with direction pair $(c_1, c_2)$ of $\langle p_1, \ldots, p_4 \rangle$. (c) A potential problem when two link boundaries are joined; the truncation of one of the links at the bends results in $p_3$ not being stabbed.

**Constructing a path from the boundary path.** Given the boundary path $\Delta(\mathcal{S})$ containing a sequence of link boundaries $\langle B_1, \ldots, B_k \rangle$, construct the $(\mathcal{C}, \varepsilon)$-simplification $P' = \langle p'_1, p'_2, \ldots, p'_{k+1} \rangle$ as follows. Let $p'_{k+1}$ be the point within $p_n$'s $\varepsilon$-circle that is furthest along $B_k$ in $B_k$'s primary direction. Let $\ell_k$ be the line through the last ray of $B_k$. Set $p'_k$ to the intersection of $\ell_k$ and $B_{k-1}$, and let $\ell_{k-1}$ be the line through $p'_k$ in the primary direction of $B_{k-1}$. Continue this process for every point $p'_i$ back to $p'_2$, setting $p'_i$ to the intersection of $\ell_i$ and $B_{i-1}$. Finally, let $p'_1$ be the point within $p_1$'s $\varepsilon$-circle lying on $B_1$ that is backmost in $B_1$'s primary direction.

```
Algorithm StabbingPath
1.  done:= false
2.  for $i = 1$ to $|\mathcal{C}|$ do
3.      for $j = 1$ to $|\mathcal{C}|$ do
4.          for $k = 1$ to $n$ do
5.              $\beta_k(c_i, c_j)$ :=null
6.  $k := 0$
7.  while not done do
8.      for $i = 1$ to $|\mathcal{C}|$ do
9.          for $j = 1$ to $|\mathcal{C}|$ do
10.             for $\ell = 1$ to $|\mathcal{C}|$ do
11.                 tmp:=ExpandBoundaryPath($\beta_k(c_i, c_j), c_\ell, \mathcal{S}$)
12.                 $\beta_{k+1}(c_j, c_\ell)$ :=SelectBestBoundaryPath($\beta_{k+1}(c_j, c_\ell)$,tmp)
13.                 if $\beta_{k+1}(c_j, c_\ell)$ stabs $\mathcal{S}$ then
14.                     $\Delta(\mathcal{S}) := \beta_{k+1}(c_j, c_\ell)$
15.                     done:=true
16.     $k := k + 1$
17. endwhile;
18. Return ComputePath($\Delta(\mathcal{S}), \mathcal{S}$)
```

Consider each of the steps of the algorithm. Steps 1–6 of the algorithm require $\mathcal{O}(|\mathcal{C}|^2 n)$ time and step 18 requires $\mathcal{O}(n)$ time. It remains to study the body within the three loops in steps 11–12. Step 12 can easily be performed in linear time since one only has to decide which of the two boundary paths that stabs

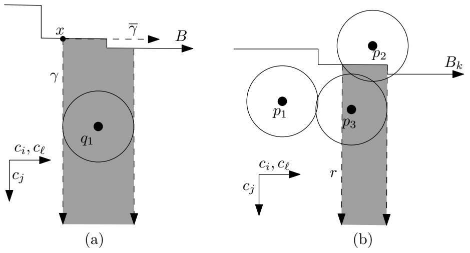


**Fig. 3.** (a) The boundary-restricted stabbing interval $I(\langle q_1\rangle, c_j, B)$ (shaded). (b) The boundary-restricting stabbing interval $I(\langle p_2, p_3\rangle, c_j, B_k)$ (shaded). The left bounding ray $r$ of this interval is the backmost ray starting on $B_k$ that ensures all points remain stabbed, and forms the first ray of $B_{k+1}$.

the most $\varepsilon$-circles. That leaves step 11, which is the step that will dominate the running time of the algorithm.

Guibas et al. [16] showed that one can compute a line that stabs the longest possible prefix of a sequence of unit discs in the correct order in $\mathcal{O}(n \log n)$ time. However, their problem has two main differences to ours. First, we only need to consider one direction. Second, when $B_{k+1}$ is joined to the end of the boundary path $\beta_k(c_i, c_j)$ then $B_k$ is truncated at its point of intersection with $B_{k+1}$. Hence we must ensure that after the truncation all points in $S_k$ are either still stabbed by $B_k$ or by $B_{k+1}$. Figure 1(c) illustrates this problem.

In the next section we will discuss how step ExpandBoundaryPath can be implemented to run in $\mathcal{O}(n^2)$ time.

## 2.2 Extending a Boundary Path with a Link Boundary

Consider the algorithm described in the previous section. Let $B_1, \ldots, B_k$ be the link boundaries in $\beta_k(c_i, c_j)$, and let $\mathcal{S}_k = \langle p_{f(k)}, \ldots, p_{l(k)}\rangle$ be the sequence of $\mathcal{S}$ stabbed by $B_k$. Thus the sequence of $\mathcal{S}$ that remains to be stabbed is $\mathcal{S}' = \langle p_{f(k+1)}, \ldots, p_n\rangle$, where $f(k+1) = l(k) + 1$.

We need to be very careful when joining two link boundaries. To facilitate this we define *boundary-restricted stabbing intervals* (Figs. 3(a), (b) give examples).

**Definition 4.** *Given a sequence of points $\mathcal{S}$, a direction $c$, and a link boundary $B$, the* boundary-restricted stabbing interval $I(\mathcal{S}, c, B)$ *is the region in the plane for which it holds that a $c$-directed ray starting on $B$ lies inside the region if and only if it stabs $\mathcal{S}$ in order.*

**Initialisation.** The task of the initialisation step is to construct the first ray in $B_{k+1}$. If the turn from $c_i$ to $c_j$ is in the same direction as the turn from $c_j$ to $c_\ell$, e.g. both are left hand turns, then we add to $B_{k+1}$ the $c_j$ directed ray that is at $\infty$ according to $c_i$. If the turns are in different directions, i.e. one is a left hand turn and one is a right hand turn, then we need to find the $c_j$-directed ray that starts as far back as possible on $B_k$ and ensures $S_k$ remains stabbed. Note that this ray starts as far back along $B_k$ as possible, while still ensuring that all points are stabbed by either $B_k$ or the ray itself. Hence, it forms the backmost bound for all possible rays in $B_{k+1}$. An example of this is illustrated in Fig. 3(b).

**Extending $B_{k+1}$ to Stab a New Point.** Once the initialisation has been completed, the main loop commences. At each iteration, the algorithm attempts to extend the set of points stabbed by $\beta_k(c_i, c_j)$ and the link boundary $B_{k+1}$.

Consider the points in $\mathcal{S}$ not stabbed by $\beta_k(c_i, c_j)$, denoted $\mathcal{S}' = \langle q_1, q_2, \ldots \rangle$. Process the points in $\mathcal{S}'$ in order, starting with $q_1$. In a generic step assume we are about to process $q_m$. Compute the stabbing interval $I(\langle q_1, \ldots, q_m \rangle, c_j, B_k)$. If the interval is empty then we stop since there is no $c_j$-oriented line that can stab $q_1, \ldots, q_m$ in order. If the interval is non-empty let $\gamma$ be the bounding line of the stabbing interval with the smallest $c_\ell$-value. Find the intersection point $x$ between $\gamma$ and $B_k$, and let $\overline{\gamma}$ be the $c_i$-directed ray starting at $x$. Process every point $p$ in $\mathcal{S}_k$ in reverse order as follows.

If $\overline{\gamma}$ cut the $\varepsilon$-circle of $p$ into two pieces then $B_k$ no longer stabs $p$, and it must be stabbed by $B_{k+1}$. Set $\gamma$ to be the bounding line of $I(\langle p, \ldots, q_m \rangle, c_i, B_k)$ with the smallest $c_\ell$-value. If the stabbing interval is empty then we are finished, as $B_{k+1}$ cannot stab $p, \ldots, q_m$ or any further points. Otherwise, continue iterating.

Once the backtracking has completed, all points in the sequence $\mathcal{S}_k$ and $\langle q_1, \ldots, q_m \rangle$ are stabbed by either $B_k$ or $\gamma$. Now consider the last ray $\gamma'$ in $B_{k+1}$. If $\gamma'$ is positioned at $\infty$, remove it from $B_{k+1}$ and replace it with the ray along $\gamma$ that starts on $B_k$. Otherwise, let $t$ be the $c_j$-most $c_\ell$-directed tangent to the $\varepsilon$-circle of $p_m$. Add $t$ and $\gamma$ to $B_{k+1}$, and truncate $\gamma'$, $t$ and $\gamma$ at their pairwise intersection points. The point $p_m$ is now stabbed by $B_{k+1}$, and can be removed from $\mathcal{S}'$ and added to $\mathcal{S}_k$. Iteration continues until $\mathcal{S}'$ is empty or a point is found that cannot be stabbed.

If each stabbing interval is computed from scratch, a straight-forward implementation would require $\mathcal{O}(n^2)$ time. This follows since the order restriction has to be checked for every pair of $\varepsilon$-discs in the sequence. However, since the above approach incrementally adds new points at the end of the sequence this can be improved as stated in Theorem 1 below (see [25] for details). The concluding result for this section follows in Theorem 2.

**Theorem 1.** *There is a data structure of size $\mathcal{O}(n)$ that answers boundary-restricted stabbing interval queries in constant time and allows additions of points to the end of the sequence in $\mathcal{O}(n)$ time.*

**Theorem 2.** ExpandBoundaryPath$(\beta_k(c_i, c_j), c_\ell, \mathcal{S})$ *can be computed in $\mathcal{O}((|\mathcal{S}_k| + |\mathcal{S}_{k+1}|)^2)$ time, where $\mathcal{S}_k$ and $\mathcal{S}_{k+1}$ are the sets of points stabbed by $B_k$ and $B_{k+1}$, respectively, in the produced boundary path.*

## 2.3 Complexity Analysis

**Time Complexity.** Recall from Section 2.1 that the running time of algorithm StabbingPath is dominated by step 11, i.e. ExpandBoundarypath. Assume that a minimum link path contains $k$ links, using Theorem 2 gives that the total running time is bounded by:

$$\mathcal{C}^3 \cdot \left( |\mathcal{S}_1| + \sum_{i=1}^{k-1} \left(|\mathcal{S}_i| + |\mathcal{S}_{i+1}|\right)^2 \right) < \mathcal{C}^3 \cdot 6 \left( \sum_{i=1}^{k-1} |\mathcal{S}_i|^2 + \sum_{i=1}^{k-1} |\mathcal{S}_{i+1}|^2 \right) = \mathcal{O}\left(\mathcal{C}^3 n^2\right).$$

The final step of extracting a minimum-link $(\mathcal{C}, \varepsilon)$-simplification from the boundary path takes $\mathcal{O}(n)$ time, therefore the entire algorithm needs $\mathcal{O}(|\mathcal{C}|^3 n^2)$ time.

**Space Complexity.** At each iteration of the algorithm, $\mathcal{O}(|\mathcal{C}|^2)$ paths are stored, each containing up to $k$ boundaries. Each link boundary uses linear space and every input point is stabbed by at most a constant number of link boundaries per boundary path, thus it follows that the algorithm requires $\mathcal{O}(|\mathcal{C}|^2 n)$ space.

### 2.4 Proof of Correctness

**Lemma 1.** *Given a starting point $q$, a preceding link boundary $B_{i-1}$ and directions $c_{i-1}$, $c_i$ and $c_{i+1}$, the pair of link boundaries $(B_i, B_{i+1})$ constructed by the algorithm stabs the furthest point of $P$ possible.*

**Theorem 3.** *Algorithm* StabbingPath$(P, \varepsilon)$ *computes a minimum link* $(\mathcal{C}, \varepsilon)$*-simplification of $P$.*

Lemma 1 and Theorem 3 are proved in [25].

## 3 Extension to Metro Map Layout

A major motivation for this paper is the automatic visualisation of metro maps. For this purpose, we extend the $\mathcal{C}$-directed path simplification algorithm of Section 2 to handle multiple intersecting paths.

We adopt the graph-based model of Hong et al. [18] to describe a metro network; a *metro map graph* consists of a graph $G$ and a set of paths covering all vertices and edges of $G$. We assume we are given a metro map graph with a set of initial coordinates for each vertex, representing the geographical locations of stations in the metro network.

The first step taken by the algorithm is to sort the set of paths according to a given measure of *importance*. Importance may be manually defined, or calculated automatically by some heuristic function. For our purposes, we define the importance of a path to be the number of vertices on the path that are also a part of some other path in the metro map graph.

Once the paths are sorted, the most important path is taken from the set and simplified using the algorithm of Section 2. Once the path simplification has been computed, the points of the path must be placed on the simplification. This may be achieved by simply projecting each original point onto the link in the path simplification that stabs it. Alternatively, the points may be redistributed along the line segment such that the distance between every adjacent pair of points is equal. This may result in a distance greater than $\varepsilon$ between each original point and its counterpart on the simplification. However, it is useful for reducing clutter and making the final diagram clearer. Once placed, the points in the simplification are fixed in their positions.

The next most important path is then taken and split into subpaths around any sequences of fixed vertices, such that each subpath contains at most one fixed vertex at either end. Each of these subpaths is simplified in turn, and all of the points in the resulting simplifications are fixed.

A small modification must be made to the path simplification algorithm to ensure that the simplified path runs exactly through any fixed points. This is achieved by restricting the allowed interval for each fixed point to the single line in each direction that passes exactly through that point.

The algorithm repeats the above steps, each time taking the next most important path from the set, splitting it and simplifying, until there are no paths left to be simplified. A complete layout has then been generated.

We found that two extensions to the path simplification algorithm were useful to improve the results: restricting the maximum turning angle at bends, and enforcing a minimum link length.

**Maximum bend angle.** Allowing turns of large angles between links in a simplification can result in undesirable sharp bends. Given an angle $\alpha$, one may want to ensure that no turn of more than $\alpha$ degrees is made in the simplified path. To implement this, simply ignore any pairs of consecutive directions separated by more than $\alpha$ degrees. Theorem 3 still holds in this case, i.e. the simplification produced will have the minimum number of links of any $(\mathcal{C}, \varepsilon)$-simplification subject to the same restriction.

**Minimum link length.** The algorithm enforces no restriction on the length of links in the simplification. This can cause a restriction on the maximum bend angle to be less effective, as the extra links produced may be of zero length. We enforce a minimum length $l_{min}$ for each link when constructing a link boundary, by trimming all boundary-restricted stabbing intervals to a distance of $l_{min}$ from the start of the previous link boundary. Problems can occur where a link needs to be shorter than $l_{min}$ in order to stab a point. To avoid this, we decrease $l_{min}$ on any iteration in which no further points are stabbed by any link boundary. If this continues to occur for several iterations, we set $l_{min}$ to zero. This approach ensures that a solution will be computed, but the number of links in the solution may not be the minimum.

## 3.1 Results

We applied the method of Section 3 to the central part of the Sydney Cityrail railway network [9], and the London Underground network [22]. The Sydney graph has 173 vertices and 182 edges, and London has 266 vertices and 308 edges. We created a Java implementation as a plugin to the graph editor *jjGraph* [12].

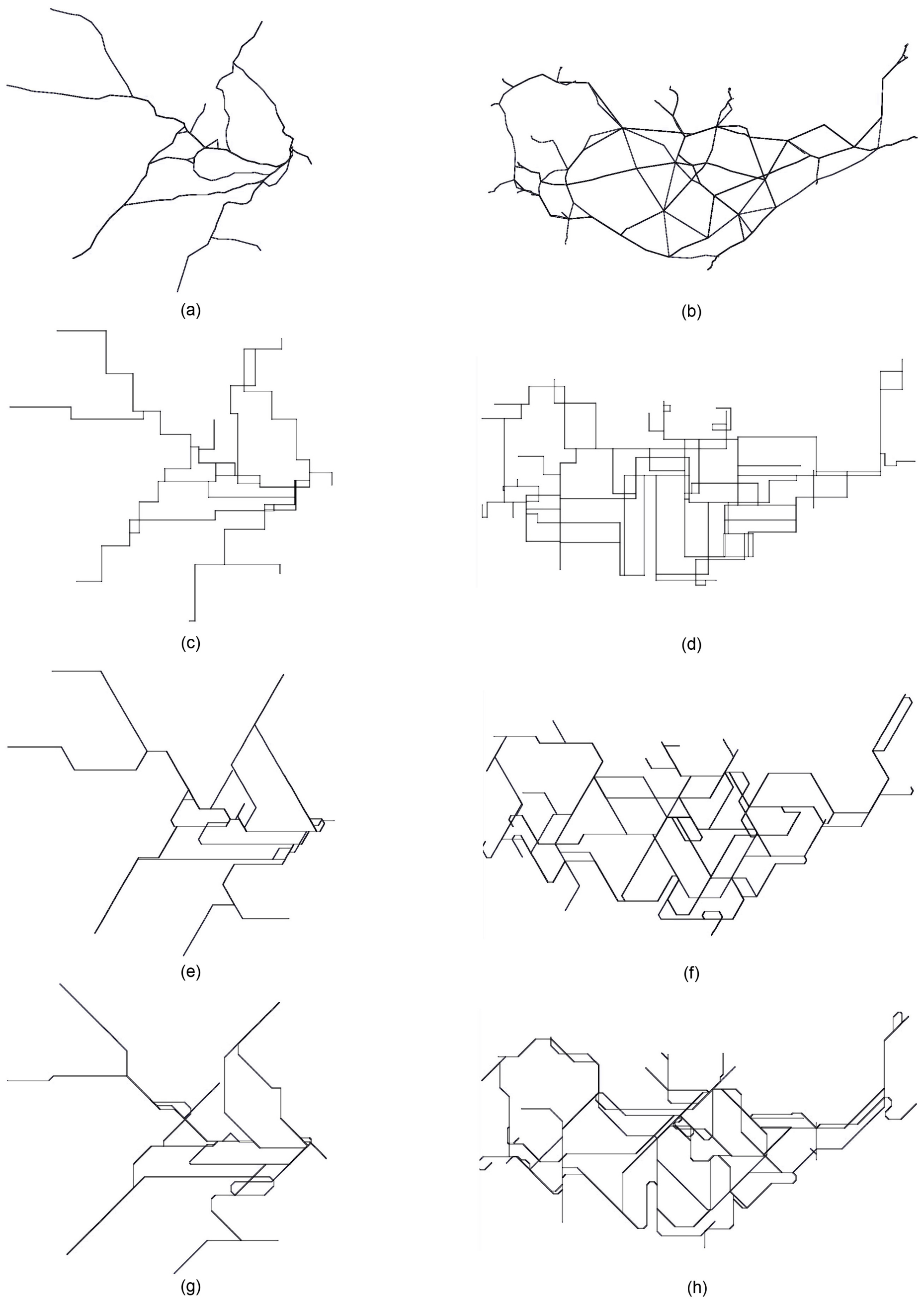


**Fig. 4.** (a) The original Cityrail geometry, and simplifications using (c) 4, (e) 6 and (g) 8 directions. (b) The original London geometry, and simplifications using (d) 4, (f) 6 and (h) 8 directions.

The input geometry for the Cityrail network was taken directly from its geographical layout, shown in Fig. 4(a). The original geography of the London network is quite dense in the centre and sparse in the exterior, so we scaled it using the centrality-based scaling technique of Merrick and Gudmundsson [24] before applying our algorithm. The scaled geometry is shown in Fig. 4(b). The scaling used betweenness centrality, with parameters $\alpha = 10, \beta = 50$ and $\gamma = 2$, and the scaling took 6445 ms to execute. Refer to the literature for details on the scaling method and its associated parameters [24].

We ran our implementation on both networks with varying parameters, and show selected results. Figs. 4(c), (e) and (g) show the Cityrail network simplified with $|\mathcal{C}| = 4$, 6 and 8 respectively. The respective running times were 154, 201 and 268 ms. Figs. 4(d), (f) and (h) show the London network simplified with $|\mathcal{C}| = 4$, 6 and 8 respectively. The running times to obtain these results were 282, 423 and 573 ms. All results were produced on a computer with a Pentium 4 3.0 GHz processor and 1GB of RAM, running Windows XP. Running times were averaged over 10 runs. Values of $\varepsilon$ were chosen by trial and error and varied between the datasets; at this stage the possibility of automatically choosing an appropriate value for $\varepsilon$ remains as future work.

## 4 Concluding Remarks and Acknowledgements

Consider a $k$-link $(\mathcal{C}, \varepsilon)$-simplification of a path $P$ produced by the algorithm of Section 2. It might be that there exists a $k$-link $(\mathcal{C}, \varepsilon')$-simplification of $P$ where $\varepsilon'$ is much smaller than $\varepsilon$. In [25] we present a fully polynomial-time approximation scheme (FPTAS) to the dual of the problem, i.e. given $k$ compute a $(\mathcal{C}, \varepsilon)$-simplification of $P$ that minimises $\varepsilon$. The FPTAS produces a $(1 + \delta)$-approximation for any given $\delta > 0$.

We thank Martin Nöllenburg and Alexander Wolff for the London data.

## References

1. H. Alt and M. Godau. Measuring the resemblance of polygonal curves. In Proc. 8th Annual Symposium on Computational Geometry, p. 102–109, 1992.
2. P. K. Agarwal and K. R. Varadarajan. Efficient Algorithms for Approximating Polygonal Chains. Discrete & Computational Geometry, 23(2):273–291, 2000.
3. J. Bose, O. Cheong, S. Cabello, J. Gudmundsson, M. van Kreveld and B. Speckmann. Area-preserving approximations of polygonal paths. J. Discrete Alg., 2006.
4. G. Barequet, D. Z. Chen, O. Daescu, M. T. Goodrich and J. Snoeyink. Efficiently approximating polygonal paths in three and higher dimensions. Algorithmica, 33(2):150–167, 2002.
5. S. Cabello, M. de Berg, and M. van Kreveld. Schematization of networks. Computational Geometry and Applications, 30:223–238, 2005.
6. W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. IJCGA, 6:59–77, 1996.
7. D. Z. Chen and O. Daescu. Space-efficient algorithms for approximating polygonal curves in two-dimensional space. IJCGA, 13:95–111, 2003.

8. D. Z. Chen, O. Daescu, J. Hershberger, P. M. Kogge, N. Mi and J. Snoeyink. Polygonal path simplification with angle constraints. CGTA, 32(3):173–187, 2005.
9. CityRail network map. Web page: http://www.cityrail.info/networkmaps/mainmap.jsp (Accessed 6th Sept 2006).
10. D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer 10(2):112–122, 1973.
11. D. Eu and G. T. Toussaint. On Approximating Polygonal Curves in Two and Three Dimensions. CVGIP: Graphical Model and Image Processing, 56(3):231–246, 1994.
12. C. Friedrich. jjGraph. Personal communication.
13. M. T. Goodrich. Efficient piecewise-linear function approximation using the uniform metric. Discrete & Computational Geometry, 14:445–462, 1995.
14. J. Gudmundsson, M. van Kreveld and D. Merrick. Schematisation of Tree Drawings. Submitted to Graph Drawing, June 2006.
15. J. Gudmundsson, G. Narasimhan and M. H. M. Smid. Distance-Preserving Approximations of Polygonal Paths. To appear in CGTA, 2006.
16. L. J. Guibas, J. Hershberger, J. S. B. Mitchell and J. Snoeyink. Approximating Polygons and Subdivisions with Minimum Link Paths. IJCGA, 3(4):383–415, 1993.
17. J. Hershberger and J. Snoeyink. Cartographic line simplification and polygon CSG formulæ in $O(n \log^* n)$ time. Computational Geometry – Theory & Applications, 11(3-4):175–185, 1998.
18. S.-H. Hong, D. Merrick, and H. A. D. do Nascimento. The metro map layout problem. In Proc. Graph Drawing 2004, p. 482–491, 2005.
19. H. Imai and M. Iri. Computational-geometric methods for polygonal approximations of a curve. Comp. Vision, Graphics and Image Processing, 36:31–41, 1986.
20. H. Imai and M. Iri. An optimal algorithm for approximating a piecewise linear function. Journal of Information Processing, 9(3):159–162, 1986.
21. H. Imai and M. Iri. Polygonal approximations of a curve-formulations and algorithms. In Computational Morphology, G. T. Toussaint (ed.), North-Holland, Amsterdam, 1988.
22. London Underground network map. Web page: http://www.tfl.gov.uk/tube/maps/ (Accessed 6th Sept 2006).
23. A. Melkman and J. O'Rourke. On polygonal chain approximation. In Computational Morphology, G. T. Toussaint (ed.), North-Holland, Amsterdam, 1988.
24. D. Merrick and J. Gudmundsson. Increasing the readability of graph drawings with centrality-based scaling. In Proc. APVIS 2006, p. 67–76, 2006.
25. D. Merrick and J. Gudmundsson. $\mathcal{C}$-Directed Path Simplification for Metro Map Layout. http://www.dmist.net/metromap.pdf (Accessed 6th Sept 2006).
26. G. Neyer. Line simplification with restricted orientations. In Proc. 6th International Workshop on Algorithms and Data Structures, p. 13–24, 1999.
27. M. Nöllenburg and A. Wolff. A mixed-integer program for drawing high-quality metro maps. In Proc. 13th International Symposium on Graph Drawing, 2005.
28. J. Stott and P. Rodgers. Metro map layout using multicriteria optimization. In Proc. 8th Interational Conference on Information Visualisation, p. 355–362, 2004.
29. G. T. Toussaint. On the Complexity of Approximating Polygonal Curves in the Plane. In Proc. of the International Symposium on Robotics and Automation (IASTED), pages 311–318, 1985.

# Minimizing Intra-edge Crossings in Wiring Diagrams and Public Transportation Maps

Marc Benkert[1,⋆], Martin Nöllenburg[1,⋆], Takeaki Uno[2], and Alexander Wolff[1,⋆]

[1] Department of Computer Science, Karlsruhe University, Germany
http://i11www.iti.uka.de/algo/group
[2] National Institute of Informatics, Tokyo, Japan
uno@nii.jp

**Abstract.** In this paper we consider a new problem that occurs when drawing wiring diagrams or public transportation networks. Given an embedded graph $G = (V, E)$ (e.g., the streets served by a bus network) and a set $L$ of paths in $G$ (e.g., the bus lines), we want to draw the paths along the edges of $G$ such that they cross each other as few times as possible. For esthetic reasons we insist that the relative order of the paths that traverse a node does not change within the area occupied by that node.

Our main contribution is an algorithm that minimizes the number of crossings on a single edge $\{u, v\} \in E$ if we are given the order of the incoming and outgoing paths. The difficulty is deciding the order of the paths that terminate in $u$ or $v$ with respect to the fixed order of the paths that do not end there. Our algorithm uses dynamic programming and takes $O(n^2)$ time, where $n$ is the number of terminating paths.

## 1 Introduction

In wiring diagrams or public transportation networks many paths must be drawn on the same underlying graph, see Figures 1 and 2. In order to make the resulting layout as understandable as possible it is desirable to (a) avoid crossings wherever possible and (b) insist that the relative order of the lines that traverse a node does not change in that node. For example, note that the subway line 5, which passes under the main station of Cologne (Köln Hbf), crosses lines 16–19 south of the station in the clipping of the public transport map of Cologne in Figure 2. The crossing is not hidden under the rectangle that represents the station. This makes it easier to follow the subway lines visually.

We model the problem as follows. We assume that we are given an undirected connected graph $G = (V, E)$ together with an embedding in the plane. The graph represents the underlying structure of the wiring or the road/tracks in the case of a transportation network. We are also given a set $L$ of *lines* in $G$. They represent the cables in a wiring diagram or the lines in a transportation network. A line $\ell \in L$ is an edge sequence $e_1 = \{v_0, v_1\}, e_2 = \{v_1, v_2\}, \ldots, e_k = \{v_{k-1}, v_k\} \in E$

⋆ Supported by grant WO 758/4-2 of the German Research Foundation (DFG).

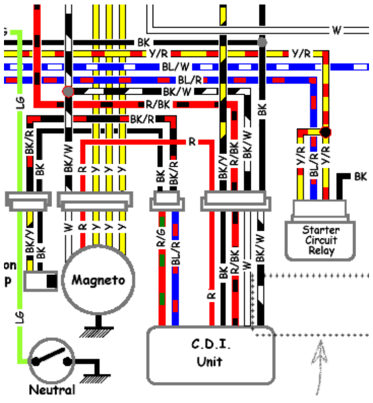


**Fig. 1.** Clipping of a wiring diagram

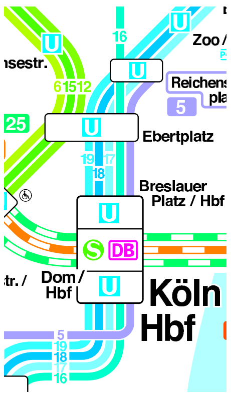


**Fig. 2.** Clipping of the public transport network of Cologne

that forms a simple path in $G$. The stations $v_0$ and $v_k$ are the *terminal stations* for line $\ell$ while $v_1, \ldots, v_{k-1}$ are *intermediate stations*. Our aim is to draw all lines in $L$ such that the number of crossings among pairs of lines in $L$ is minimized.

To get a flavor of the problem observe that the structure of $G$ enforces certain crossings, see Figure 3(a): lines $\ell_1$ and $\ell_2$ use exactly the path $\langle u, w_1, w_2, v\rangle$ together. The graph structure (indicated by the first and last line segments of each line) enforces that $\ell_1$ enters station $u$ above $\ell_2$ while it leaves $v$ below $\ell_2$, thus $\ell_1$ and $\ell_2$ have to cross somewhere between $u$ and $v$. However, fixing the location of the crossing of $\ell_1$ and $\ell_2$ determines crossings with other lines that have a terminal stop in $w_1$ or $w_2$. If there is a line $\ell$ that enters $u$ between $\ell_1$ and $\ell_2$ and terminates at $w_2$ (see Figure 3(b)), the crossing between $\ell_1$ and $\ell_2$ should be placed between $w_2$ and $v$. Now suppose there is another line $\ell'$ that enters $v$ between $\ell_2$ and $\ell_1$ from the right and terminates at $w_2$, see Figure 3(c). Then at least one of the lines $\ell$ and $\ell'$ intersects one of the lines $\ell_1$ or $\ell_2$, no matter where the crossing between $\ell_1$ and $\ell_2$ is placed.

We can abstract from geometry as follows. Given an edge $\{u, v\}$ in $G$ and the orders of the lines in $u$ and $v$ that traverse $u$ and $v$, respectively, we ask

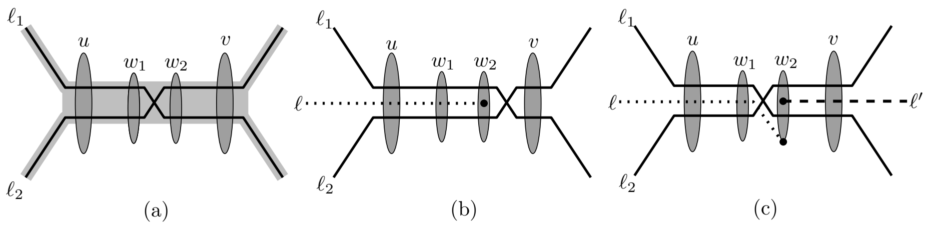


**Fig. 3.** Different placements of the necessary intersection between lines $\ell_1$ and $\ell_2$ on the path $u, w_1, w_2, v$. In (c) at least one of the lines $\ell$ and $\ell'$ has to intersect one of the lines $\ell_1$ or $\ell_2$.

for the order of *all* lines that enter $\{u, v\}$ in $u$ and for the order in which *all* lines leave $\{u, v\}$ in $v$. Then the number of crossings on $\{u, v\}$ is the number of transpositions needed to convert one order into the other. The difficulty is deciding the order of the lines that terminate in $u$ or $v$ with respect to the fixed order of the lines that traverse both $u$ and $v$. This is the *one-edge layout* problem that we study in Section 2.

In contrast to the well-known NP-hard problem of minimizing crossings in a two-layer bipartite graph [3] the one-edge layout problem is polynomially solvable. The main reason is that there is an optimal layout of the lines that pass through edge $\{u, v\}$ such that no two lines that terminate in $u$ intersect and no two lines that terminate in $v$ intersect. This observation allows us to split the problem and to apply dynamic programming. It is then rather easy to come up with an $O(n^5)$-time solution and with some effort we could reduce the running time to $O(n^2)$, where $n$ is the number of lines that do not terminate in $u$ or $v$.

A solution of the general line layout problem, i.e., a simultaneous crossing-minimal solution for all edges of a graph, would be of interest as a second (and mostly independent) step for drawing bus or metro maps, a topic that has received some attention lately, see the work of Nöllenburg and Wolff [4] and the references therein. However, in that direction of research the focus has so far been exclusively on drawing the underlying graph nicely, and not on how to embed the bus or metro lines along the network. We give some hints in Section 3 why already the *two-edge layout* problem seems to be substantially harder than the one-edge layout.

A vaguely similar problem has been considered by Cortese et al. [1]. Given the drawing of a planar graph $G$ they widen the edges and vertices of the drawing and ask if a given combinatorial cycle in $G$ has a plane embedding in the widened drawing.

## 2 A Dynamic Program for One-Edge Layout

In this section we consider the following special case of the problem.

*Problem 1. One-edge layout*
We are given a graph $G = (V, E)$ and an edge $e = \{u, v\} \in E$. Let $L_e$ be the set of lines that use $e$. We split $L_e$ into three subgroups: $L_{uv}$ is the set of lines that pass through $u$ and $v$, i.e., neither $u$ or $v$ is a terminal station. $L_u$ is the set of lines that pass through $u$ and for which $v$ is a terminal station and $L_v$ is the set of lines that pass through $v$ and for which $u$ is a terminal station. We assume that there are no lines that exclusively use the edge $\{u, v\}$ as they could be placed top- or bottommost without causing any intersections. Furthermore we assume that the lines for which $u$ is an intermediate station, i.e., $L_{uv} \cup L_u$, enter $u$ in a predefined order $S_u$. Analogously, we assume that the lines for which $v$ is an intermediate station, i.e., $L_{uv} \cup L_v$, enter $v$ in a predefined order $S_v$. The task is to find a layout of the lines in $L_e$ such that the number of pairs of intersecting lines is minimized.

Note that the number of crossings is determined by inserting the lines in $L_u$ into the order $S_v$ and by inserting the lines in $L_v$ into $S_u$. The task is to find insertion orders that minimize the number of crossings. Observe that the orders $S_u$ and $S_v$ themselves already determine the number of crossings between pairs of lines in $L_{uv}$ and that the insertion orders of $L_u$ in $S_v$ and of $L_v$ in $S_u$ do not change this number. Thus, we will not take crossings between lines in $L_{uv}$ into account anymore. On the other hand, fixing an insertion order affects the number of crossings between lines in $L_u \cup L_v$ and $L_{uv}$ and the number of crossings between lines in $L_u$ and $L_v$ in a non-trivial way. Figure 4 shows that a line $\ell_u \in L_u$ can indeed cross a line $\ell_v \in L_v$ in the unique optimal solution. Throughout the paper lines in $L_{uv}$ are drawn solid, lines in $L_u$ dotted and lines in $L_v$ dashed. We will now show that no two lines in $L_u$ (and analogously in $L_v$) cross in an optimal solution. This nice property is the key for solving the one-edge layout in polynomial time.

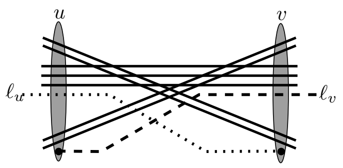


**Fig. 4.** The lines in $L_{uv}$ are drawn solid. In an optimal solution $\ell_u \in L_u$ and $\ell_v \in L_v$ intersect.

**Lemma 1.** *In any optimal solution for the one-edge layout problem no pair of lines in $L_u$ and no pair of lines in $L_v$ intersects.*

*Proof.* Assume to the contrary that there is an optimal solution $\sigma$ with a pair of lines in $L_u$ that intersects. Among all such pairs in $\sigma$ let $\{\ell, \ell'\}$ be the one whose intersection point $p$ is rightmost. W.l.o.g. $\ell$ is above $\ell'$ in $u$. Let $\ell_p$ and $\ell'_p$ be the parts of $\ell$ and $\ell'$ to the right of $p$, see Figure 5(a). Since $\sigma$ is crossing minimal, the courses of $\ell_p$ and $\ell'_p$ intersect the minimum number of lines in $L_{uv} \cup L_v$ in order to get from $p$ to $v$. In particular, the number of crossings between $\ell_p$ and lines of $L_{uv} \cup L_v$ and between $\ell'_p$ and lines of $L_{uv} \cup L_v$ must be the same otherwise we could place $\ell_p$ parallel to $\ell'_p$ (or vice versa) which would reduce the number of crossings. However, since the number of crossings to the right is the same we can easily get rid of the crossing between $\ell$ and $\ell'$ by replacing $\ell_p$ by a copy of $\ell'_p$ infinitesimally close above $\ell'_p$, see Figure 5(b). The proof for $L_v$ is analogous. $\square$

From now on we assume that no two lines of $L_u$ are consecutive in $S_u$ and analogously no two lines of $L_v$ are consecutive in $S_v$. The reason for this assumption is that a set of consecutive lines can simply be drawn parallelly in an optimal layout. Thus a single line suffices to determine the optimal course for the whole bundle. Technically, we can deal with this case by merging a bundle of $k$ consecutive lines of $L_u$ or $L_v$ to one line and assigning a weight of $k$ to it. The dynamic program will then run in a weighted fashion that counts $k \cdot k'$ crossings for a crossing of two lines with weights $k$ and $k'$. For simplification we only explain the unweighted version of the problem in detail.

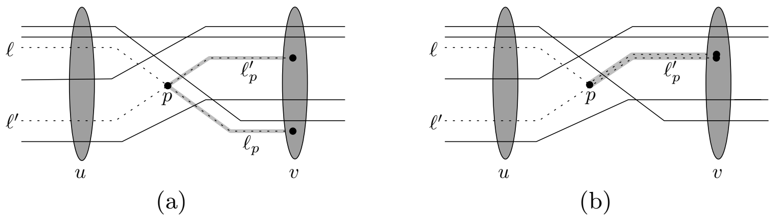


**Fig. 5.** Two lines of $L_u$ do not intersect in an optimal solution

Let $n$, $n_u$ and $n_v$ be the number of lines in $L_{uv}, L_u$ and $L_v$, respectively. Note that by the above assumption $n_u, n_v \leq n+1$ holds.

Recall that by assumption the lines in $L_{uv} \cup L_u$ enter $u$ in a predefined order and the lines in $L_{uv} \cup L_v$ leave $v$ in a predefined order. Let $S_u = (s_1^u < \cdots < s_{n+n_u}^u)$ be the bottom-up order of lines in $L_{uv} \cup L_u$ in $u$ and $S_v = (s_1^v < \cdots < s_{n+n_v}^v)$ be the bottom-up order of lines in $L_{uv} \cup L_v$ in $v$. A line $\ell$ in $L_v$ can terminate below $s_1^u$, between two neighboring lines $s_i^u$, $s_{i+1}^u$, or above $s_{n+n_u}^u$. We denote the position of $\ell$ by the index of the lower line and by 0 if it is below $s_1^u$. Let $S_v|L_v = (s_{\pi(1)}^v < \ldots < s_{\pi(n_v)}^v)$ denote the order on $L_v$ induced by $S_v$ and let $S_u|L_u = (s_{\mu(1)}^u < \ldots < s_{\mu(n_u)}^u)$ denote the order on $L_u$ induced by $S_u$. Here, $\pi$ and $\mu$ are injective functions that filter the lines $L_v$ out of all ordered lines $L_{uv} \cup L_v$ in $S_v$ and the lines $L_u$ out of all ordered lines $L_{uv} \cup L_u$ in $S_u$, see Figure 6.

*Preprocessing.* The orders $S_u$ and $S_v$ already determine the number of necessary crossings between pairs of lines in $L_{uv}$. Let $\mathrm{cr}_{uv}$ denote this number. Since $\mathrm{cr}_{uv}$ is fixed there is no need to consider the corresponding crossings in the minimization. We will now fix the course of a line in $L_v$. This line, say $\ell = s_{\pi(j)}^v$, has index $\pi(j)$ in $S_v$, and we fix the course of $\ell$ by choosing its terminal position $i$ in the order $S_u$. We denote the number of crossings between $\ell$ and all lines in $L_{uv}$ by $\mathrm{cr}_v(i,j)$. This number is determined as follows. The line $\ell$ crosses a line $\ell' \in L_{uv}$ with left index $i'$ and right index $j'$ if and only if either it holds that $i' \leq i$ and $j' > \pi(j)$ or it holds that $i' > i$ and $j' < \pi(j)$. The table $\mathrm{cr}_v$ for all lines in $L_v$ has $(n + n_u + 1) \times n_v = O(n^2)$ entries. For fixed $i$ we can compute the row $\mathrm{cr}_v(i, \cdot)$ as follows. We start with $j = 1$ and compute the number of lines in $L_{uv}$ that intersect line $\ell$ with indices $i$ and $\pi(j)$. Then, we increment $j$ and obtain $\mathrm{cr}_v(i, j+1)$ by $\mathrm{cr}_v(i,j)$ minus the number of lines in $L_{uv}$ that are no longer intersected plus the lines that are newly intersected. As any of the $n$ lines in $L_{uv}$ receives this status 'no longer' or 'newly' at most once and this status can easily be checked by looking at $S_v$, this takes $O(n)$ time per row. Thus, in total we can compute the matrix $\mathrm{cr}_v$ in $O(n^2)$ time.

We define $\mathrm{cr}_u(i,j)$ analogously to be the number of crossings of the lines in $L_{uv}$ with a line in $L_u$ that has index $\mu(i)$ in $S_u$ and position $j$ in $S_v$. Computing $\mathrm{cr}_u$ is analogous to $\mathrm{cr}_v$.

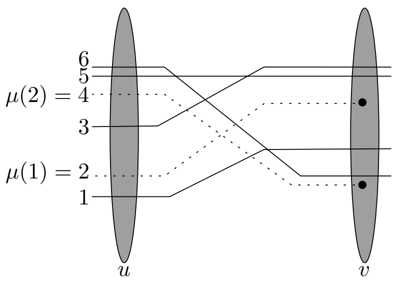


**Fig. 6.** The order $S_u$ and the induced suborder $S_u|L_u = (s_2^u < s_4^u)$

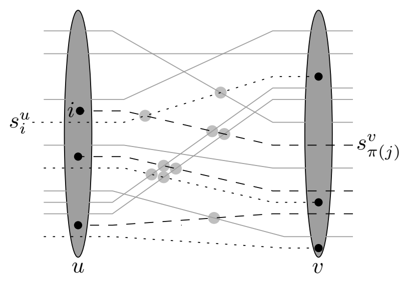


**Fig. 7.** Configuration corresponding to $F(i,j)$: line $s_{\pi(j)}^v$ terminates at position $i$ in $u$

*Dynamic Program.* Assume that we fix the destination of $s_{\pi(j)}^v$ to some $i \in \{0,\dots,n+n_u\}$. Then we define $F(i,j)$ as the minimum number of crossings of (a) the lines in $\{s_1^u,\dots,s_i^u\} \cap L_u$ with the lines in $L_{uv} \cup L_v$ and (b) the lines in $\{s_1^v,\dots,s_{\pi(j)}^v\} \cap L_v$ with the lines in $L_{uv} \cup L_u$. This situation is depicted in Figure 7, where only the crossings indicated by gray disks are counted in $F(i,j)$. Then the values $F(i,j)$ define an $(n+n_u+1) \times n_v$-matrix $F$.

Once the last column $F(\cdot,n_v)$ of the matrix $F$ is computed, i.e., all lines in $L_v$ are placed, we can determine the optimal solution for $L_e$ as

$$F^* := \min\{F(i,n_v) + C(i,n+n_u,n_v+1) \mid i = 0,\dots,n\},$$

where $C(i,n+n_u,n_v+1)$ is the remaining number of crossings of lines in $L_u \cap \{s_{i+1}^u,\dots,s_{n+n_u}^u\}$ with $L_{uv} \cup L_v$, which are not yet counted in $F(i,n_v)$.

Before turning to the recursive computation of $F(i,j)$ we introduce another notation. Let us assume that $s_{\pi(j-1)}^v$ terminates at position $k$ and $s_{\pi(j)}^v$ terminates at position $i$, where $0 \le k \le i \le n+n_u$ and $j \in \{1,\dots,n_v\}$. Then let $C(k,i,j)$ denote the minimum number of crossings that the lines $L_{k,i}^u := \{s_{k+1}^u,\dots,s_i^u\} \cap L_u$ cause with $L_{uv} \cup L_v$. In other words $C(k,i,j)$ counts the minimal number of crossings of all lines of $L_u$ in the interval defined by the endpoints of the two lines $s_{\pi(j-1)}^v$ and $s_{\pi(j)}^v$. This situation is illustrated in Figure 8, where those crossings that are marked with gray disks are counted in the term $C(k,i,j)$. The following theorem gives the recursion for $F$ and shows its correctness.

**Theorem 1.** *The values $F(i,j)$, $i = 0,\dots,n+n_u$, $j = 1,\dots,n_v$, can be computed recursively by*

$$F(i,j) = \begin{cases} \min_{k \le i}\{F(k,j-1) + C(k,i,j) + cr_v(i,j)\} & \text{if } i \ge 1, j \ge 2 \\ \sum_{l=1}^{j} cr_v(0,l) & \text{if } i = 0, j \ge 1 \\ C(0,i,1) + cr_v(i,1) & \text{if } i \ge 1, j = 1. \end{cases} \quad (1)$$

*Proof.* The base cases of Equation (1) consist of two parts. In the first row, an entry $F(0,j)$ means that all lines $s_{\pi(1)}^v,\dots,s_{\pi(j)}^v$ terminate at position 0 in $u$ and hence the required number of crossings is just the number of crossings of these lines with $L_{uv}$, which equals the sum given in Equation (1). In the first column, an entry $F(i,1)$ reflects the situation that line $s^v_{\pi(1)}$ terminates at position $i$. The required number of crossings in this case is simply $\mathrm{cr}_v(i,1)$, the number of crossings of $s^v_{\pi(1)}$ with $L_{uv}$, plus $C(0,i,1)$, the number of crossings of $L^u_{0,i}$ with $L_{uv} \cup L_v$.

The general case of Equation (1) means that the value $F(i,j)$ can be composed of the optimal placement $F(k,j-1)$ of the lines below and including $s^v_{\pi(j-1)}$ (which itself terminates at some position $k$ below $i$), the number $C(k,i,j)$ of crossings of lines in $L_u$ in the interval between $k$ and $i$, and the number of crossings $\mathrm{cr}_v(i,j)$ of $s^v_{\pi(j)}$ at position $i$.

Due to Lemma 1 we know that $s^v_{\pi(j)}$ cannot terminate below $s^v_{\pi(j-1)}$ in an optimal solution. Hence, for $s^v_{\pi(j)}$ terminating at position $i$, we know that $s^v_{\pi(j-1)}$ terminates at some position $k \le i$. For each $k$ we know by the induction hypothesis that $F(k,j-1)$ is the correct minimum number of crossings as defined above. In order to extend the configuration corresponding to $F(k,j-1)$ with the next line $s^v_{\pi(j)}$ in $L_v$ we need to add two terms: (a) the number of crossings of $L^u_{k,i}$ with $L_{uv} \cup L_v$, which is exactly $C(k,i,j)$, and (b) the number $\mathrm{cr}_v(i,j)$ of crossings that the line $s^v_{\pi(j)}$ (terminating at position $i$) has with $L_{uv}$. Note that potential crossings of $s^v_{\pi(j)}$ with lines in $L^u_{k,i}$ are already considered in the term $C(k,i,j)$. Figure 8 illustrates this recursion: $s^v_{\pi(j)}$ is placed at position $i$ in the order $S_u$, and $s^v_{\pi(j-1)}$ terminates at position $k$. The crossings of the configuration corresponding to $F(i,j)$ that are not counted in $F(k,j-1)$, are the $C(k,i,j)$ crossings of the marked, dotted lines of $L_u$ (indicated by gray disks) and the $\mathrm{cr}_v(i,j)$ encircled ones of $s^v_{\pi(j)}$ with $L_{uv}$.

Finally, we have to show that taking the minimum value of the sum in Equation (1) for all possible terminal positions $k$ of line $s^v_{\pi(j-1)}$ yields an optimal solution for $F(i,j)$. Assume to the contrary that there is a better solution $F'(i,j)$. This solution induces a solution $F'(k,j-1)$, where $k$ is the position of $s^v_{\pi(j-1)}$ in $S_u$. Lemma 1 restricts $k \le i$ and hence $s^v_{\pi(j-1)}$ runs completely below $s^v_{\pi(j)}$. Therefore we have $F'(k,j-1) \le F'(i,j) - C(k,i,j) - \mathrm{cr}_v(i,j) < F(i,j) - C(k,i,j) - \mathrm{cr}_v(i,j) \le F(k,j-1)$. This contradicts the minimality of $F(k,j-1)$. □

If we store in each cell $F(i,j)$ a pointer to the corresponding predecessor cell $F(k,j-1)$ that minimizes Equation (1) we can reconstruct the optimal edge layout: starting at the cell $F(i,n_v)$ that minimizes $F^*$, we can reconstruct the genesis of the optimal solution using backtracking. Obviously, using the combinatorial solution to place all endpoints of $L_e$ in the correct order and then connecting them with straight-line segments results in a layout that has exactly $F^*$ crossings in addition to $\mathrm{cr}_{uv}$, the invariable number of crossings of $L_{uv}$.

Now, we can give a first, naive approach: As mentioned earlier the tables $\mathrm{cr}_u$ and $\mathrm{cr}_v$ can be computed in $O(n^2)$ time. For the computation of one cell entry $C(k,i,j)$ we only have to look at the at most $n$ lines $L^u_{k,i}$ and their possible $n+1$

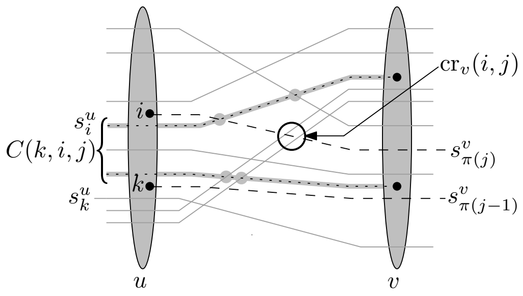


**Fig. 8.** The recursion for $F(i,j)$: lines $s_{\pi(j-1)}^v$ and $s_{\pi(j)}^v$ terminate at pos. $k$ and $i$, resp.
$C(k,i,j)$ = min. # crossings of the marked dotted lines $L_{k,i}^u$ with $L_{uv} \cup L_v$ $=^{\text{here}}$ 4
$\mathrm{cr}_v(i,j)$ = number of crossings of $s_{\pi(j)}^v$ with $L_{uv}$ $=^{\text{here}}$ 2

terminal positions in $v$. Once we have fixed a terminal position of a line $\ell \in L_{k,i}^u$, we have to compute the number of crossings that $\ell$ has with $L_{uv} \cup L_v$. For the crossings with $L_{uv}$ we simply have to look at the corresponding value of $\mathrm{cr}_u$. For the crossings with $L_v$ it is sufficient to look at the index of the terminal position because we know that position $k$ is the terminal position of $s_{\pi(j-1)}^v$ and position $i$ is the terminal position of $s_{\pi(j)}^v$. Thus, computing one of the $O(n^3)$ cells of the table $C$ requires $O(n^2)$ time, so in total we need $O(n^5)$ time for filling $C$. This dominates the computation of the table $F$. In the remainder of this section we show how to speed up the computation of $F$ and $C$.

*Improving the Running Time.* Let us—for the moment—assume that the values $C(k,i,j)$ and $\mathrm{cr}_v(i,j)$ are available in constant time. Then the computation of the $(n + n_u + 1) \times n_v$ matrix $F$ still needs $O(n^3)$ time because the minimum in Equation (1) is over a set of $O(n)$ elements. The following series of lemmas shows how we could bring the running time down to $O(n^2)$. First we show a relation for the entries of the matrix $C$.

**Lemma 2.** *$C(k,i,j)$ is additive in the sense that $C(k,i,j) = C(k,l,j)+C(l,i,j)$ for $k \le l \le i$.*

*Proof.* Since $C(k,i,j)$ denotes the number of crossings of the lines in $L_u \cap \{s_{k+1}^u, \ldots, s_i^u\}$ and no two of these lines intersect each other (recall Lemma 1) we can split the layout corresponding to $C(k,i,j)$ at any position $l$, $k \le l \le i$ and get two (possibly non-optimal) configurations for the induced subproblems. This implies $C(k,i,j) \ge C(k,l,j) + C(l,i,j)$.

Conversely, we can get a configuration for $C(k,i,j)$ by putting together the optimal solutions of the subproblems. W.l.o.g. this introduces no additional crossings (they could be removed as in the proof of Lemma 1). Hence we have $C(k,i,j) \le C(k,l,j) + C(l,i,j)$. □

Now we show that we do not need to compute all entries of $C$.

**Lemma 3.** *Given the matrix $C$, the matrix $F$ can be computed in $O(n^2)$ time.*

*Proof.* Having computed entry $F(i-1,j)$ we can compute $F(i,j)$ in constant time as follows:

$$F(i,j) = \min \begin{cases} F(i-1,j) + C(i-1,i,j) - \mathrm{cr}_v(i-1,j) + \mathrm{cr}_v(i,j), \\ F(i,j-1) + \mathrm{cr}_v(i,j). \end{cases} \tag{2}$$

The correctness follows from Equation (1), Lemma 2, and the fact that $C(i,i,j)$ vanishes:

$$\begin{aligned} F(i,j) &\overset{(1)}{=} \min \begin{cases} \min\limits_{k<i}\{F(k,j-1) + C(k,i,j) + \mathrm{cr}_v(i,j)\}, \\ F(i,j-1) + C(i,i,j) + \mathrm{cr}_v(i,j) \end{cases} \\ &\overset{\text{L. 2}}{=} \min \begin{cases} \min\limits_{k\leq i-1}\{F(k,j-1) + C(k,i-1,j) + C(i-1,i,j) + \mathrm{cr}_v(i,j)\}, \\ F(i,j-1) + \mathrm{cr}_v(i,j) \end{cases} \\ &\overset{(1)}{=} \min \begin{cases} F(i-1,j) - \mathrm{cr}_v(i-1,j) + C(i-1,i,j) + \mathrm{cr}_v(i,j), \\ F(i,j-1) + \mathrm{cr}_v(i,j) \end{cases} \end{aligned}$$

In the first column, we can reformulate the recursion for $i \geq 1$ as follows:

$$\begin{aligned} F(i,1) &\overset{(1)}{=} C(0,i,1) + \mathrm{cr}_v(i,1) \\ &\overset{\text{Lemma 2}}{=} C(0,i-1,1) + C(i-1,i,1) + \mathrm{cr}_v(i,1) \\ &\overset{(1)}{=} F(i-1,1) - \mathrm{cr}_v(i-1,1) + C(i-1,i,1) + \mathrm{cr}_v(i,1) \end{aligned}$$

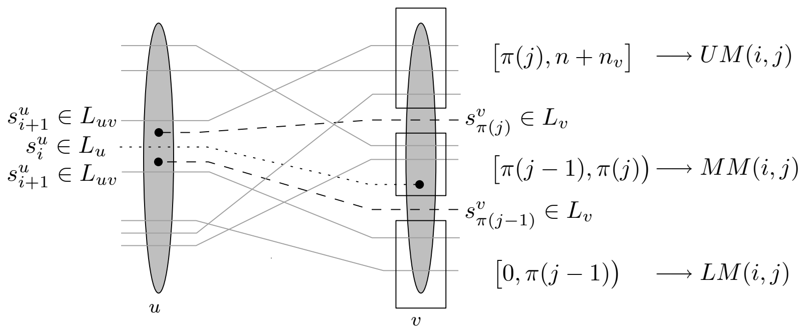
Hence the whole matrix $F$ can be computed in $O([n+n_u]\cdot n_v) = O(n^2)$ time. □

Observe that due to the reformulation in Lemma 3 we only need the values $C(i-1,i,j)$ explicitly in order to compute $F$. Now we will show that we can compute these relevant values in $O(n^2)$ time. For simplification we introduce the following notation: $C'(i,j) := C(i-1,i,j)$.

**Lemma 4.** *The values $C'(i,j)$ $(i = 1,\dots,n+n_u,\ j = 1,\dots,n_v)$ can be computed in $O(n^2)$ time.*

*Proof.* We compute $C'(i,j)$ row-wise, i.e., we fix $i$ and increase $j$. Recall that $C(k,i,j)$ was defined as the minimal number of crossings of the lines in $L^u_{k,i} = \{s^u_{k+1},\dots,s^u_i\} \cap L_u$ with the lines in $L_{uv} \cup L_v$ under the condition that $s^v_{\pi(j-1)}$ ends at position $k$ and $s^v_{\pi(j)}$ ends at position $i$. For $C'(i,j) = C(i-1,i,j)$ this means we have to consider crossings of the set $L^u_{i-1,i} = \{s^u_i\} \cap L_u$. Hence, we distinguish two cases: either $s^u_i$ is a line in $L_{uv}$ and then $L^u_{i-1,i}$ is empty or $s^u_i \in L_u$ and we have to place the line $s^u_i$ optimally. Clearly, in the first case we have $C'(i,j) = 0$ for all $j$ as there is no line to place in $L^u_{i-1,i}$.

Now we consider the case that $s^u_i \in L_u$. For each $j$ we split the set of candidate terminal positions for $s^u_i$ into the intervals $[0,\pi(j-1))$, $[\pi(j-1),\pi(j))$, and $[\pi(j), n+n_v]$. Let $\mathrm{LM}(i,j)$, $\mathrm{MM}(i,j)$, and $\mathrm{UM}(i,j)$ denote the minimum number of crossings of $s^u_i$ with $L_{uv} \cup L_v$ for terminal positions in $[0,\pi(j-1))$, $[\pi(j-1),\pi(j))$, and $[\pi(j), n+n_v]$, respectively. Now we have $C'(i,j) =$



$s^u_{i+1} \in L_{uv}$
$s^u_i \in L_u$
$s^u_{i+1} \in L_{uv}$
$u$
$v$
$[\pi(j), n+n_v] \longrightarrow UM(i,j)$
$s^v_{\pi(j)} \in L_v$
$[\pi(j-1), \pi(j)) \longrightarrow MM(i,j)$
$s^v_{\pi(j-1)} \in L_v$
$[0, \pi(j-1)) \longrightarrow LM(i,j)$

**Fig. 9.** Splitting the candidate terminal positions for $s^u_i$ into three intervals w.r.t. $\pi(j-1)$ and $\pi(j)$

$\min\{\text{LM}(i,j), \text{MM}(i,j), \text{UM}(i,j)\}$ as the minimum of the three distinct cases. The situation is illustrated in Figure 9.

Next, we have to show how to compute MM, LM, and UM. First, we consider MM. Recall that $\text{cr}_u(i,j)$ was defined as the number of crossings of the lines in $L_{uv}$ with the line $s^u_{\mu(i)}$ that terminates at position $j$ in $v$. It follows that

$$\text{MM}(i,j) = \min\{\text{cr}_u(\mu^{-1}(i), k) \mid \pi(j-1) \leq k < \pi(j)\}, \tag{3}$$

where $\pi(0)$ is defined as 0. Because of Lemma 1 there are no lines of $L_v$ intersecting the tunnel between $s^v_{\pi(j-1)}$ and $s^v_{\pi(j)}$. Hence, if the line $s^u_i$ terminates at position $k \in [\pi(j-1), \pi(j))$ it does not cross any line of $L_v$ and Equation (3) is correct. We can calculate $\text{MM}(i,j)$ by a straight-forward minimum computation through $j = 1, \ldots, n_v$ which takes $O(n + n_v) = O(n)$ time for each value of $i$.

Secondly, we consider LM. Initially, in the case that $j = 1$ there is no line $s^v_{\pi(j-1)}$ and the corresponding interval is empty. Hence we set $\text{LM}(i,1) = \infty$. Then we recursively compute

$$\text{LM}(i,j+1) = \min\{\text{LM}(i,j) + 1, \text{MM}(i,j) + 1\}. \tag{4}$$

Observe that for $\text{LM}(i,j+1)$ we merge the previous intervals corresponding to $\text{LM}(i,j)$ and $\text{MM}(i,j)$. Moreover the line $s^v_{\pi(j)}$, which previously ended at position $i$, now terminates at position $i-1$. Hence, in order to reach its terminal position in the interval $[0, \pi(j))$, the line $s^u_i$ has to cross $s^v_{\pi(j)}$ in addition to the crossings counted before by $\text{MM}(i,j)$ and $\text{LM}(i,j)$. This explains the recursion in Equation (4). The computation again requires $O(n)$ time for each value of $i$.

Finally, we initialize $\text{UM}(i,n_v) = 1 + \min\{\text{cr}_u(\mu^{-1}(i), k) \mid \pi(n_v) \leq k \leq n+n_v\}$ as for $j = n_v$ the line $s^u_i$ crosses the line $s^v_{\pi(n_v)}$ but no other line of $L_v$. In decreasing order we compute

$$\text{UM}(i,j-1) = \min\{\text{UM}(i,j) + 1, \text{MM}(i,j) + 1\} \tag{5}$$

analogously to LM, which again requires $O(n)$ time. As the whole procedure needs linear time for each $i = 1, \ldots, n + n_u$ the total running time is $O(n^2)$. ☐

Putting the intermediate results in Lemmas 3 and 4 together, we conclude:

**Theorem 2.** *The one-edge layout problem can be solved in $O(n^2)$ time.*

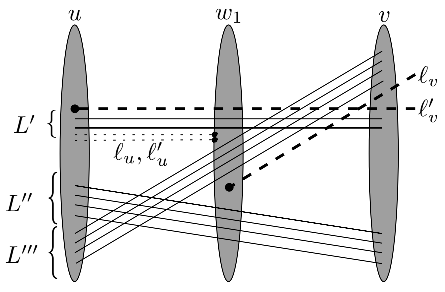
So far, the algorithm requires $O(n^2)$ space to store the tables $F, C', \mathrm{cr}_v$, and $\mathrm{cr}_u$. If we are only interested in the minimum *number* of crossings this can easily be reduced to $O(n)$ space as all tables can be computed row-wise: in $F$ we need only two consecutive rows at a time and we can discard previous rows; in the other tables the rows are independent and can be computed on demand. This does not affect the time complexity. However, to restore the optimal placement we need the pointers in $F$ to do backtracking and hence we cannot easily discard rows of the matrix. But we can still reduce the required space to $O(n)$ with a method similar to a divide-and-conquer version of the Needleman-Wunsch algorithm for biological sequence alignment [2] adding a factor of 2 to the time complexity. Basically, the idea is to keep only the pointers in one column of the matrix to reconstruct the optimal position of the corresponding line. This line cuts the problem into two smaller subproblems which are solved recursively.

## 3 Generalization to a Path

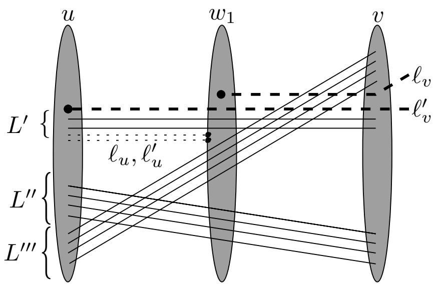
Surely it is desirable to draw lines on a more general fraction of the graph than only on a single edge. However, the problem seems to become significantly harder even for two edges. Let us first define the problem on a path.

*Problem 2. Path layout*
Given a graph $G = (V, E)$ and a simple path $P = \langle u, w_1, \ldots, w_m, v\rangle$ in $G$. Let $L_P$ be the set of lines that use at least one edge in $P$. We split $L_P$ into three subgroups: $L_{uv}^P$ is the set of lines that have no terminal station in $\{u, w_1, \ldots, w_m, v\}$, $L_u^P$ is the set of lines for which $u$ is an intermediate station and that have a terminal station in $\{w_1, \ldots, w_m, v\}$, and $L_v^P$ is the set of lines for which $v$ is an intermediate station and that have a terminal station in $\{u, w_1, \ldots, w_m\}$. We assume that there are no lines having both terminal stations in $\{u, w_1, \ldots, v\}$ as these could be placed top- or bottommost causing the minimum number of crossings with lines in $L_{uv}^P$. We also assume that any two lines $\ell_1$, $\ell_2$ that use exactly the subpath $x, \ldots, y \subseteq P$ together and for which neither $x$ nor $y$ is a terminal station enter $P$ in $x$ in a predefined order and leave $P$ in $y$ in a predefined order. The task is to find a layout of the lines in $L_P$ such that the number of pairs of intersecting lines is minimized.

We tried to apply the same dynamic-programming approach as for the one-edge case. However, the dilemma is that the generalized version of Lemma 1 does not hold, namely that no two lines in $L_v^P$ intersect. Thus, the problem instance cannot be separated into two independent subproblems, which seems to forbid dynamic programming. In Figure 10(a) we give an instance where two lines of $L_v^P$ cross in the optimal solution. Here, the lines in $L_{uv}^P$ are drawn solid. Recall that we do not have to take the intersections of lines in $L_{uv}^P$ into account as



(a) Optimal solution (7 crossings).



(b) Manipulating $\ell_v$ (8 crossings).

**Fig. 10.** The two lines $\ell_v$,$\ell'_v \in L^P_v$ cross in the optimal solution

these are again given by the orders in $S_u$ and $S_v$. The two lines $\ell_u$ and $\ell'_u \subseteq L^P_u$ are only needed to force the bundle crossings between the bundles $L' \subset L^P_{uv}$ and $L''' \subset L^P_{uv}$ to be on the edge $\{w_1, v\}$. In the optimal solution the lines $\ell_v, \ell'_v \in L^P_v$ intersect causing a total number of 7 crossings between lines in $L^P_v$ with lines in $L^P_v \cup L^P_{uv}$. We have to argue that any solution in which $\ell_v$ and $\ell'_v$ do not cross produces more than 7 crossings. We look at the optimal solution and argue that getting rid of the crossing between $\ell_v$ and $\ell'_v$ by manipulating the course of either $\ell_v$ or $\ell'_v$ produces at least 8 crossings. First, we consider manipulating the course of $\ell_v$, see Figure 10(b). However then, as $\ell_v$ has to be above $\ell'_v$, it has to cross the 4 lines in the bundle $L'''$ resulting in a total number of 8 crossings. Similarly, manipulating the course of $\ell'_v$ would also result in at least 8 crossings.

## 4 Concluding Remarks

Clearly our work is only a first step in exploring the layout of lines in graphs. What is the complexity of the problem if two edges of the underlying graph are considered, what about longer paths, trees and finally, general plane graphs? A variant of the problem where lines must terminate bottom- or topmost in their terminal stations is also interesting. This requirement prevents gaps in the course of continuing lines.

## References

1. P. F. Cortese, G. D. Battista, M. Patrignani, and M. Pizzonia. On embedding a cycle in a plane graph. In P. Healy and N. S. Nikolov, editors, *Proc. 13th Int. Symp. Graph Drawing (GD'05)*, volume 3843 of *LNCS*, pages 49–60. Springer-Verlag, 2006.
2. R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis.* Cambridge University Press, 1998.
3. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM J. Alg. Disc. Meth.*, 4:312–316, 1983.
4. M. Nöllenburg and A. Wolff. A mixed-integer program for drawing high-quality metro maps. In P. Healy and N. S. Nikolov, editors, *Proc. 13th Int. Symp. Graph Drawing (GD'05)*, volume 3843 of *LNCS*, pages 321–333. Springer-Verlag, 2006.

# Upright-Quad Drawing of *st*-Planar Learning Spaces

David Eppstein

Computer Science Department, University of California, Irvine
eppstein@uci.edu

**Abstract.** We consider graph drawing algorithms for learning spaces, a type of *st*-oriented partial cube derived from antimatroids and used to model states of knowledge of students. We show how to draw any *st*-planar learning space so all internal faces are convex quadrilaterals with the bottom side horizontal and the left side vertical, with one minimal and one maximal vertex. Conversely, every such drawing represents an *st*-planar learning space. We also describe connections between these graphs and arrangements of translates of a quadrant.

## 1 Introduction

A *partial cube* is a graph that can be given the geometric structure of a hypercube, by assigning the vertices bitvector labels in such a way that the graph distance between any pair of vertices equals the Hamming distance of their labels. Partial cubes can be used to describe benzenoid systems in chemistry [12], weak or partial orderings modeling voter preferences in multi-candidate elections [11], integer partitions in number theory [8], and the hyperplane arrangements familiar to computational geometers [6, 14]. In previous work we found algorithms for drawing arbitrary partial cubes, as well as partial cubes that have drawings as planar graphs with symmetric faces [5].

Here we consider graph drawing algorithms for *learning spaces* (also called *knowledge spaces*), a type of partial cube derived used to model states of knowledge of students [4]. These graphs can be large; Doignon and Falmagne [4] write "the number of knowledge states obtained for a domain containing 50 questions in high school mathematics ranged from about 900 to a few thousand." Thus, it is important to have efficient drawing techniques that can take advantage of the special properties of these graphs.

Our goal in graph drawing algorithms for special graph families is to combine the standard graph drawing aesthetic criteria of vertex separation, area, etc., with a drawing style from which the specific graph structure we are interested in is visible. Ideally, the drawing should be of a type that exists only for the graph family we are concerned with, so that membership in that family may be verified by visual inspection of the drawing. For instance, in our previous work [5], the existence of a planar drawing in which all faces are symmetric implies that the graph of the drawing is a partial cube, although not all partial cubes have such drawings. Another result of this type is our proof [9] that the graphs having delta-confluent drawings are exactly the distance-hereditary graphs.

The learning spaces considered in this paper are directed acyclic graphs with a single source and a single sink. It is natural, then, to consider *st*-planar learning spaces, those for which there exists a planar embedding with the source and sink on the same face. As we show, such graphs can be characterized by drawings of a very specific type: Every *st*-planar learning space has a dominance drawing in which all internal faces are convex quadrilaterals with the bottom side horizontal and the left side vertical. We call such a drawing an *upright-quad drawing*, and we describe linear time algorithms for finding an upright-quad drawing of any *st*-planar learning space. Conversely, every upright-quad drawing comes from an *st*-planar learning space in this way.

## 2 Learning Spaces

Doignon and Falmagne [4] consider sets of concepts that a student of an academic discipline might learn, and define a *learning space* to be a family $\mathcal{F}$ of sets modeling the possible states of knowledge that a student could have. Some concepts may be learnable only after certain prerequisites have been learned, so $\mathcal{F}$ may not be a power set. However, there may be more than one way of learning a concept, and therefore more than one set of prerequisites the knowledge of which allows a concept to be learned. We formalize these intuitive concepts mathematically with the following axioms:

**[L1]** If $S \in \mathcal{F}$ and $S \neq \emptyset$, then there exists $x \in S$ such that $S \setminus \{x\} \in \mathcal{F}$. That is, any state of knowledge can be reached by learning one concept at a time.
**[L2]** If $S$, $S \cup \{x\}$, and $S \cup \{y\}$ belong to $\mathcal{F}$, then $S \cup \{x,y\} \in \mathcal{F}$. That is, learning one concept cannot interfere with the ability to learn a different concept.

These axioms characterize families $\mathcal{F}$ that form *antimatroids* [13, Lemma III.1.2]. We define a *learning space* to be a graph having one vertex for each set in an antimatroid $\mathcal{F}$, and with a directed edge from each set $S \in \mathcal{F}$ to each set $S \cup \{x\} \in \mathcal{F}$. If $U = \bigcup \mathcal{F}$, we say that it is a learning space *over* $U$. Antimatroids also arise in other contexts than learning; e.g., the family of intersections of a point set in $\mathbb{R}^d$ with complements of convex bodies forms an antimatroid. In the remainder of this section we outline some standard antimatroid theory needed for the rest of our results.

**Lemma 1.** *If $\mathcal{F}$ satisfies axioms L1 and L2, and $K \subset L$ are two sets in $\mathcal{F}$, with $|L \setminus K| = n$, then there is a chain of sets $K_0 = K \subset K_1 \subset \cdots \subset K_n = L$, all belonging to $\mathcal{F}$, such that $K_i = K_{i-1} \cup \{q_i\}$ for some $q_i$.*

*Proof.* We use induction on $|K| + |L|$. If $K$ is empty, let $x$ be given by axiom L1 for $S = L$, and combine $q_n = x$ with the chain formed by induction for $K$ and $L \setminus \{x\}$. Otherwise, let $x$ be as given by axiom L1 for $S = K$, and form by induction a chain from $K \setminus \{x\}$ to $L$. By repeatedly applying axiom L2 we may add $x$ to each member of this chain not already containing it, forming a chain with one fewer step from $K$ to $L$. □

Similar repetitive applications of axiom L2 to the chain resulting from Lemma 1 proves the following:

**Lemma 2.** *If $\mathcal{F}$ satisfies axioms L1 and L2, and $K \subset L$ are two sets in $\mathcal{F}$, with $K \cup \{q\} \in \mathcal{F}$ and $q \notin L$, then $L \cup \{q\} \in \mathcal{F}$.*

**Lemma 3 (Cosyn and Usun [1]).** *Let $\mathcal{F}$ satisfy the conclusions of Lemmas 1 and 2. Then the union of any two members of $\mathcal{F}$ also belongs to $\mathcal{F}$, and $\mathcal{F}$ is* well-graded*;*

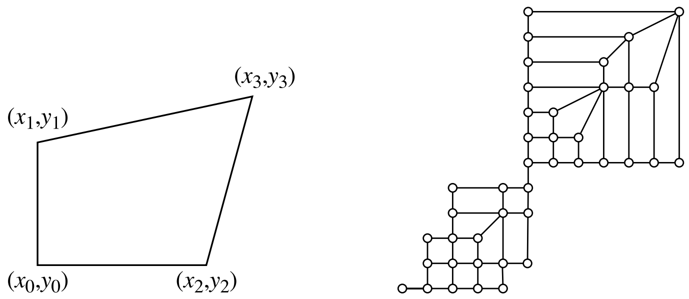


**Fig. 1.** Left: An upright quadrilateral. Right: An upright-quad drawing.

*that is, that any two sets in $\mathcal{F}$ can be connected by a sequence of sets, such that any two consecutive sets in the sequence differ by a single element and the length of the sequence equals the size of the symmetric difference of the two sets.*

As their edges are oriented from smaller sets to larger ones, learning spaces are directed acyclic graphs. Lemma 3 implies that any learning space $G$ is a partial cube when viewed as an undirected graph. Axiom 1 implies that the empty set belongs to $\mathcal{F}$, and that any other set has an incoming edge; that is, the empty set forms the unique source in $G$. Closure under unions implies that $\bigcup \mathcal{F}$ is the unique sink in $G$. Thus, $G$ is *st-oriented* (or has a *bipolar orientation*): it is a DAG with a single source and a single sink. In this paper we are particularly concerned with learning spaces for which this orientation is compatible with a planar drawing of the graph, in that the source and sink can both be placed on the outer face of a planar drawing. A graph admitting such a drawing is an *st-planar learning space*.

## 3 Upright-Quad Drawings

In any point set in the plane, we say that $(x,y)$ is *minimal* if no point $(x',y')$ in the set has $x' < x$ or $y' < y$, and *maximal* if no point $(x',y')$ in the set has $x' > x$ or $y' > y$.

We define an *upright quadrilateral* to be a convex quadrilateral with a unique minimal vertex and a unique maximal vertex, such that the edges incident to the minimal vertex are horizontal and vertical. That is, it is the convex hull of four vertices $\{(x_i,y_i) \mid 0 \le i < 4\}$ where $x_0 = x_1 < x_2 \le x_3$ and $y_0 = y_2 < y_1 \le y_3$ (Figure 1(left)). We define the *bottom edge* of an upright quadrilateral to be the horizontal edge incident to the minimal vertex, the *left edge* to be the vertical edge incident to the minimal vertex, and the *top edge* and *right edge* to be the edges opposite the bottom and left edges respectively.

We define an *upright-quad drawing* of a graph $G$ to be a placement of the vertices of the graph in the plane, with the following properties:

**[U1]** The placement forms a planar straight line drawing. That is, any two vertices are assigned distinct coordinates, and if the edges of $G$ are drawn as straight line segments then no two edges intersect except at their endpoints.

**[U2]** There is a unique vertex of $G$ that is the minimal point among the locations of its neighbors in $G$, and a unique vertex of $G$ that is the maximal point among the locations of its neighbors in $G$.

**[U3]** Every interior face of the drawing is an upright quadrilateral, the sides of which are edges of the drawing.

In an upright-quad drawing, all edges connect a pair of points $(x,y)$ and $(x',y')$ with $x' \le x$ and $y' \le y$; if we orient each such edge from $(x',y')$ to $(x,y)$ then the resulting graph is directed acyclic with a unique source and sink. As we now show, with this orientation the drawing is a *dominance drawing*: that is, the dominance relation in the plane and the reachability relation in the graph coincide.

**Lemma 4.** *For any two vertices $(x',y')$ and $(x,y)$ in an upright-quad drawing, $(x' \le x) \wedge (y' \le y)$ if and only if there exists a directed path in the orientation specified above from $(x',y')$ to $(x,y)$.*

*Proof.* In one direction, if there exists a directed path from $(x',y')$ to $(x,y)$, then each edge in the path steps from a vertex to another vertex that dominates it, and the result holds by transitivity of dominance.

In the other direction, suppose that $(x,y)$ dominates $(x',y')$; we must show the existence of a directed path from $(x',y')$ to $(x,y)$. To do so, we show that we can find an outgoing edge to another vertex dominated by $(x,y)$; the result follows by induction on the number of vertices. First consider the case that $(x',y')$ is the minimal corner of some upright quadrilateral of the drawing; then there exist both horizontal and vertical outgoing edges from $(x',y')$. $(x,y)$ cannot belong to the bounding rectangle of these two edges, for if it did we could not use them as part of an empty upright quadrilateral, so at least one of the two edges leads to another vertex that is also dominated by $(x,y)$.

In the second case, suppose $(x',y')$ belongs to the bottom side of some upright quadrilateral of the drawing but is not the minimal vertex of that face. The sequence of faces of the drawing on a vertical line through $x'$, above $(x',y')$, must project to a sequence of nested intervals on the $y$ axis, for each consecutive pair of faces shares an edge which has the same projection onto the $y$ axis as the lower of the two faces. Therefore, $(x,y)$ cannot project to a point interior to the projection of the face above $(x',y')$, so the horizontal outgoing edge from $(x',y')$ leads to a vertex that is also dominated by $(x,y)$. The case that $(x',y')$ belongs to the left side of an upright quadrilateral but is not its minimal vertex is symmetric to this one.

Finally, suppose that $(x',y')$ is not on the bottom or left side of any upright quadrilateral. Then there can only be a single edge outgoing from $(x',y')$, and there can be no interior faces of the drawing directly above or to the right of this edge. Thus, again, $(x,y)$ cannot project into the interior of the projection of this edge in either coordinate axis, so this edge leads to a vertex that is also dominated by $(x,y)$. □

As is well known to the graph drawing community [2,3], a dominance drawing exists for any *st*-oriented plane graph in which the source $s$ and sink $t$ of the orientation belong to the outer face of the plane embedding; such a graph is known as an *st-planar graph*. However, due to property U3, not every *st*-planar dominance drawing is an upright-quad drawing.

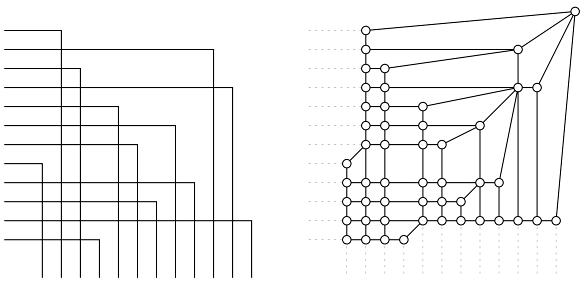


**Fig. 2.** Left: an arrangement of quadrants. Right: the region graph of the arrangement, drawn with each vertex (except the top right one) at the maximal point of its region.

## 4 Arrangements of Quadrants

Consider a collection of convex wedges in the plane, all translates of each other. Recall [5, 10] that a *weak pseudoline arrangement* is a collection of curves in the plane, each topologically equivalent to a line and extending to infinity at both ends, such that any two non-disjoint curves meet in a single crossing point. If no two wedges have boundaries on the same line, the boundary curves of the wedges form such an arrangement, for any two translates of the same wedge can only meet in a single crossing point.

By an appropriate linear transformation, we may transform our wedges to any desired orientation and convex angle, without changing the combinatorics of their arrangement. For later convenience, we choose a standard form for such arrangements in which each wedge is a translate of the negative quadrant $\{(x,y) \mid x,y \leq 0\}$ (Figure 2(left)). For such wedges, the condition that no two quadrants share a boundary line is equivalent to all translation vectors having distinct $x$ and $y$ coordinates. We call an arrangement of translated negative quadrants satisfying this distinctness condition an *arrangement of quadrants*. We refer to the curves of the arrangement, and to the wedges they form the boundaries of, interchangeably.

As with any arrangement of curves, we may define a *region graph* that is the planar dual of the arrangement: it has one vertex per region of the arrangement, with two vertices adjacent whenever the corresponding regions are adjacent across a nonzero length of curve of the arrangement. For our arrangements of quadrants, it is convenient to draw the region graph with each region's vertex in the unique maximal point of its region, except for the upper right region which has no maximal point. We draw the vertex for the upper region at any point with $x$ and $y$ coordinates strictly larger than those of any curve in our arrangement. The resulting drawing is shown in Figure 2(right).

**Theorem 1.** *The placement of vertices above produces an upright-quad drawing for the region graph of any arrangement of quadrants.*

*Proof.* The drawing's edges consist of all finite segments of the arrangement curves, together with diagonal segments connecting corners of arrangement curves within a region to the region's maximal point; therefore it is planar. Each finite region of the arrangement is bounded above and to the right by a quadrant, either a single curve of the arrangement or the boundary of the intersection of two of the wedges of the arrangement. Each finite region is also bounded below and to the left by a staircase formed by a union of wedges of the arrangement; the drawing's edges subdivide this region into upright quadrilaterals by diagonals connecting the concave corners of the region to its maximal point. A similar sequence of upright quadrilaterals connects the staircase formed by the union of all arrangement wedges to the point representing the upper right region, which is the unique maximal vertex of the drawing. The unique minimal vertex of the drawing represents the region formed by the intersection of all arrangement wedges. Thus, all requirements of an upright-quad drawing are met. □

**Theorem 2.** *The region graph of any arrangement of quadrants can be oriented to represent an st-planar learning space.*

*Proof.* We associate with each vertex of the region graph the set of wedges that do not contain any point of the region corresponding to the vertex. Each region other than the one formed by intersecting all wedges of the arrangement (associated with the empty set) has at least one arrangement curve on its lower left boundary; crossing that boundary leads to an adjacent region associated with a set of wedges omitting the one whose boundary was crossed; therefore axiom L1 of a learning space is satisfied.

If a region of the arrangement, associated with set $S$, has a single arrangement curve $c$ as its upper right boundary, then all supersets of $S$ associated with other regions are also supersets of $S \cup \{c\}$. Thus, in this case, there can be no two distinct sets $S \cup \{x\}$ and $S \cup \{y\}$ in the family of sets associated with the region graph, and axiom L2 of a learning space is satisfied vacuously.

On the other hand, if a region $r$ of the arrangement, associated with set $S$, has curve $x$ as its upper boundary and curve $y$ as its right boundary, then the only sets in the family formed by adding a single element to $S$ can be $S \cup \{x\}$ and $S \cup \{y\}$. In this case, the region diagonally opposite $r$ across the vertex where $x$ and $y$ meet is associated with the set $S \cup \{x,y\}$ and again axiom L2 of a learning space is met. □

Not all upright-quad drawings are formed from arrangements of quadrants as described here. For instance, a single square is itself an upright-quad drawing, but not one formed in this way. Nevertheless, as we describe in the rest of the paper, Theorems 1 and 2 have converses, in that any *st*-planar learning space can be given an upright-quad drawing and any upright-quad drawing is combinatorially equivalent to the region graph of an arrangement of quadrants. Thus, these three seemingly different concepts, *st*-planar learning spaces, upright-quad drawings, and region graphs of arrangements of quadrants, are shown to be three faces of the same underlying mathematical objects.

## 5 Drawing *st*-Planar Learning Spaces

As we have seen, learning spaces are *st*-oriented. Thus, when considering drawing algorithms for these graphs, it is natural to consider the special case in which the *st*-orientation is consistent with a planar embedding; that is, when the graph is *st*-planar. As we show in this section, every *st*-planar learning space has an upright-quad drawing. An example of an *st*-planar learning space is shown in Figure 3; in the left view, the vertices of a dominance drawing of the graph are labeled by the corresponding sets in the

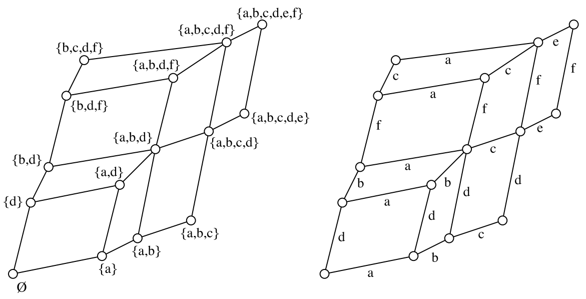


**Fig. 3.** An *st*-planar learning space

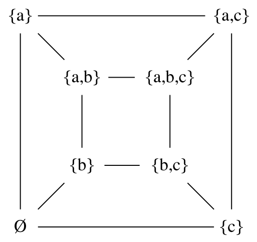


**Fig. 4.** A learning space that is planar but not *st*-planar (the power set on three elements)

family $\mathcal{F}$, while on the right view, each edge is labeled by the single element by which the sets at the two ends of the edge differ. However, not all planar learning spaces are *st*-planar; Figure 4 shows an example of a learning space that is planar but not *st*-planar.

**Lemma 5.** *Let G be an st-planar learning space. Then every interior face of G is a quadrilateral, with equal labels on opposite pairs of edges.*

*Proof.* Let $b$ be the bottom vertex of any interior face $f$, with outgoing edges to $b \cup \{x\}$ and $b \cup \{y\}$. Then by axiom L2 of learning spaces, $G$ must contain a vertex $b \cup \{x,y\}$. This vertex must be the top vertex of $f$, for otherwise the edges from $b \cup \{x\}$ to $b \cup \{x,y\}$ and from $b \cup \{y\}$ to $b \cup \{x,y\}$ would have to pass above the top vertex, implying a subset relationship from the top vertex to $b \cup \{x,y\}$, which is absurd. □

Define a *zone* of a label $x$ in an *st*-planar learning space $G$ to be the set of interior faces containing edges labeled by $x$. By Lemma 5, zones consist of chains of faces linked by opposite pairs of edges. We may form a curve arrangement $\mathcal{A}(G)$ from an *st*-planar graph $G$ by drawing a curve through each face of each zone, crossing only edges of $G$ with the label of the zone. Within each face, there are two curves, which we may draw in such a way that they cross once; they may also be extended to infinity past the exterior edges of the drawing without any crossings in the exterior face (Figure 5(left)). $\mathcal{A}(G)$ can be viewed as a form of planar dual to $G$, in that it has one vertex within each face of $G$, one face containing each vertex of $G$, and one arrangement segment crossing each edge of $G$; however it lacks a vertex dual to the outer face of $G$.

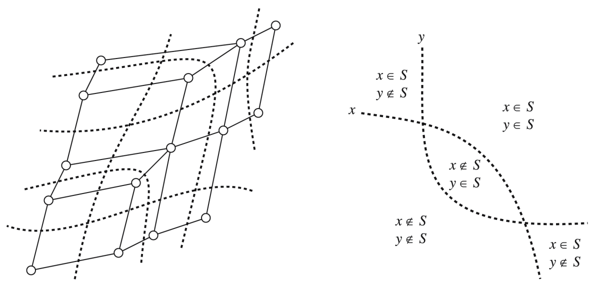


**Fig. 5.** Left: The curve arrangement $\mathcal{A}(G)$ dual to an *st*-planar learning space. Right: Two crossings between the same two curves lead to a contradiction, so $\mathcal{A}$ must be a weak pseudoline arrangement (Lemma 6).

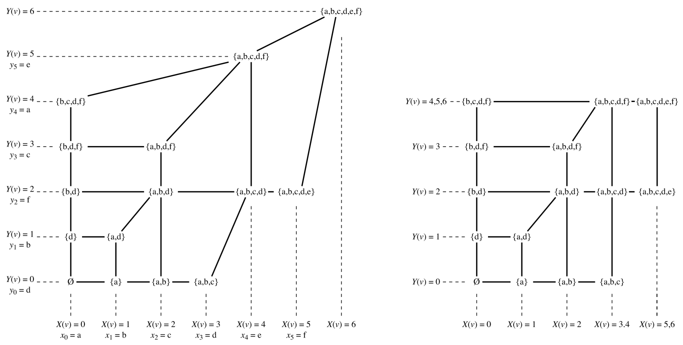


**Fig. 6.** Left: coordinates for conversion of *st*-planar learning space to upright-quad drawing. Right: the same drawing with compacted coordinates.

**Lemma 6.** *If $G$ is an st-planar learning space, then $\mathcal{A}(G)$ is a weak pseudoline arrangement.*

*Proof.* The curves in $\mathcal{A}(G)$ are topologically equivalent to lines and meet only at crossings. Suppose for a contradiction that two curves labeled $x$ and $y$ in $\mathcal{A}(G)$ cross more than once. Then (Figure 5(right)) two different regions between these curves would contain vertices corresponding to sets containing $x$ and not containing $y$ or vice versa. But then every path from one such set to another would cross one or the other of the two curves, contradicting the assumption that $G$ is a partial cube and has shortest paths labeled only by the elements of the symmetric difference of the two path endpoints. □

We are now ready to define the vertex coordinates for our upright-quad drawing algorithm. Consider the sequence of labels $x_0, x_1, \ldots x_{\ell-1}$ occurring on the right path from the bottom to the top vertex of the external face of the drawing. For any vertex $v$ of our given *st*-planar learning space, let $X(v) = \min\{i \mid x_i \notin v\}$. If $v$ is the topmost vertex of the drawing, define instead $X(v) = \ell$. Similarly, consider the sequence of labels $y_0, y_1, \ldots y_{\ell-1}$ occurring on the left path from the bottom to the top vertex of the external face of the drawing. For any vertex $v$ of our given $st$-planar learning space, let $Y(v) = \min\{i \mid y_i \notin v\}$. If $v$ is the topmost vertex of the drawing, define instead $Y(v) = \ell$.

**Lemma 7.** *Let $G$ be an st-planar learning space, with $y_i$ as above, let $i < j < k$, and suppose that the curves labeled $y_i$ and $y_k$ both cross the curve labeled $y_j$ in the arrangement $\mathcal{A}(G)$. Then the crossing with $y_k$ occurs to the left of the crossing with $y_i$.*

*Proof.* Otherwise, the arrangement would contain a set containing $y_i$ but not $y_j$ in the region left of the crossing between $y_i$ and $y_j$, and a set containing $y_k$ but not $y_j$ in the region right of the crossing between $y_j$ and $y_k$. However, as $y_i$ and $y_k$ could only cross above $y_k$, there could be no sets containing both $y_i$ and $y_k$ but not $y_j$, violating the closure of a learning space's sets under union (Lemma 3). □

**Lemma 8.** *If we place each vertex $v$ of an st-planar learning space $G$ at the coordinates $(X(v), Y(v))$, the result is an upright-quad drawing of $G$.*

*Proof.* It is clear from the definitions that each edge of $G$ connects vertices with monotonically nondecreasing coordinates. We show that each internal face is an upright quadrilateral. Consider any such face $f$, with bottom vertex $b$, top vertex $t$, bottom and top edges labeled $x_i$, and left and right edges labeled $y_j$. Then, for any edge label $y_k$ with $k < j$, $y_k \in b$; for otherwise, the curve for $y_k$ would cross the curve for $y_j$ to the right of $f$, and curves $x_i$, $y_j$, and $y_k$ would violate Lemma 7. Thus, the vertices $b$ and $b \cup \{x\}$ of $f$ are placed at $y$-coordinate value $j$, and the other two vertices have $y$-coordinates larger than $j$. Symmetrically, the vertices $b$ and $b \cup \{y\}$ of $f$ have $x$-coordinate $i$, and the other two vertices have $x$-coordinates larger than $i$.

This shows that all edges that are bottom or left edges of an interior face of the drawing are horizontal or vertical. If $e$ is not such an edge, then it belongs to the left or right exterior path of the drawing. If on the left path, it connects a vertex $\{y'_i \mid i' < i\}$ to $\{y'_i \mid i' \leq ii\}$ and thus has strictly increasing $y$ coordinates; symmetrically, if on the right path, it has strictly increasing $y$ coordinates. Thus all such edges also have the correct dominance order for their vertices.

As all edges are oriented correctly, the drawing must have a unique minimal vertex and a unique maximal vertex, the source and sink of $G$ respectively. Together with each face being an upright quadrilateral, this property shows that the drawing is an upright-quad drawing. □

A drawing produced by the technique of Lemma 8 is shown in Figure 6(left). As in standard $st$-planar dominance drawing algorithms [2], we may compact the drawing by merging coordinate values $X(v) = i$ and $X(v) = i + 1$ whenever the merge would preserve the dominance ordering of the vertices; a compacted version of the same drawing is shown on the right of Figure 6.

**Theorem 3.** *Every st-planar learning space $G$ over a set $U$, having $n$ vertices, has an upright-quad drawing in an integer grid of area $(|U| + 1)^2$ that may be found in time $O(n)$.*

*Proof.* We construct an $st$-planar embedding for $G$, form from it the dual curve arrangement $\mathcal{A}(G)$, and use the indices of the curves to assign coordinates to vertices as above.

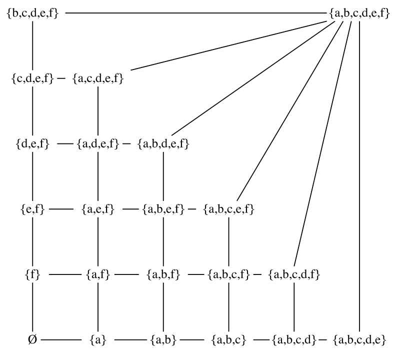


**Fig. 7.** The family of sets formed by the union of a prefix and a suffix of some ordered universe forms an *st*-planar learning space with $1+(|U|+1)|U|/2$ states

The coordinates of the vertices in a face $f$ may be assigned by referring only to the labels of edges in $f$, in time $O(|f|)$; therefore, all coordinates of $G$ may be assigned in linear time. The area bound follows easily. □

**Corollary 1.** *Any st-planar learning space over a set U has at most* $1+(|U|+1)|U|/2$ *states.*

*Proof.* Our drawing technique assigns each vertex (other than the topmost one) a pair of coordinates associated with a pair of elements $\{x_i, y_j\} \subset U$ (possibly with $x_i = y_j$), and each pair of elements can supply the coordinates for only one vertex. Thus, there can only be one more vertex than subsets of one or two members of $U$. □

The bound of Corollary 1 is tight, as the family of sets $\mathcal{F}$ that are the unions of a prefix and a suffix of a totally ordered set $U$ (Figure 7) forms an *st*-planar learning space with exactly $1+(|U|+1)|U|/2$ states.

## 6 From Drawings to Quadrant Arrangements

Define a *zone* of an upright-quad drawing to be a maximal sequence of interior faces adjacent on opposite sides of each quadrilateral (Figure 8(left)). A zone consists of a vertical sequence of quadrilaterals sharing horizontal sides and a horizontal sequence of quadrilaterals sharing vertical sides, connected across a diagonal edge; either or both sequence may be empty. We consider any bridge of the graph to form a zone of its own.

**Theorem 4.** *Each upright-quad drawing is the region graph for an arrangement of quadrants.*

*Proof.* For each zone $z_i$, we choose a coordinate value $x_i$, larger than the $x$-coordinate of the left endpoint of the bottom edge of the zone and (if the bottom edge is non-vertical) smaller than the $x$-coordinate of the right endpoint of the edge. We similarly choose a coordinate value $y_i$, larger than the $y$-coordinate of the bottom endpoint of the left edge

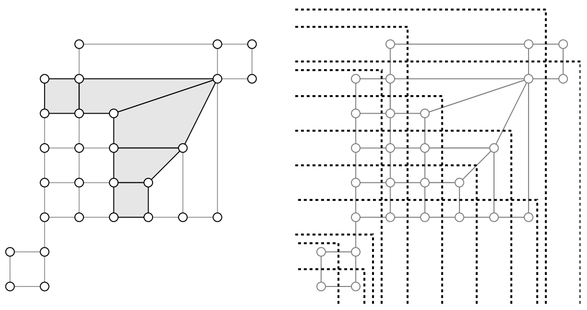


**Fig. 8.** Left: A zone in an upright-quad drawing. Right: an arrangement of quadrants through each zone.

of the zone and (if the left edge is non-horizontal) smaller than the $y$-coordinate of the top endpoint of the edge. We choose these coordinates in such a way that, if zone $i$ meets the right exterior path of the drawing prior to zone $j$, then $x_i < x_j$, and, if zone $i$ meets the left exterior path of the drawing prior to zone $j$, then $y_i < y_j$. We then draw a curve for zone $z_i$ by combining a horizontal ray left from $(x_i, y_i)$ with a vertical ray down from $(x_i, y_i)$ (Figure 8(right)). This curve is easily seen to cross all faces of zone $z_i$, and no other interior faces of the drawing; thus, the arrangement $\mathcal{A}$ of these curves forms a planar dual to the drawing (except, as before, that it does not have a vertex representing the external face). □

**Corollary 2.** *Each upright-quad drawing represents an st-planar learning space.*

*Proof.* This follows from Theorem 2 and Theorem 4. □

As different sets of translation vectors for quadrants form combinatorially equivalent arrangements if and only if the sorted orders of their $y$-coordinates form the same permutations with respect to the sorted order of their $x$-coordinates, a bound on the number of $st$-planar learning spaces follows.

**Corollary 3.** *There are at most $n!$ combinatorially distinct st-planar learning spaces over a set of n unlabeled items.*

More precisely, with high probability any permutation corresponds to a learning space with only two combinatorially distinct upright-quad drawings (one formed by flipping the other diagonally), so the number of distinct $st$-planar learning spaces is $\frac{1}{2}n!(1-o(1))$.

## 7 Conclusions

We have characterized $st$-planar learning spaces, both in terms of the existence of an upright-quad drawing and as the region graphs of quadrant arrangements. Our technique for drawing these graphs provides good vertex separation and small area, and it is straightforward to verify from its drawing that a graph is an $st$-planar learning space.

Our results can be viewed as showing that the *convex dimension* of an antimatroid is two if and only if its order dimension is two. It is known that the order dimension is always upper bounded by the convex dimension [13, Corollary III.6.10]. However, these two quantities are not always equal. For instance, the antimatroid over $\{0,1,2,3,4\}$ formed by the subsets that either don't contain 0 or do contain three or more items has order dimension at most five, while it has convex dimension six. We note that the convex dimension is computable in polynomial time as the width of a related poset [13, Theorem III.6.9], and are hopeful that this result may also lead to interesting methods of graph drawing, analogously to how our minimum-dimensional lattice embedding technique [7] led to drawing algorithms for arbitrary nonplanar partial cubes [5].

Alternatively, when confronted with the task of drawing large nonplanar learning spaces, it may be helpful to find large *st*-planar subgraphs and apply the techniques described here to those subgraphs. Additionally, it would be of interest to extend our *st*-planar drawing techniques to broader classes of partial cubes.

## References

1. E. Cosyn and H. Uzun. Axioms for learning spaces. To be submitted to *Journal of Mathematical Psychology*, 2005.
2. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.
3. G. Di Battista, R. Tamassia, and I. G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discrete & Comput. Geom.* 7:381–401, 1992.
4. J.-P. Doignon and J.-C. Falmagne. *Knowledge Spaces*. Springer-Verlag, 1999.
5. D. Eppstein. Algorithms for drawing media. *Proc. 12th Int. Symp. Graph Drawing (GD 2004)*, pp. 173–183. Springer-Verlag, Lecture Notes in Computer Science 3383, 2004, arXiv:cs.DS/0406020.
6. D. Eppstein. Cubic partial cubes from simplicial arrangements. arXiv.org, October 2005, arXiv:math.CO/0510263.
7. D. Eppstein. The lattice dimension of a graph. *Eur. J. Combinatorics* 26(5):585–592, July 2005, http://dx.doi.org/10.1016/j.ejc.2004.05.001, arXiv:cs.DS/0402028.
8. D. Eppstein. What is the dimension of the set of partitions? Unpublished web document, http://11011110.livejournal.com/6402.html, 2006.
9. D. Eppstein, M. T. Goodrich, and J. Y. Meng. Delta-confluent drawings. *Proc. 13th Int. Symp. Graph Drawing (GD 2005)*, pp. 165–176. Springer-Verlag, Lecture Notes in Computer Science 3843, 2006, arXiv:cs.CG/0510024.
10. H. de Fraysseix and P. Ossona de Mendez. Stretching of Jordan arc contact systems. *Proc. 11th Int. Symp. Graph Drawing (GD 2003)*, pp. 71–85. Springer-Verlag, Lecture Notes in Computer Science 2912, 2003.
11. Y.-F. Hsu, J.-C. Falmagne, and M. Regenwetter. The tuning in-and-out model: a random walk and its application to presidential election surveys. Submitted, 2002.
12. W. Imrich and S. Klavžar. *Product Graphs*. John Wiley & Sons, 2000.
13. B. Korte, L. Lovász, and R. Schrader. *Greedoids*. Algorithms and Combinatorics 4. Springer-Verlag, 1991.
14. S. V. Ovchinnikov. Media theory: representations and examples. To appear in *Discrete Applied Mathematics*, arXiv:math.CO/0512282.

# Choosing Colors for Geometric Graphs Via Color Space Embeddings

Michael B. Dillencourt, David Eppstein, and Michael T. Goodrich

Dept. of Computer Science, Univ. of California, Irvine, CA 92697-3425 USA
{dillenco,eppstein,goodrich}@ics.uci.edu

**Abstract.** Graph drawing research traditionally focuses on producing geometric embeddings of graphs satisfying various aesthetic constraints. After the geometric embedding is specified, there is an additional step that is often overlooked or ignored: assigning display colors to the graph's vertices. We study the additional aesthetic criterion of assigning distinct colors to vertices of a geometric graph so that the colors assigned to adjacent vertices are as different from one another as possible. We formulate this as a problem involving perceptual metrics in color space and we develop algorithms for solving this problem by embedding the graph in color space. We also present an application of this work to a distributed load-balancing visualization problem.

**Keywords:** graph drawing, graph coloring, color space, color perception.

## 1 Introduction

Graphs are frequently visualized by embedding them in geometric spaces. That is, geometric representations are natural tools for visualizations; hence, we embed graphs in geometric spaces in order to display them. For instance, producing geometric embeddings of combinatorial graphs so as to satisfy various aesthetic constraints is a major component of *graph drawing* (e.g., see [5,9,10,13]).

Once a graph has been embedded in a geometric space, such as $\mathbf{R}^2$, we refer to it as a *geometric graph.* That is, a geometric graph is a graph $G = (V, E)$ such that the vertices are geometric objects in $\mathbf{R}^d$ and the edges are geometric objects connecting pairs of vertices. Note that this definition is more general than the definition of "geometric graph" popularized by Alon and Erdös [1], in that they define a geometric graph to be a graph $G = (V, E)$ such that the vertices are distinct points in $\mathbf{R}^2$ and edges are straight line segments. For example, we allow a geometric graph to be a planar map, where the vertices are regions and the edges are defined by regions that share a common border.

Intuitively, a geometric graph $G$ is a graph that is "almost drawn," because displaying $G$ requires assigning colors to its vertices. One obvious method of doing this—a very common one—is to ignore the issue and color all the vertices black. In this paper, we examine the color-choosing step more carefully. In particular, are interested in methods for choosing colors for the vertices of a geometric graph so as to make distinctions between vertices as apparent as possible. We are also interested in the related *map coloring* problem, where we color the faces of a map so as to make the distinctions between adjacent faces as strong as possible. Part of the challenge is choosing a good set of colors, but we also want to assign colors to vertices in a way that makes the colors assigned to adjacent vertices as different as possible. That is, we are interested in a bi-criterion color assignment problem, where all the colors are different from one another and adjacent colors are really different.

## 1.1 Previous Related Work

Graph coloring is a classic problem in algorithmic graph theory (e.g., see [3]). Given a graph $G$, the traditional version of this problem is to color the vertices of $G$ with as few colors as possible so that adjacent vertices always have different colors. The traditional graph coloring problem is posed as a "coloring" problem purely for abstraction's sake, however: no paint or pixels are involved. Even so, there has been some prior work on algorithms for coloring geometric graphs (in the traditional sense). For example, there has been some prior research on coloring quadtrees [2], intersection graphs [6], and arrangements [7]. In addition, there has been a host of prior work on the traditional version of graph coloring for purely combinatorial graphs (e.g., see [3]).

Also of interest is work that has been published in the information visualization literature on methods for choosing colors effectively for data presentation. Healey [8] presents a heuristic for choosing a well-separated set of colors for visualizing segmentation data in images. Likewise, Levkowitz and Herman [12], Robertson [17], and Ware [23] discuss various ways for effectively building color maps that correspond to data values in an image or data visualization (e.g., a bar chart histogram). Rheingans and Tebbs [16] describe an interactive approach that constructs a color scale by tracing a path through color space. Brewer [4] describes several guidelines for choosing colors for data visualization, focusing primarily on ways of representing linear numerical scales. There are also several good books on the subject of color use for data visualization (e.g., see [20,21,22]). In spite of this wealth of previous work on color selection for data visualization, we are unfamiliar with any prior work that uses adjacency information to select dissimilar colors for visualization purposes.

## 1.2 Our Results

In this paper, we investigate the following problem for geometric graph coloring:

> *Maximizing minimum color difference.* Given a geometric graph $G$ and a color space $\mathcal{C}$, assign visibly distinct colors from $\mathcal{C}$ to the vertices of $G$ so to maximize the minimum color difference across the endpoints of edges in $G$.

We investigate this problem in terms of embeddings of $G$ in the human-perceptible subset of the color space $\mathcal{C}$. This embedding of $G$ in $\mathcal{C}$ is purely to find good colors to assign to the vertices of $G$, however. The actual coordinates for $G$'s vertices and equations for $G$'s edges will still use $G$'s geometric embedding in $\mathbf{R}^d$ (e.g., as produced by an existing graph-drawing algorithm). Nevertheless, the placement of vertices and edges in our embedding of $G$ in $\mathcal{C}$ implies a "goodness" score on the degree to which adjacent vertices are well-separated and non-adjacent vertices are fairly-separated (which corresponds to a similar degree of separation for the vertex colors when we display $G$ using its original geometry). We design a force-directed algorithm to produce such embeddings.

By planar duality, our algorithms are also applicable to the *map coloring* problem, where we are given a planar map and asked to color the regions with distinct colors so that the color difference between bordering regions is as large as possible. We give an application of this map coloring problem to an interesting data visualization problem for load-balancing distributed numerical algorithms.

## 2 Color Space

Since we wish to assign colors to the vertices of a geometric graph so that colors assigned to adjacent vertices look as different as possible, it is useful to have a precise, mathematical notion of color and color difference.

Pure colors can be defined in terms of wavelengths of light, with the visible spectrum of colors going roughly from 400 nm (violet) to 800 nm (red). Humans perceive color, however, as a combination of intensity signals from three types of cone cells in our eyes:

- **S cone cells:** These cells respond to short wavelengths and typically have their peak transmission around 440 nm (violet). (For historical reasons, these cells are often referred to as "blue" cone cells.)
- **M cone cells:** These cells respond to medium wavelengths and typically have their peak transmission around 550 nm (yellow-green). (For historical reasons, these cells are often referred to as "green" cone cells.)
- **L cone cells:** These cells respond to long wavelengths and typically have their peak transmission around 570 nm (yellow). (For historical reasons, these cells are often referred to as "red" cone cells.)

This physiology forms the basis of all color displays, from old-fashioned color TVs to modern-day color LCD panels and plasma displays, for these displays create what we perceive as colors by an additive combination of three color intensities, such as red, green, and blue (RGB, as exemplified by the sRGB space [19] used by many digital cameras and color displays). This physiology also forms the basis of most color printing, as well, where printers create what we perceive as colors by the subtractive combination of three color intensities, such as cyan, magenta, and yellow (CMY). Thus, colors can be viewed as belonging to a three-dimensional *color space*. Moreover, RGB color spaces, which define three-dimensional cubes of color values, correspond to the way most modern devices display colors.

Ironically, even though RGB spaces are the most popular for display devices, humans are very poor at interpreting the perceived color that results from the addition of intensity values in red, green, and blue. Moreover, perceived color differences do not define a uniform metric in RGB spaces. Our brains instead use the following notions:

- **Hue:** the actual color, e.g., "blue," "yellow," "orange," etc., as defined by a radial value around a color wheel.
- **Saturation:** the vividness or dullness of the color.
- **Luminosity:** the lightness or darkness of the color.

Thus, human perceived color defines a three-dimensional color space, called HSL, which corresponds to two cylindrical cones joined at their base, as shown in Figure 1. The two apexes of these cones correspond to opposite corners in RGB space. As with RGB, however, perceived color differences do not define a uniform metric in the HSL color space. Moreover, the geometry of the HSL space makes choosing colors inside its double-cone of visible colors more challenging.

CIE L*a*b* (or "Lab," for short) is an absolute color space that defines each color uniquely as a combination of Luminosity (L), a value, a*, which is a signed number that indicates the degree of magenta (positive) or green (negative) in a color, and a value, b*, which is a signed number that indicates the degree of yellow (positive) or blue (negative) in a color. Geometrically, Lab is a slightly distorted version of the HSL double-cone, with color points addressed using Cartesian coordinates. Thus, defining the subset of visible colors in Lab space is admittedly more challenging. Offsetting this drawback, however, is the fact that empirical evidence supports the claim that Euclidean distance in this color space corresponds to perceptual color difference [18]. There is a related, CIE L*u*v* color space, which also is designed to provide a uniform color-difference metric,

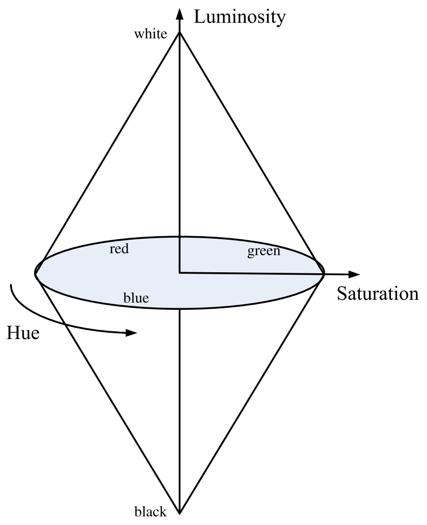


**Fig. 1.** The Hue-Saturation-Luminosity (HSL) color space

but the Lab color space seems to be more uniform. Thus, the Lab color space is the more popular of the two.

There is a tradeoff between the two most popular color spaces, then. RGB corresponds better to display hardware and it defines a simple cube geometry for the space of visible colors. But perceived color difference is not a uniform metric in RGB. Lab space, on the other hand, has a more complex geometry and requires a translation to RGB for display purposes, but it supports a uniform color-difference metric. In this paper, therefore, we explore color choosing algorithms for both of these spaces.

## 3 Application

A specific application motivating this research is a problem in distributed programming. The Navigational Programming (NavP) methodology [14] for converting a sequential program into a parallel distributed program using migrating threads consists of the following three steps:

1. **Data Distribution:** The data used by the program is distributed over the network. The guiding heuristic principle is minimizing communication cost while balancing the load on each processing element (PE).
2. **Computation Distribution:** Navigational commands ("hop" statements) are inserted into the sequential code. This step produces a *distributed sequential program*, a single thread that "follows" the data through the network.
3. **Pipelining:** The single migrating thread produced in step 2 is broken into multiple threads, which are then formed into a pipeline by adding appropriate synchronization commands.

The methodology incorporates a feedback loop: information obtained in step 3 can be used to improve the data distribution in a subsequent application of the three steps.

The data distribution step is based on constructing a Navigational Trace Graph (NTG), which relates communication costs to data placement, and then applying a graph partitioning heuristic. In the NTG, the vertices are the data elements, and edge weights between vertices reflect the cost of placing the corresponding data elements on different machines [15]. Among the factors influencing the edge weights between two data items are (1) whether one of them is used directly in the computation of the other; (2) whether they are physically allocated close together in the sequential program; (3) whether they are referenced in temporally consecutive statements in the sequential program. The first factor is a source of communication overhead if the data elements are assigned to different PE's, while the second and the third factors capture locality information that is implicit in the sequential code and may affect performance (e.g., by increasing cache reuse). Additional factors affect the partitioning: these include balancing the computational load and amount of data on each PE, and introducing constraints that certain data elements must be on different PE's (so as not to preemptively exclude parallelism that might be introduced in step 3).

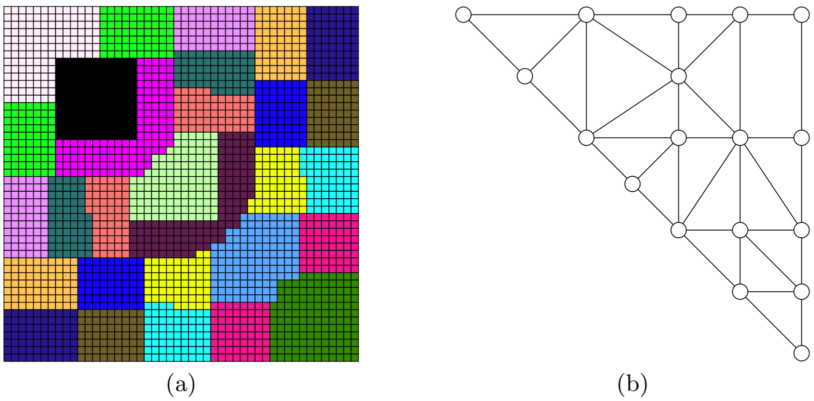


**Fig. 2.** The Navigational Programming application: (a) an example partition of an array of data into 18 regions for a NavP application, colored by our algorithm; and (b) the corresponding graph of adjacent regions

Since the interaction of these constraints can be complex, it is important to be able to visualize the resulting data partitions. One ingredient of a good visualization tool is effective use of color. Because the individual sets in the data partition are not necessarily connected, the color-assignment scheme should follow two basic principles: (1) The colors assigned to regions in the partition should be highly dissimilar, to make it easy to see the boundaries between regions. (2) All colors used should be somewhat dissimilar from each other, so that it is apparent which disconnected regions belong to the same set in the partition.

An example of a partition produced by this system is shown in Figure 2. The underlying sequential code for which this partition was constructed, adapted from Lee and Kedem [11], can be conceptualized as a sequence of scans over a square matrix, where the scans alternate between row-major and column-major order and also alternate directions. In each scan, the value of each element $A[i, j]$ is computed as a function of its neighbors and also the neighbors of its transposed element $A[j, i]$. As can be seen, the partition is somewhat irregular. Because of the strong data affinity between each element and its transpose, the sets are symmetric about the diagonal of the matrix, and some of them are not connected. Thus, because of the irregularity of the partitioning and the potential complexity of the partitions, it is useful to have a high-quality assignment of colors to regions in order to visualize the regions and their interactions.

## 4 What Is a Good Coloring?

Formally, our problem can be stated as follows. We are given as input an undirected graph $G$, the vertices of which have been partitioned into *regions* $r_i$. We would like to display this region structure, overlaid on a conventional drawing of the graph, by assigning distinct colors to vertices in different regions. Our task is to choose a color for each region, satisfying the following constraints:

– Each region must have a different color, and the colors assigned to regions must be visually distinct.
– If two regions $r_i$ and $r_j$ are adjacent in $G$ (that is, if some vertex in $r_i$ and some vertex in $r_j$ are adjacent), then it is especially important that regions $r_i$ and $r_j$ be given dissimilar colors. We desire that the colors of such adjacent regions be as dissimilar as possible, subject to the first constraint that all region colors be visually distinct.

To solve this problem, we construct a *region graph* $R$ (as in Figure 2(b)). We form one vertex in $R$ per region $r_i$, with regions $r_i$ and $r_j$ connected by an edge in $R$ if and only if they are adjacent. We view the problem of assigning colors to the regions as one of embedding $R$ geometrically, into a three-dimensional space representing the gamut of colors available on the display device. Ideally, distances in this space should represent the visual dissimilarity of a pair of colors. As mentioned above, color spaces such as Lab have been designed so that this dissimilarity can be approximated by a Euclidean distance in that space.

Thus, we have a geometric graph embedding problem: assign color coordinates in a color space $\mathcal{C}$ to each vertex of the region graph $R$, according to the dissimilarity criteria identified above. However, unlike the embedding problems coming from traditional graph drawing problems, we want to place vertices so that edges are long rather than short.

In order to formalize the problem, we define a *coloring* to be any mapping $\chi$ from the vertices of $R$ to the color space of interest. Let $d_{i,j}$ denote the distance between $\chi(r_i)$ and $\chi(r_j)$, as measured by an appropriate distance function corresponding to visual dissimilarity. Let $D$ be the dimension of the color space; in most instances we will have $D = 3$. Let $n$ be the number of vertices in the region graph. For any region $r_i$, let $N_i$ denote the set of adjacent regions in $R$. Finally, let $\Delta$ denote the diameter of the color space into which we are embedding our region graph. We define a quality measure $q(\chi)$ by the following equation:

$$q(\chi) = \sum_{r_i} \Big( \sum_{r_j \in R \setminus \{r_i\}} \frac{1}{d_{i,j}^{D+1}} + \frac{n^{1+1/D}}{\Delta^D} \sum_{r_j \in N_i} \frac{1}{d_{i,j}|N_i|} \Big).$$

One of our goals in defining a function of this form is that, by making the quality a sum of relatively simple terms, we may find its gradient easily, simplifying the application of standard numerical optimization techniques. There are two terms per region in this sum, both normalized to be of roughly equal significance.

The first term has the form $\sum_{r_j} d_{i,j}^{-(D+1)}$. We expect, in a good embedding of the region graph, that the regions will be roughly uniformly distributed around the region graph. The exponent $D + 1$ in this term is chosen with this assumption in mind: for infinitely many uniformly spaced regions with a spacing of $\delta$, $\sum d_{i,j}^{-(D+1)}$ will converge to $\Theta(\delta^{-(D+1)})$, being influenced most strongly by the regions nearest $r_i$. On the other hand, a similar sum with an exponent of $D$ or less would diverge, and thus lowering the exponent in this term would cause our quality measure to be dominated more by global than local concerns. For $n$ vertices in a $D$-dimensional region of diameter $\Delta$, we expect spacing $\delta = \Theta(\Delta n^{-1/D})$, and thus we expect

$$\sum_{r_j \in R \setminus R_i} \frac{1}{d_{i,j}^{D+1}} = \Theta(\delta^{-(D+1)}) = \Theta(\frac{n^{1+1/D}}{\Delta^{D+1}}).$$

The second term has the form $\sum_{r_j \in N_i} 1/(d_{i,j}|N_i|)$. We hope, especially in the case of relatively sparse region graphs, to have $d_{i,j}$ roughly proportional to $\Delta$ for all edges between adjacent regions $r_i$ and $r_j$. If these edges are all sufficiently long, the normalization by $|N_i|$ will leave this term roughly proportional to $1/\Delta$. The low exponent on the distance is acceptable as we wish this part of the quality measure to act long-range, causing adjacent regions to be placed far apart. The normalization factor $n^{1+1/D}\Delta^{-}D$ prior to the second term in our definition of $q$ is chosen to make the two terms of the sum roughly proportional.

## 5 Finding a Good Coloring

The problem of finding a good coloring can be approached with a standard gradient descent or hill climbing heuristic: choose initial vertex locations in color space, and then gradually move the locations in a direction that causes the most local improvement in our quality measure. This requires calculating the gradient of our quality measure, which is most easily done when our color space forms a normed vector space, preferably Euclidean. Lab color is ideal for this task, as it has been designed so that Euclidean distances in Lab color closely approximate visual dissimilarity. The same approach can also be applied directly to RGB-based color spaces such as sRGB, with some degradation in the goodness of fit between our quality measure and the visual dissimilarity of the resulting colors.

As our quality measure is linear, we may compute the gradient separately for the term of it applying to each region $r_i$. The gradient at $r_i$ is a vector-valued quantity, formed by summing for each $r_j$ a vector directed away from $r_j$. If $r_i$ and $r_j$ are not adjacent, this vector has length $(D+1)/d_{i,j}^{D+2}$. If $r_i$ and $r_j$ are adjacent, we add another vector in the same direction with length

$$\frac{n^{1+1/D}}{|N_i|\Delta^D} d_{i,j}^{-2}.$$

Gradient descent with these vectors will cause the locations of regions in color space to spread apart rapidly. But we do not allow this to continue unconstrained, as we must confine the colors of each region to the gamut of displayable colors on the intended output device. We considered several options for this confinement:

- We could add an additional term in the quality measure penalizing colors outside the allowable gamut. However, we do not wish to penalize colors near the boundaries of the gamut, because those boundaries provide saturated colors that are easy to visually distinguish. Nor do we wish to allow colors to drift very far beyond the gamut. So the penalty term would have to have a very steep derivative, making the numerical optimization more difficult.

- We considered clipping any color outside the gamut to the nearest color within the gamut. Like the penalty term, this method would affect only the colors that reach the boundaries of the allowable gamut, and the numerical optimization procedure would have difficulty propagating the effects of this clipping to the interior of the gamut. More significantly, this truncation could distort points near boundaries of the color space where the tangent plane is not perpendicular to the line from the point to the center of the gamut. Effectively, the truncation and the outward repulsive forces of the gradient descent would push these points along the boundary away from the center.
- We experimented with a procedure that, after each step of gradient descent, rescales the entire color space, so that all vertices again lie within the gamut of allowable colors. This seems to work acceptably well for symmetric color spaces such as the sRGB gamut. However, when we tried it with Lab colors, for which the color space is more stretched out in some directions than others, we found that this method tended to accentuated this stretching, causing the gamut to be compressed in the other directions. In particular, this led to significant desaturation of the resulting Lab colors.
- We finally settled on the following procedure: after each step of gradient descent, rescale (rather than truncating) the out-of-gamut points, while leaving the other points in place. In our experiments this method performed better than the other ones above, allowing the gradient descent to improve the color placement without distorting the gamut.

Our implementation chooses initial vertex locations at random within the color gamut. Then in each iteration it attempts to move the locations of the vertices in color space, one vertex at a time by three types of moves: random jumps, swaps with other vertices, and moving by a fixed step size in the gradient direction. For each of these three move types our algorithm accepts the move only when it improves the overall quality of the coloring. If an iteration fails to find any quality improvement, we reduce the step size and terminate the algorithm when this step size falls below a preset threshold.

## 6 Results of Our Implementation

As a proof of concept, we implemented our algorithm both for the sRGB and Lab color spaces, and compared the results with those from an algorithm that chooses colors randomly. As the color spaces we use form eight-vertex convex polyhedra, our algorithm will tend to choose colors at those vertices for graphs with eight or fewer regions. For this reason, we chose for our experiments a larger region graph in the form of an eighteen-vertex triangulation.

We believe that, ultimately, the most appropriate way to evaluate our results is human usability testing, but such experiments are beyond the scope of this paper. Our numerical quality measure is not suitable for comparing different algorithms, first because it is specific to a color space and would not allow easy comparison of colorings in different spaces, and second, because any such comparison would

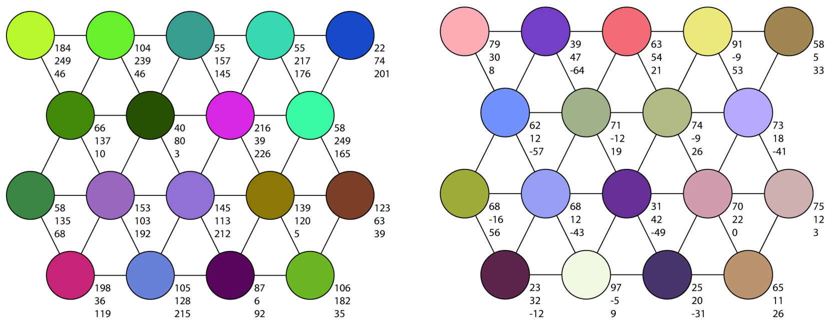


**Fig. 3.** Results of implementation: random assignment of colors to vertices. Left: random sRGB colors; right: random Lab colors.

not test how well our quality measure itself models the ease of understanding of drawings using our colorings. Thus, we only attempted a limited subjective analysis, based due to space limitations only on the results of a single run of each algorithm. Once each of our algorithms was working correctly, we ran it only once in order to avoid biasing our results by choosing subjectively among multiple runs; we note that an automated choice among multiple runs based on our quality measure would be possible, but would not differ in principle from a single run of a more sophisticated optimization procedure than our randomized hill climbing algorithm. The random nature of our algorithms means that the precise colors generated in our evaluation are not repeatable, and more systematic usability testing is needed to verify our results.

In the first implementation (Figure 3(left)), we chose random colors independently for all vertices, uniformly among the $2^{24}$ possible sRGB values. As expected, this did not work very well. The random assignment did not prevent several very similar colors from being chosen, often for adjacent regions. We performed a similar experiment with colors chosen uniformly at random in Lab color space, within the convex hull of the eight extreme sRGB colors (Figure 3(right)); the resulting colors seemed less heavily dominated by greens than the random sRGB results, but still included several very similar adjacent pairs of colors.

In the second implementation (Figure 4(left)) we applied our gradient descent optimization algorithm directly to the sRGB color space, using the quality measure we defined earlier via the Euclidean distance in this space despite the fact that this distance is known to fit human vision poorly. The algorithm chose a diverse selection of well saturated colors, and all pairs of adjacent regions have easily distinguishable colors. However, there are several nonadjacent colors that are difficult to distinguish: two yellows (254,254,0 and 255,255,145), two cyans (0,255,246 and 140,255,255), three blues (1,129,255, 0,0,245, and 130,0,255), two reds (255,112,99 and 255,0,1), and two pinks (255,4,255 and 255,171,255). We believe these faults are due to the poor match between Euclidean distance in sRGB color space and human visual dissimilarity.

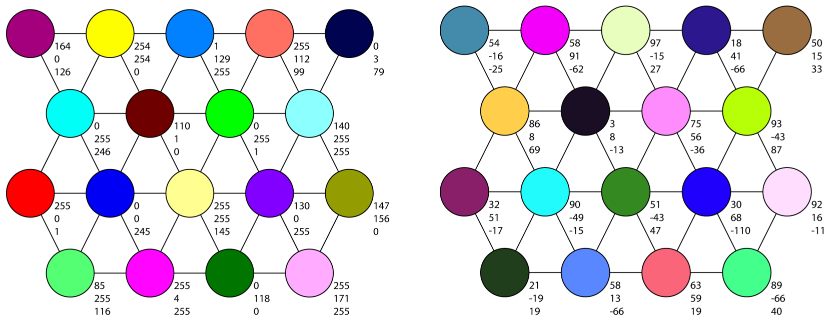


**Fig. 4.** Results of implementation: gradient descent in sRGB color space (left) and in Lab color space (right)

Finally, we applied our gradient descent algorithm for coordinates in the Lab color space (Figure 4(right)). The gamut of representable colors in Lab space is significantly larger than that for sRGB, so in order to ensure that our algorithm generated colors that could be displayed, we restricted all colors to a gamut formed geometrically as the convex hull in Lab space of the eight colors black, white, red, green, blue, cyan, magenta, and yellow forming the most extreme values of the sRGB color space. When our gradient descent algorithm caused vertices to be assigned colors outside this convex hull, as described above, we returned them to the gamut by a rescaling operation centered at the neutral gray color with Lab coordinates $50, 0, 0$. As with the sRGB output, the result of this algorithm was a collection of diverse well saturated colors, with all pairs of adjacent regions having easily distinguishable colors. Compared to the sRGB results, there were fewer sets of difficult to distinguish nonadjacent colors: primarily the two darker pinks (58,91,-62 and 75,56,-36). It is also somewhat difficult to distinguish the dark green (21,-19,19) from the black (3,8,-13). On the whole, it seems that using Lab color has led to a better selection of colors, and equally good assignment of the chosen colors to the vertices of the region graph.

## 7 Conclusion

We have given what we believe is the first color assignment algorithm that uses adjacency information in the input geometric graph to choose colors that are very different for adjacent vertices. For possible future work, one could consider a weighted version of the problem, where edges of the input geometric graph are weighted (e.g., by length) and we wish to assign colors so that the colors assigned to vertices of low-weight edges are more dissimilar than those on high-weight edges. Another interesting adaptation would be to perform our color assignment algorithm for color spaces corresponding to color-blind people (of which there are six types that collectively make up roughly 8% of the male population).

## References

1. N. Alon and P. Erdős. Disjoint edges in geometric graphs. *Discrete Comput. Geom.*, 4:287–290, 1989.
2. M. W. Bern, D. Eppstein, and B. Hutchings. Algorithms for coloring quadtrees. *Algorithmica*, 32(1):87–94, 2002.
3. J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* Macmillan, London, 1976.
4. C. A. Brewer. Color use guidelines for data representation. In *Proc. Section on Statistical Graphics, American Statistical Association*, pages 55–60, 1999.
5. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing.* Prentice Hall, Upper Saddle River, NJ, 1999.
6. D. Eppstein. Testing bipartiteness of geometric intersection graphs. In *Proc. 15th Symp. Discrete Algorithms*, pages 853–861. ACM and SIAM, 2004.
7. S. Felsner, F. Hurtado, M. Noy, and I. Streinu. Hamiltonicity and colorings of arrangement graphs. In *Proc. 11th Symp. Discrete Algorithms*, pages 155–164. ACM and SIAM, 2000.
8. C. G. Healey. Choosing effective colours for data visualization. In R. Yagel and G. M. Nielson, editors, *IEEE Visualization '96*, pages 263–270, 1996.
9. M. Jünger and P. Mutzel. *Graph Drawing Software.* Springer, 2003.
10. M. Kaufmann and D. Wagner. *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes in Computer Science.* Springer-Verlag, 2001.
11. P. Lee and Z. M. Kedem. Automatic data and computation decomposition on distributed memory parallel computers. *ACM Trans. Programming Languages and Systems*, 24(1):1–50, 2002.
12. H. Levkowitz and G. T. Herman. Color scales for image data. *IEEE Computer Graphics and Applications*, 12(1):72–80, 1992.
13. T. Nishizeki. *Planar Graph Drawing*, volume 12 of *LNSC.* World Scientific, 2004.
14. L. Pan, M. K. Lai, K. Noguchi, J. J. Huseynov, L. F. Bic, and M. B. Dillencourt. Distributed parallel computing using Navigational Programming. *Int. J. Parallel Programming*, 32(1):1–37, 2004.
15. L. Pan, J. Xue, M. K. Lai, L. Bic, M. B. Dillencourt, and L. Bic. Toward automatic data distributions for migrating computations. Submitted for publication, 2006.
16. P. Rheingans and B. Tebbs. A tool for dynamic explorations of color mappings. *Computer Graphics*, 24(2):145–146, 1990.
17. P. K. Robertson. Visualizing color gamuts: A user interface for the effective use of perceptual color spaces in data displays. *IEEE Computer Graphics and Applications*, 8(5):50–64, 1988.
18. Y. Rubner, J. Puzicha, C. Tomasi, and J. M. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. *Computer Vision and Image Understanding*, 84:25–43, 2001.
19. M. Stokes, M. Anderson, S. Chandrasekar, and R. Motta. A Standard Default Color Space for the Internet – sRGB. Available online at http://www.w3.org/pub/WWW/Graphics/Color/sRGB.html, 1996.
20. M. Stone. *A Field Guide to Digital Color.* Morgan Kaufmann, 2nd edition, 2003.
21. E. R. Tufte. *The Visual Display of Quantative Information.* Graphics Press, Cheshire, Connecticut, 1983.
22. E. R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, Connecticut, 1990.
23. C. Ware. Color sequences for univariate maps: Theory, experiments, and principles. *IEEE Computer Graphics and Applications*, 8(5):41–49, 1988.

# Morphing Planar Graphs in Spherical Space[⋆]

Stephen G. Kobourov and Matthew Landis

Department of Computer Science
University of Arizona
{kobourov,mlandis}@cs.arizona.edu

**Abstract.** We consider the problem of intersection-free planar graph morphing, and in particular, a generalization from Euclidean space to spherical space. We show that there exists a continuous and intersection-free morph between two sphere drawings of a maximally planar graph, provided that both sphere drawings have convex inscribed polytopes, where sphere drawings are the spherical equivalent of plane drawings: intersection-free geodesic-arc drawings. In addition, we describe a morphing algorithm along with its implementation. Movies of sample morphs can be found at http://www.cs.arizona.edu/~mlandis/smorph.

## 1 Introduction

Morphing refers to the process of transforming one shape (the source) into another (the target). Morphing is widely used in computer graphics, animation, and modeling; see a survey by Gomes *et al.* [8]. In planar graph morphing we would like to transform a given source graph to another pre-specified target graph. A smooth transformation of one graph into another can be useful when dealing with dynamic graphs and graphs that change through time where it is crucial to preserve the mental map of the user. The mental map preservation is often accomplished by minimizing the changes to the drawing and by creating smooth transitions between consecutive drawings.

In this paper we consider the problem of morphing between two drawings, $D_s$ and $D_t$, of the same maximally planar graph $G = (V, E)$ on the sphere, where maximally planar graphs (or fully-triangulated graphs) are planar graphs in which every face is a triangle. The source drawing $D_s$ and the target drawing $D_t$ are sphere drawings (generalizations of Euclidean plane drawings to spherical space). The main objective is to find a continuous and intersection-free morph from $D_s$ to $D_t$. Note that the restriction to maximally planar graphs is not a loss of generality, as planar graphs are easily augmented to maximally planar.

### 1.1 Previous Work

Morphing has been extensively studied in graphics, animation, modeling and computational geometry, e.g., morphing 2D images [10], polygons and polylines [14], 3D objects [11] and free form curves [13].

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

Graph morphing, refers to the process of transforming a given graph $G_1$ into another graph $G_2$. Early work on this problem includes a result by Cairns in 1944 [4] who shows that if $G_1$ and $G_2$ are maximally planar graphs with the same embedding, then there exists a non-intersecting morph between them. Later, Thomassen [16] showed that if $G_1$ and $G_2$ are isomorphic convex planar graphs with the same outer face, then there exists a non-intersecting morph between them that preserves convexity. Erten *et al.* show how to morph between drawings with straight-line segments, bends, and curves [6]. This algorithm makes use of compatible triangulations [2] and the convex representation of a graph via barycentric coordinates [7,17].

While Thomassen [16] proved that an intersection-free morph exists, his approach neither provides a polynomial bound on the number of steps needed, nor yields a practical morphing algorithm. Floater and Gotsman [7] and Gotsman and Surazhsky [10,15] describe practical morphing techniques, although these approaches neither compute explicit vertex trajectories, nor guarantee a polynomial bound on the complexity of these trajectories. Recently, Lubiw *et al.* [12] developed the first algorithm for intersection-free morphing with well-behaved complexity for a special case of graphs drawings, namely, orthogonal graph drawings. This work follows an earlier result by Biedl *et al.* [3] where each edge has the same bends in the same direction in the source and target drawings.

As the sphere and the plane are topologically the same, it is natural to attempt to generalize the non-intersecting morph algorithm from Euclidean space to spherical space. Alfeld *et al.* [1] and Gotsman *et al.* [9] define analogues of barycentric coordinates on the sphere, for spherical Bernstein-Bézieri polynomials and for spherical mesh parameterization, respectively. However, barycentric coordinates are problematic in spherical space. One problem is that unlike on the Euclidean plane, three points on a sphere define two finite regions. A system of barycentric coordinates must distinguish between these two regions. A second problem arises from the non-linearity introduced by the sphere. The system of equations used to determine the drawing at any stage of the morph has non-unique solutions, and it is not easy to guarantee smoothness of the morph.

## 1.2 Our Results

Our approach to morphing spherical drawings focuses on affine transformations of the inscribed polytopes of the given spherical drawings. The inscribed polytope of a spherical drawing is obtained by replacing the geodesic edges by straight-line segments. We apply rotations, translations, scaling and shearing to the inscribed polytope, while projecting its endpoints onto the surface of the sphere throughout the transformations. At an intermediate stage, we use the intersection-free morphing algorithm for plane drawings together with a gnomonic projection to/from the sphere. Our approach yields a continuous and intersection-free morph for sphere drawings of maximally planar graphs, provided that the source and target drawings have convex inscribed polytopes. Note that in general, the inscribed polytope of a sphere drawing is star-shaped, though not necessarily convex. Therefore, while we do not resolve the general problem of morphing spherical drawings, we describe an approach which works for a subclass of spherical drawings and which hopefully can be used to resolve the general problem.

## 2 Background

We begin with some mathematical background about sphere drawings and spherical projections. The concept of a straight line in Euclidean space generalizes to that of a *geodesic* in Riemannian spaces, where the geodesic between two points is defined as a continuously differentiable curve of minimal length between them. Thus, geodesics in Euclidean geometry are straight lines, and in spherical geometry they are arcs of great circles. The generalization of an intersection-free straight-line drawing of a planar graph in spherical space uses geodesics instead of straight-lines.

**Definition 1.** A *sphere embedding* of a graph is a clockwise order of the neighbors for each vertex in the graph. A drawing is a drawing of an embedding if neighbors of nodes in the drawing match the order in the embedding. Note that 3-connected planar graphs in general, and maximally graphs in particular, have a unique sphere embedding, up to reflection.

**Definition 2.** A *geodesic-arc sphere drawing* of a graph is the sphere analogue of a *straight-line drawing* of a graph. The drawing is determined entirely by a mapping of the vertices of the graph onto the sphere. An edge between two nodes is drawn as the geodesic arc between them. We assume that no two nodes are antipodal, as there is no unique geodesic arc between two antipodal points.

**Definition 3.** An *intersection-free, geodesic-arc sphere drawing* of a graph is a sphere drawing of the graph in which no two edges intersect, except at a node on which they are both incident. We refer to such drawings as *sphere drawings* for short. Note that sphere drawings are a generalization of straight-line plane drawings from Euclidean space to spherical space.

**Definition 4.** Given a sphere drawing $D$ of a planar graph $G$, the *inscribed polytope* $P$ of $D$ is obtained by replacing the (geodesic) edges in the spherical drawing by straight-line segments. The inscribed polytope $P$ is by definition simple and star-shaped, though not necessarily convex.

**Definition 5.** The *gnomonic projection* is a non-conformal map projection obtained by projecting a point on the surface of the sphere from the sphere's center to the point in a plane that is tangent to the south pole. Since this projection sends antipodal points to the same point on the plane, it can only be used to project one hemisphere at a time. In a gnomonic projection, geodesics are mapped to straight lines and vice versa [5].

Note that a stereographic projection from the sphere to the plane, with the north pole as the focus of the projection, unambiguously maps each point from

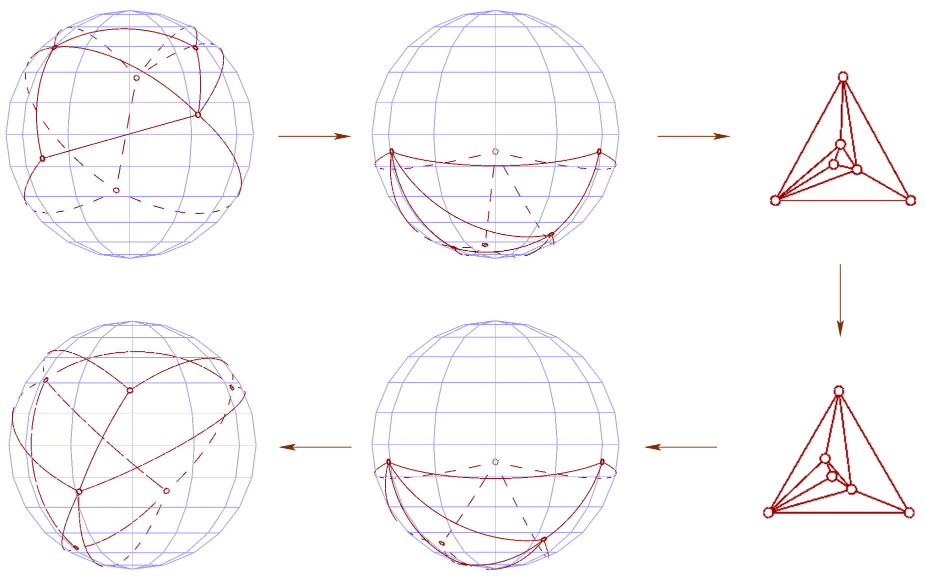


**Fig. 1.** Screenshots from our implementation, illustrating the morphing sequence: $D_s \rightarrow D'_s \rightarrow D''_s \rightarrow D''_t \rightarrow D'_t \rightarrow D_t$

the sphere to a point on the plane. In this case, however, a sphere drawing is mapped to an intersection-free drawing of the graph in the plane but that drawing is not a straight-line one. As the graph morphing algorithm for plane drawings assumes edges are straight-line segments, we use a gnomonic projection.

## 3 Morphing Between Sphere Drawings

The algorithm for morphing between two sphere drawings $D_s$ and $D_t$ of the same underlying graph $G$ can be broken into several stages:

1. Choose an outer face $f_0$ of the underlying graph;
2. Morph the source sphere drawing $D_s$ of $G$ into $D'_s$, where $D'_s$ is a sphere drawing of $G$ such that the north pole is inside $f_0$ and the entire drawing is below the equator;
3. Morph the target sphere drawing $D_t$ of $G$ into $D'_t$, where $D'_t$ is a sphere drawing of $G$ such that the north pole is inside $f_0$ and the entire drawing is below the equator;
4. Project $D'_s$ and $D'_t$ using a gnomonic projection onto the plane tangent to the south pole to the drawings $D''_s$ and $D''_t$;
5. Morph $D''_s$ into $D''_t$ using the morphing algorithm for plane drawings [6].

In practice, step 3 of the above algorithm is used in the reverse direction and altogether, the morphing sequence is: $D_s \rightarrow D'_s \rightarrow D''_s \rightarrow D''_t \rightarrow D'_t \rightarrow D_t$; see

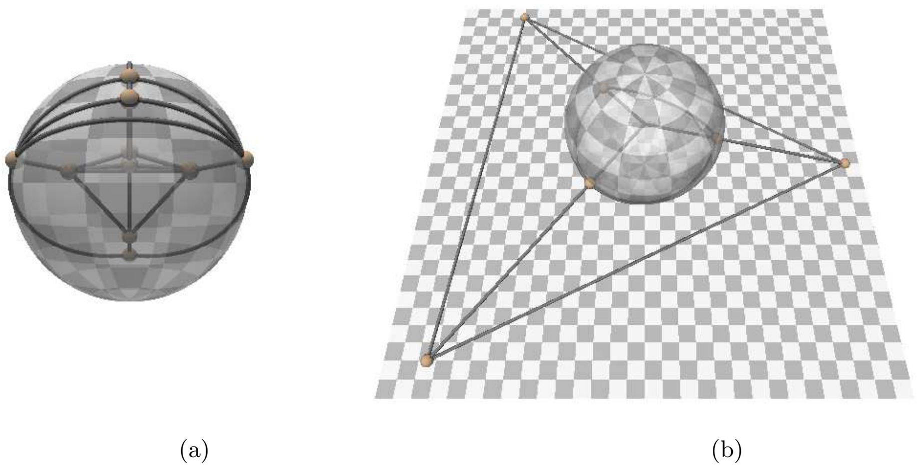


**Fig. 2.** (a) Projecting from a polytope that contains the origin to the surface of the sphere; (b) Gnomonic projection to and from the sphere

Fig. 1. By the definition of a gnomonic projection, since $D'_s$ and $D'_t$ are both strictly in the lower hemisphere, their projections $D''_s$ and $D''_t$ onto the plane tangent to the south pole are plane drawings. This implies the correctness of steps 4 and 5 and so, to argue the correctness of the overall approach, we must show that steps 2 and 3 of the algorithm above can be accomplished without introducing crossings in the morph.

## 3.1 Maintaining a Smooth and Intersection-Free Morph

Our approach to morphing sphere drawings uses a series of affine transformations to the inscribed polytope of the underlying graph (steps 2 and 3). We also rely on the barycentric morphing approach for plane drawings (steps 4 and 5). Thus, throughout the morph of our sphere drawing, we often track two positions for each vertex: the actual position of the vertex on the sphere in the sphere drawing, and the other, in some other construct, such as a 3D polytope, as in Fig. 2(a), or a plane drawing, as in Fig. 2(b). When transformations to the construct are applied, the positions of the vertices on the sphere change appropriately. A useful visualization for this approach is to imagine a spoke for each vertex, going from the origin of the sphere through both positions associated with that node. As one position changes, so does the other. For simplicity, assume the sphere is centered at $(0, 0, 0)$ with radius 1 and that the projection plane is $z = -1$.

Our first results deal with projecting a polytope on the surface of a sphere and the effect of affine transformations on the polytope to its projection.

**Theorem 1.** *A strictly convex polytope containing the center of a sphere yields a sphere drawing of that polytope's skeletal graph when its vertices are normalized to lie on the sphere.*

*Proof Sketch:* First, note that the geodesic arc between two vertices on the sphere is the same as the projection of the straight line between those two vertices of the polytope. Suppose that the projection of the polytope onto the sphere has a crossing. Consider the point $p$ on the sphere where two edges intersect. This point must be the projection of two different polytope edges onto the sphere. This implies that there exists a ray that starts at the center and intersects two separate edges of the polytope. Let $p_1$ and $p_2$ be the two points obtained from the intersection of each of these edges with the ray through the origin. Without loss of generality, let $p_1$ be the point that is further from the center. Then there exists a line segment from the center of the sphere to $p_1$ that passes through $p_2$. This contradicts the assumption that the polytope is strictly convex. Hence, the resulting sphere drawing must be intersection-free. □

Affine transformations of a convex polytope result in a convex polytope [5]. This observation, together with Theorem 1 yields the following Theorem:

**Theorem 2.** *Affine transformations to a convex polytope $P$ that contains the center of a sphere, result in sphere drawings of that polytope's skeletal graph when its vertices are normalized to lie on the sphere, if the origin remains inside $P$ throughout the transformation.*

As we are not assuming that the inscribed polytope obtained from a sphere drawing contains the origin, and we propose to deal with sphere drawings strictly contained in the lower hemisphere, we need an analogous theorem dealing with polytopes not containing the origin.

**Theorem 3.** *A strictly convex polytope $P$ not containing the center of a sphere yields a sphere drawing of that polytope's skeletal graph when its vertices are normalized to lie on the sphere if, for some face $f_1$, the ray from the origin to any point on the polytope intersects $f_1$ before any other part of the polytope, and none of the faces of $P$ lie in planes containing the origin.*

*Proof Sketch:* The face $f_1$ acts as a shield for rays emanating from the origin. Given a point $p$ of the polytope we can determine its projection $p'$ on the surface of the sphere by taking the intersection of the ray from $(0, 0, 0)$ through $p$ with the sphere. As in Theorem 1, we may have a crossing in the spherical drawing if the ray passes through more than one edge of $P$. Since $P$ is convex, the ray intersects $P$'s faces in at most two places. If the ray hits fewer than two faces, then clearly it is not going to intersect two edges.

Consider the cases where the ray enters $P$ through face $f_1$ (by assumption, it must) and exits through some other point. If the ray does not intersect an edge of $f_1$ at its entry point, then the only place at which it can intersect an edge of $P$ is its exit point. There can thus be at most one intersection, and we cannot have a crossing from this. If the ray hits an edge of $f_1$ at its entry point then, since $f_1$ shields all other faces from the origin, for the ray to hit another edge at its exit point, there would have to be a face adjacent to $f_1$ lying in a plane containing the origin (such that the ray would pass through the edge of $f_1$ and

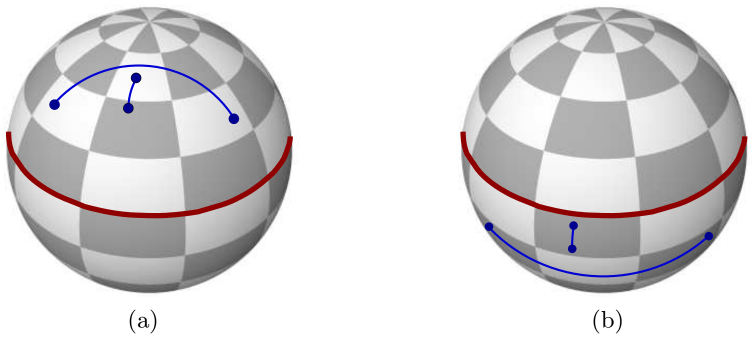


**Fig. 3.** Linear scaling of the vertices to the southern hemisphere may introduce crossings: (a) the endpoints of the long edge are below those of the short edge; (b) linear scaling could bring all the vertices to the southern hemisphere but at some intermediate stage the two edges intersect

remain in the adjacent face until it exits $P$ through another of that face's edges). This case is disallowed by another of the theorem's assumptions. □

## 3.2 Sliding Sphere Drawings to the Equator

The obvious method of "sliding" a sphere drawing down to the lower hemisphere is to do a simple linear scale of the drawing, either by z-coordinates in Euclidean coordinates, or by $\phi$ in spherical coordinates. This approach, however, does not always work. It is easy to construct an example with two non-intersecting geodesics in the upper hemisphere that must cross on their way to the lower hemisphere if linear scaling is used; see Fig. 3. Therefore, we consider the approach where we manipulate the inscribed polytope.

**Theorem 4.** *There exists a continuous and intersection-free morph that moves a sphere drawing $D$, of a maximally planar graph $G$, to a drawing of $G$ such that the vertices of a chosen face $f_0$ are on the equator and all others are strictly below the equator, provided that the inscribed polytope $P$ of $D$ is convex.*

*Proof Sketch:* Consider the inscribed convex polytope $P$ corresponding to the sphere drawing $D$. We have two cases: either $P$ contains the origin or it does not.

**Case 1 ($P$ contains the origin):** First rotate $P$ so that the outward normal to $f_0$ is parallel to $(0, 0, 1)$. Let $v_0$ be the average of the points of $f_0$. Since $P$ is convex, the segment between the origin and $v_0$ lies entirely within $P$. We can thus apply to $P$ a translation along the vector $-v_0$ and be assured that $P$ contains the origin throughout the transformation, hence Theorem 1 applies. Now $f_0$ lies within the $xy$-plane, so when we project its points onto the sphere, they lie on the equator. Since $P$ is convex, we know all other points of $P$ are on one side of $f_0$. Since the outward normal of $f_0$ is pointing up, the other points are then below $f_0$, and hence below the equator.

**Case 2 ($P$ does not contain the origin):** Here we rely on Theorem 3, instead. First we need to show that its preconditions are true: that there exists some face $f_1$ that acts as an shield that eclipses the rest of the polytope from the origin, and no faces lie in planes containing the origin.

Since $P$ does not contain the origin, there exists some plane that passes through the origin such that $P$ lies entirely on one side of that plane. Thus $D$ has one face, which we conveniently call $f_1$, which encompasses a half-sphere. The face $f_1$ must eclipse the rest of $P$ from the origin. The edges in $D$ that make up $f_1$ match the edge of the spherical region eclipsed by $f_1$ in $P$. Since $f_1$ is the outermost face, there can be no nodes outside of this region.

The second condition is straightforward: the only way three points on a sphere can lie on a plane containing the origin is if they all lie on a great circle (a circle whose center is the same as the sphere's). A face made by three such points is either defined by a great circle, in which case the face itself, and hence $P$, contains the origin (so we would have already dealt with it by case 1), or the three points all lie within a half-circle, in which case the arc between the two most distant completely overlaps the arcs between the other two pairs, in which case we did not have a valid sphere drawing to begin with.

We have shown Theorem 3 applies, and can thus apply any affine transformations to $P$ that maintains $f_1$'s eclipse of the rest of the polytope. We use shearing, as it is an affine transformation and straight lines remain straight. If the application of a transformation were to negate $f_1$'s "eclipse" property, then it would have to introduce a clear path from the origin to some edge in $P$ not on $f_1$.

Shearing and rotation do not affect the origin, so we can apply these transformations (around the origin, anyway) while maintaining a valid sphere drawing in the projection. Let $v_0$ be the centroid of $f_1$. We rotate $P$ so that $v_0$ lies in the $xy$-plane on the line $y = x$. Now $v_0$ lies at $(a, a, 0)$, for some $a$. Simultaneously shear $P$ in $x$ and $y$ with the factor $-1$, so that $v_0$ ends up at the origin. We now have a convex polyhedron that contains the origin, and we have reduced the problem to case 1. $\square$

## 3.3 Sliding Sphere Drawings to the Lower Hemisphere

From Theorem 4 we know that we can transform $D$ into a drawing such that the vertices of a face $f_0$ are on the equator and all the rest are strictly below the equator. At this stage it is easy to argue that there exists an $\epsilon > 0$ such that we can translate the polytope by an additional $\epsilon$ vertically down, so that all the points on the sphere (including those that form $f_0$) are strictly below the equator.

In practice, however, the valid values of $\epsilon$ can be arbitrarily small, making this simple approach unattractive for morphing. The value of $\epsilon$ depends on the placement of the vertices of $f_0$ around the equator. If two vertices of $f_0$ are near-antipodal, then the edge between them can pass arbitrarily close to the south pole when we translate $P$ strictly below the equator. This would make it difficult to prevent crossings in the spherical drawing. Instead, we use scaling and shearing (both affine transformations) of the polytope $P$ to make $f_0$ an equilateral triangle. We consider $f_0$ by itself in the plane, calculate the transformations necessary to make it equilateral (shear around its centroid until it is isosceles, and then scale to make it equilateral), and apply them to $P$ as a whole.

Our goal is to move all vertices outside of $f_0$ low enough on the sphere so that we can guarantee $f_0$ blocks their view of the origin. As we show below, it suffices to move the rest of the points below the Antarctic circle ($66^o$S, $z \approx -0.9135$) to ensure that they are eclipsed by an $f_0$ whose vertices lie on the Tropic of Capricorn ($23.5^o$S, $z \approx -0.3987$). These two values also provide a bound on the area of the straight-line plane drawing obtained as the gnomonic projection of the sphere drawing. With the next theorem we derive the general relation that must exist between these two latitudes in order to guarantee we obtain an intersection-free sphere drawing, as per Theorem 3, and it is straight-forward to verify that that these two values satisfy the relation.

**Theorem 5.** *There exists a continuous and intersection-free morph that moves a sphere drawing $D$, of a maximally planar graph $G$, to a drawing of $G$ such that all the vertices are strictly below the equator, provided that the inscribed polytope $P$ of $D$ is convex.*

*Proof Sketch:* Here is the outline of the proof. We begin with $f_0$ as a triangle in the $xy$-plane. We apply scaling and shearing to $P$ to transform $f_0$ into an equilateral triangle. We choose a value $z_1$ that we want to translate $f_0$ down to, and calculate a scaling factor $s$ as a function of $z_1$ and $z_3$, the highest $z$-coordinate of any point outside $f_0$. We scale $P$ in $x$ and $y$ by a factor of $\frac{1}{s}$, and project it back onto the sphere. Note that this leaves $f_0$ in the $xy$-plane. The scaling factor was computed so that when we translate $P$ down by $z_1$ the face $f_0$ eclipses the rest of $P$, yielding a valid sphere drawing at each stage by Theorem 3. Since $f_0$ is now strictly below the equator, and all other nodes are below $f_0$, the entire drawing is below the equator. Next we provide some of the details about this argument.

We begin where Theorem 4 left off. The inscribed polytope $P$ has the designated face $f_0$ on the equator and all other vertices in the southern hemisphere. We skip the details about scaling and shearing to $P$ to transform $f_0$ into an equilateral triangle, and focus on calculating the scaling factor $s$ needed to ensure that when we translate $P$ below the equator, the spherical drawing contains no crossings.

Since we have transformed $f_0$ into an equilateral triangle, we know exactly where its arcs lie, and can calculate the lowest point on the sphere covered by $f_0$. We would like to translate $P$ down so that $f_0$ lies in the plane $z = z_1$ (say, the Tropic of Capricorn). Rotate $P$ so that one of $f_0$'s vertices lies on the $y$-axis. Then the coordinates of that point are $(0, \sqrt{1 - z_1^2}, z_1)$. Since $f_0$ is equilateral, we can easily find that its other two points are at $(\frac{\sqrt{3}y_1}{2}, \frac{-y_1}{2}, z_1)$ and $(\frac{-\sqrt{3}y_1}{2}, \frac{-y_1}{2}, z_1)$. Since these two are symmetric around the $y$-axis, we can use the arc between these to find the lowest point of $f_0$ on the sphere. The midpoint of the spherical arc is the projection of the midpoint of the Euclidean line between these two points, given by the average of the two points:

$$m = (0, \frac{-y_1}{2}, z_1) = (0, \frac{-\sqrt{1-z_1^2}}{2}, z_1)$$

We need its magnitude to project it onto the sphere:

$$||m|| = \sqrt{\frac{-\sqrt{(1-z_1^2)}}{2}^2 + z_1^2} = \sqrt{\frac{1-z_1^2}{4} + z_1^2} = \sqrt{\frac{1}{4} + \frac{3}{4}z_1^2} = \frac{1}{2}\sqrt{3z_1^2+1}$$

The midpoint $m$ had a $z$-coordinate of $z_1$ and so, when projected onto the sphere, it has a $z$-coordinate of $\frac{z_1}{||m||}$. Thus, the lowest point $z_2$ of $f_0$ on the sphere would be

$$z_2 = \frac{z_1}{||m||} = \frac{2z_1}{\sqrt{3z_1^2+1}}.$$

If we move all points of $D$ not in $f_0$ below $z_2$, then we can translate $P$ down and guarantee that $f_0$ still eclipses $P$ from the origin, and thus maintain a valid sphere drawing throughout. Using the Tropic of Capricorn for $z_1$ yields a value for $z_2$ that is above the Arctic Circle, so using the two familiar latitudes guarantees valid sphere drawings throughout. To make sure all vertices outside $f_0$ are below $z_2$, we scale $P$ down around the $z$-axis by some constant factor $s$. This scaling has the effect of moving all the vertices not in $f_0$ towards the south pole. We can calculate the scale-factor $s$ necessary to move all nodes below $z_2$ as follows.

Let $z_3$ be the maximum $z$-coordinate of any node in $D$ not in $f_0$. We would like to scale the point $(x, y, z_3)$ to $(\frac{x}{s}, \frac{y}{s}, z_3)$, such that when it is projected back onto the sphere, its $z$-coordinate is below $z_2$. To project $(\frac{x}{s}, \frac{y}{s}, z_3)$ onto the sphere we first find its magnitude. Since the original point lies on the sphere, we have $x^2 + y^2 = 1 - z_3^2$ and the magnitude is given by:

$$\sqrt{\frac{x^2}{s^2} + \frac{y^2}{s^2} + z_3^2} = \sqrt{\frac{x^2+y^2}{s^2} + z_3^2} = \sqrt{\frac{1-z_3^2}{s^2} + z_3^2}.$$

As our goal is to have the scaled, projected point lie below $z_2$, so we need to find a value for $s$ such that: $\frac{z_3}{\sqrt{\frac{1-z_3^2}{s^2}+z_3^2}} < z_2$. Solving for $s$ gives us: $s > \sqrt{\frac{1-z_3^2}{\frac{z_2^2}{z_3^2} - z_3^2}}$.

Using the scaling factor guarantees all points outside $f_0$ fall below $f_0$'s arcs on the sphere when projected, and thus $f_0$ eclipses $P$ throughout the translation, and we can move $f_0$ on the sphere down to the plane $z = z_1$ with the translation $(0, 0, -z)$. □

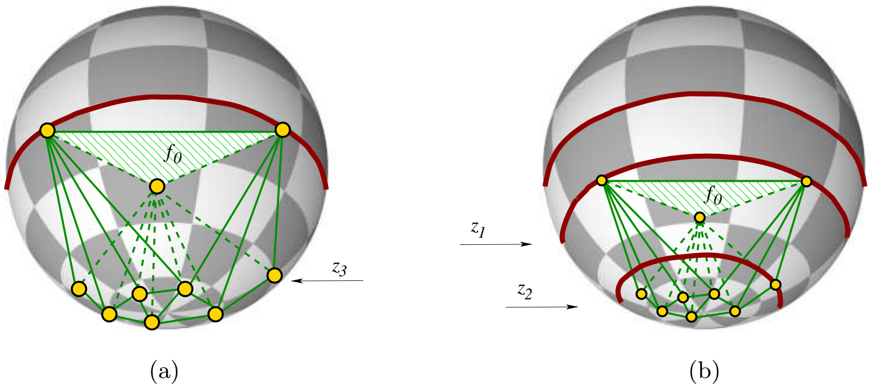


**Fig. 4.** The polytope $P$ has a face $f_0$ on the equatorial plane. The highest $z$-coordinate of a vertex not on $f_0$ is given by $z_3$. We would like to translate the polytope straight down so that $f_0$ is on the Tropic of Capricorn plane, given by $z = z_1$. We ensure that all vertices other than those in $f_0$ are below the Antarctic circle, given by plane $z = z_2$.

### 3.4 The Complete Morph

We have shown that we can morph a sphere drawing to another sphere drawing that is entirely in one hemisphere. Then, starting with the source drawing $D_s$ we can morph it to a drawing $D'_s$ that is strictly below the equator. We can do the same with the target sphere drawing $D_t$ and morph it to a sphere drawing $D'_t$ that is strictly below the equator. We then obtain the gnomonic projections $D''_s$ and $D''_t$ of the two drawings onto the plane tangent to the south pole. We then apply the planar morph algorithm to morph between these two plane drawings. Throughout the planar morph, the sphere drawing is the inverse gnomonic projection of the current state of the plane drawing. Finally, we invert the $D_t \rightarrow D'_t$ morph to arrive at the target drawing.

In order to perform the planar morph, we must ensure that the outer face in $D''_s$ and $D''_t$ is the same. We must match the upper faces in $D'_s$ and $D'_t$. Theorem 4 allows us to use whichever face we wish, therefore matching is not a problem.

## 4 Conclusions and Open Problems

We have shown that under certain conditions we can morph between spherical drawings such that the morph is continuous and intersection-free. More images and movies are available at `http://www.cs.arizona.edu/~mlandis/smorph`. Several important open problems remain:

1. Does there exist a continuous and intersection-free morph between any pair of sphere drawings of an underlying 3-connected graph?

2. In the planar morph stage, what is actually computed is not the trajectories of the vertices, but their locations at any stage in the morph. Is there a morph with trajectories of polynomial complexity?
3. Is there a more direct way to use spherical barycentric coordinates with interpolating between convex representations of graph to obtain a spherical morph, that does not involve reducing the problem to a planar morph?

## Acknowledgments

We thank Alon Efrat, Scott Drysdale and Anna Lubiw, for the stimulating discussions about this problem.

## References

1. P. Alfeld, M. Neamtu, and L. L. Schumaker. Bernstein-Bézier polynomials on circle, spheres, and sphere-like surfaces. *Computer Aided Geometric Design Journal*, 13:333–349, 1996.
2. B. Aronov, R. Seidel, and D. Souvaine. On compatible triangulations of simple polygons. *Computational Geometry: Theory and Applications*, 3:27–35, 1993.
3. T. C. Biedl, A. Lubiw, and M. J. Spriggs. Morphing planar graphs while preserving edge directions. In *13th Symposium on Graph Drawing (GD)*, pages 13–24, 2005.
4. S. S. Cairns. Deformations of plane rectilinear complexes. *American Math. Monthly*, 51:247–252, 1944.
5. H. Coxeter. *Introduction to Geometry*. Wiley, 1961.
6. C. Erten, S. G. Kobourov, and C. Pitta. Intersection-free morphing of planar graphs. In *11th Symposium on Graph Drawing*, pages 320–331, 2003.
7. M. Floater and C. Gotsman. How to morph tilings injectively. *Journal of Computational and Applied Mathematics*, 101:117–129, 1999.
8. J. Gomes, L. Darsa, B. Costa, and D. M. Vello. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann, 1999.
9. C. Gotsman, X. Gu, and A. Sheffer. Fundamentals of spherical parameterization for 3D meshes. In *SIGGRAPH '03*, pages 358–363, 2003.
10. C. Gotsman and V. Surazhsky. Guaranteed intersection-free polygon morphing. *Computers and Graphics*, 25(1):67–75, Feb. 2001.
11. J. F. Hughes. Scheduled Fourier volume morphing. *Computer Graphics*, 26(2):43–46, July 1992.
12. A. Lubiw, M. Petrick, and M. Spriggs. Morphing orthogonal planar graph drawings. In *17th Symposium on Discrete Algorithms (SODA)*, pages 222–230, 2006.
13. T. Samoilov and G. Elber. Self-intersection elimination in metamorphosis of two-dimensional curves. *The Visual Computer*, 14:415–428, 1998.
14. T. W. Sederberg and E. Greenwood. A physically based approach to 2-D shape blending. In *SIGGRAPH*, pages 25–34, July 1992.
15. V. Surazhsky and C. Gotsman. Controllable morphing of compatible planar triangulations. *ACM Transactions on Graphics*, 20(4):203–231, Oct. 2001.
16. C. Thomassen. Deformations of plane graphs. *J. Combin. Theory Ser. B*, 34: 244–257, 1983.
17. W. T. Tutte. How to draw a graph. *Proc. London Math. Society*, 13(52):743–768, 1963.

# $k$-Colored Point-Set Embeddability of Outerplanar Graphs*

Emilio Di Giacomo[1], Walter Didimo[1], Giuseppe Liotta[1], Henk Meijer[2], Francesco Trotta[1], and Stephen K. Wismath[3]

[1] Dip. di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia
{digiacomo, didimo, liotta, francesco.trotta}@diei.unipg.it
[2] Department of Computing and Information Science, Queen's University
henk@cs.queensu.ca
[3] Department of Mathematics and Computer Science, University of Lethbridge
wismath@cs.uleth.ca

**Abstract.** This paper addresses the problem of designing drawing algorithms that receive as input a planar graph $G$, a partitioning of the vertices of $G$ into $k$ different semantic categories $V_0, \cdots, V_{k-1}$, and $k$ disjoint sets $S_0, \cdots, S_{k-1}$ of points in the plane with $|V_i| = |S_i|$ ($i \in \{0, \cdots, k-1\}$). The desired output is a planar drawing such that the vertices of $V_i$ are mapped onto the points of $S_i$ and such that the curve complexity of the edges (i.e. the number of bends along each edge) is kept small. Particular attention is devoted to outerplanar graphs, for which lower and upper bounds on the number of bends in the drawings are established.

## 1 Introduction

Semantic constraints for the vertices of a graph $G$ define the placement that these vertices must have in a readable visualization of $G$ [3,9,12]. For example, in the context of data base design some particularly relevant entities of an ER schema may be required to be represented in the center and/or along the boundary of the diagram (see, e.g., [13]). A possible way of modeling semantic constraints for a (sub)set $\{v_1, v_2, \cdots, v_h\}$ of the vertices of a graph $G$ is to specify a set $\{p_1, p_2, \cdots, p_h\}$ of locations for their placement. Often, it is sufficient for the application that every vertex $v_i$ ($i = 1, \cdots, h$) is placed to any location $p_j$ ($1 \leq j \leq h$), that is the mapping of each vertex to a specific location is not part of the input. A key reference in this scenario is the work by Kaufmann and Wiese [10]. Given a planar graph $G$ with $n$ vertices and a set $S$ of $n$ distinct points in the plane, they show how to compute a planar drawing of $G$ such that each vertex is mapped to any point of $S$ and every edge bends at most twice, which is proved to be worst case optimal. It is also known that for specific classes

* Research partially supported by MIUR under Project "ALGO-NEXT(Algorithms for the Next Generation Internet and Web: Methodologies, Design and Experiments)", and NSERC. Omitted or sketched proofs can be found in [4].

of graphs, such as outerplanar graphs and trees, the number of bends per edge can be reduced to zero (see, e.g., [1,2,7]). The work by Kaufmann and Wiese, however, does not seem to be immediately scalable to applications where the vertices of $G$ are grouped based on their relevance or meaning and for each of these groups a different semantic constraint should be applied (for example a subset of the vertices on the boundary, some others in a central position, and so on). A solution in this case could be to fix in advance the location of each vertex in each semantic category and then route the edges. Halton [8] and, independently, Pach and Wenger [11], showed that a planar graph $G$ always admits a planar drawing such that the location of each vertex is part of the input; however, fixing the vertex positions in advance may give rise to drawings with high visual complexity. Pach and Wenger [11] show that a linear number of bends per edge is asymptotically optimal in the worst case even for graphs as simple as paths.

This paper studies the above mentioned problem without imposing that the position of the vertices is part of the input. The input is a planar graph $G$, a partitioning of the vertices of $G$ into $k$ different semantic categories $V_0, \cdots, V_{k-1}$, and $k$ disjoint sets $S_0, \cdots, S_{k-1}$ of points in the plane with $|V_i| = |S_i|$ ($i \in \{0, \cdots, k-1\}$). The required output is a planar drawing such that the vertices of $V_i$ are mapped to the points of $S_i$; for each vertex $v \in V_i$ the drawing algorithm can choose which point of $S_i$ represents $v$. We study the visual complexity of these types of drawings, expressed in terms of number of bends per edge. The intuition is that if the number of categories is constant, then a constant number of bends per edge may be sufficient, at least for simple classes of planar graphs. This type of investigation was started in [5,6], where the apparently simple case of $k = 2$ is studied. In [6] and in [5] a constant number of bends per edge is proved to be sufficient for constructing planar poly-line drawings of subclasses of outerplanar graphs, including paths, cycles, caterpillars, and wreaths. In [5] it is also shown that there exists a 2-outerplanar graph $G$ with a vertex partition $V_0, V_1$ and two disjoint sets $S_0, S_1$ of points such that any planar drawing of $G$ that maps a vertex $v \in V_i$ to a distinct point of $S_i$ ($i = 0, 1$) has at least one edge with a linear number of bends. The 2-outerplanarity of the counterexample on one hand, and the outerplanarity of the families of graphs for which a constant number of bends per edge is possible, motivated us to further investigate how many bends are required for general outerplanar graphs and then extend the research to cases where $k > 2$.

In this paper, each integer $i \in \{0, \cdots, k-1\}$ identifying a partition set of the vertices of $G$ is called a *color*, $G$ is called a *$k$-colored graph*, and the set of points $S = S_0 \cup \cdots \cup S_{k-1}$ such that $|V_i| = |S_i|$ (for each each color $i \in \{0, \cdots, k-1\}$) is called a *$k$-colored set compatible with* $G$. A planar drawing of $G$ such that each $v \in V_i$ is drawn as a distinct point $p \in S_i$ is a *point-set embedding* of $G$ on $S$. Graph $G$ is *$k$-colored point-set embeddable* if it admits a point-set embedding on *every* $k$-colored set compatible with $G$. Our main results are as follows.

– Every outerplanar 2-colored graph is 2-colored point-set embeddable with at most 5 bends per edge. Also, a 2-colored embedding of this type can be computed in $O(n \log n)$ time. (See Section 3).

– For every positive integer $h > 0$, there exists an outerplanar 3-colored graph $G$, whose number of vertices depends on $h$, and a set of points $S$ compatible with $G$ such that every point-set embedding of $G$ on $S$ has an edge with more than $h$ bends. (See Section 4).
– For $k$ colors in which one restricts all the points of the point set with the same color to vertical regions separated by vertical lines, at most $4k + 1$ bends per edge are required for outerplanar graphs. The drawings can be computed in $O(n \log n + kn)$ time. (See Section 5).

## 2 Preliminaries

Let $G = (V, E)$ be a graph. A *k-coloring* of $G$ is a partition $\{V_0, V_1, \ldots, V_{k-1}\}$ of $V$ where the integers $0, 1, \ldots, k-1$ are called *colors*. For each vertex $v \in V_i$ $(0 \leq i \leq k-1)$ we denote by $col(v)$ the color $i$ of $v$. For any subset of vertices $U \subseteq V_i$ $(0 \leq i \leq k-1)$ we denote by $col(U)$ the color $i$ of all the elements of $U$. A $k$-coloring of $G$ is *proper* if for every edge $(u, v) \in E$ we have $col(u) \neq col(v)$. A graph $G$ with a (proper) $k$-coloring is called a *(proper) k-colored graph.* Let $S$ be a set of distinct points in the plane. For any point $p \in S$ we denote by $x(p)$ and $y(p)$ the $x$- and $y$-coordinates of $p$, respectively. A *k-coloring* of $S$ is a partition $\{S_0, S_1, \ldots, S_{k-1}\}$ of $S$. A set of points $S$ with a $k$-coloring is called a *k-colored set.* For each point $p \in S_i$ $(0 \leq i \leq k-1)$, $col(p)$ denotes the color $i$ of $p$, and for any subset $R \subseteq S_i$ $(0 \leq i \leq k-1)$, $col(R)$ denotes the color $i$ of all the elements of $R$. A $k$-colored set $S$ is *compatible with* a $k$-colored graph $G$ if $|V_i| = |S_i|$ for every $0 \leq i \leq k-1$; if $G$ is planar we say that $G$ has a *point-set embedding* on $S$ if there exists a planar drawing of $G$ such that: (i) every vertex $v$ is mapped to a distinct point $p$ of $S$ with $col(p) = col(v)$, (ii) each edge $e$ of $G$ is drawn as a polyline $\lambda$; a point shared by any two consecutive segments of $\lambda$ is called a *bend* of $e$. A *k-colored sequence* $\sigma$ is a sequence of (possibly repeated) colors $c_0, c_1, \ldots, c_{n-1}$ such that $0 \leq c_j \leq k-1$ $(0 \leq j \leq n-1)$. We say that $\sigma$ is *compatible with* a $k$-colored graph $G$ if, for every $0 \leq i \leq k-1$, color $i$ occurs $|V_i|$ times in $\sigma$. Let $S$ be a $k$-colored set. Throughout the paper we always assume that the points of $S$ have different $x$-coordinates (if not we can rotate the plane so to achieve this condition). Let $p_0, p_1, \ldots, p_{n-1}$ be the points of $S$ with $x(p_0) < x(p_1) < \ldots < x(p_{n-1})$. The $k$-colored sequence $col(p_0), col(p_1), \ldots col(p_{n-1})$ is called the *k-colored sequence induced by S*, and is denoted as $seq(S)$.

A graph $G$ is *Hamiltonian* if it has a simple cycle that contains all its vertices; such a cycle is called a *Hamiltonian cycle* of $G$. If $G$ is a $k$-colored graph and $\sigma = c_0, \ldots, c_{n-1}$ is a $k$-colored sequence compatible with $G$, a *k-colored Hamiltonian cycle of G consistent with* $\sigma$ is a Hamiltonian cycle $v_0, v_1, \ldots, v_{n-1}$ such that $col(v_i) = c_i$ $(0 \leq i \leq n-1)$. If such a cycle exists, $G$ is said to be *k-colored Hamiltonian consistent with* $\sigma$. Let $G$ be a planar $k$-colored graph and let $\sigma$ be a $k$-colored sequence compatible with $G$. It is always possible to augment $G$ with dummy edges so that the resulting (not necessarily planar) graph has a $k$-colored Hamiltonian cycle $\mathcal{H}$ consistent with $\sigma$ and including all dummy edges. Let $\Psi$

be a planar embedding of $G$ and suppose that $\Gamma$ is a drawing of $G \cup \mathcal{H}$ such that: (i) The drawing of $G$ in $\Gamma$ preserves $\Psi$; (ii) no two dummy edges of $\mathcal{H}$ cross; (iii) each edge $e$ of $G$ crosses at most $d$ times the dummy edges of $\mathcal{H}$. Each crossing between an edge $e$ of $G$ and a dummy edge of $\mathcal{H}$ is replaced in $\Gamma$ with a dummy vertex, which we call a *division vertex for* $e$, and we say that $\mathcal{H}$ is an *augmenting* $k$*-colored Hamiltonian cycle of* $G$ *consistent with* $\sigma$ *with at most* $d$ *division vertices per edge*. We also say that $\mathcal{H}$ (with at most $d$ division vertices per edge) *is constructed on* $\Psi$.

**Lemma 1.** *Let $G$ be a planar $k$-colored graph and let $\sigma$ be a $k$-colored sequence compatible with $G$. If $G$ admits a point-set embedding with at most $b \in \mathbb{N}^+$ bends per edge on every $k$-colored set $S$ such that $seq(S) = \sigma$, then $G$ admits an augmenting $k$-colored Hamiltonian cycle consistent with $\sigma$ and with at most $b-1$ division vertices per edge constructed on some planar embedding of $G$.*

*Sketch of Proof:* Let $S$ be a set of points $p_0, p_1, \ldots, p_{n-1}$ that lie on the $x$-axis such that $x(p_j) < x(p_{j+1})$ for $0 \leq j < n-1$ and $seq(S) = \sigma$. By hypothesis $G$ admits a point-set embedding on $S$ with at most $b$ bends per edge. Assume vertex $v_i$ of $G$ is placed at point $p_i$ for all $i$. For all $0 \leq i < n-1$ we add straight line edges $(v_i, v_{i+1})$ if they are not adjacent in $G$. If $v_{n-1}$ and $v_0$ are not adjacent in $G$ we connect them with a polyline as follows: from $v_{n-1}$ we draw a horizontal line segment sufficiently far to the right, from $v_0$ we draw a horizontal line segment sufficiently far to the left, and we complete the polyline by connecting the two line segments to a point sufficiently high above the drawing of $G$. It can be shown that if an edge of $G$ is intersected more than $b-1$ times, then this edge is drawn with at least $b+1$ bends. □

**Lemma 2.** *Let $G$ be a planar $k$-colored graph and let $\sigma$ be a $k$-colored sequence compatible with $G$. If $G$ has a planar embedding on which an augmenting $k$-colored Hamiltonian cycle consistent with $\sigma$ and with at most $b \in \mathbb{N}^+$ division vertices per edge can be constructed, then $G$ admits a point-set embedding with at most $2b+1$ bends per edge on every $k$-colored set $S$ such that $seq(S) = \sigma$.*

*Sketch of Proof:* The proof is similar to a technique used in [10]. Let $S$ be a $k$-colored set such that $seq(S) = \sigma$. By hypothesis it is possible to find an augmenting $k$-colored Hamiltonian cycle $\mathcal{H}$ of $G$ consistent with $\sigma$ such that each edge has at most $b$ division vertices. We add all division vertices to $G$ and color these vertices with color $k$. Let $G'$ denote the new graph, $\mathcal{H}'$ the new Hamiltonian cycle and $n'$ the number of vertices of $G'$. Add $n' - n$ elements to $S$ to create $S'$. We can do this in such a way that $\mathcal{H}'$ is consistent with $S'$. We first draw $\mathcal{H}'$ on $S'$ from left to right, and the final edge from the last point of $S'$ to the first point of $S'$ is connected to a point drawn sufficiently high. Only the last edge of $\mathcal{H}'$ has a bend. The remaining edges of $G'$ are either inside or outside $\mathcal{H}'$. The inside edges are drawn above $S'$, the others are drawn below $S'$. As is shown in [10], we can add these edges in a planar fashion by placing the bends sufficiently far up or down. This results in a drawing with at most one

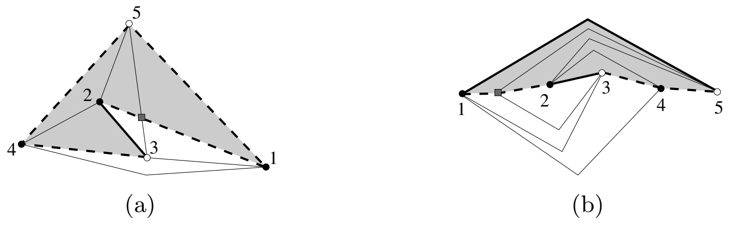


**Fig. 1.** (a) A planar 2-colored graph $G$. An augmenting 2-colored Hamiltonian cycle $\mathcal{H}$ of $G$ with at most one division vertex per edge is highlighted in bold. Dummy edges are dashed lines and dummy vertices are squares. The interior of $\mathcal{H}$ is shown as a shaded area. (b) A point-set embedding with at most 3 bends per edge. Edge $(3, 5)$ has 3 bends (one in correspondence of the dummy vertex).

bend per edge of $G'$, which implies that there is a drawing of $G$ with at most $2b + 1$ bends per edge. For an illustration, see Figure 1. □

A graph is *outerplanar* if it admits a planar embedding such that all vertices are on the external face. A graph is 2-*outerplanar* if it admits a planar embedding such that removing all vertices on the external face, results in all the remaining vertices being on the external face. The following result is proved in [5].

**Theorem 1.** [5] *For every $n \geq 4$ there exists a 2-outerplanar 2-colored graph $G$ with $2n$ vertices and a 2-colored set compatible with $G$ such that the maximum number of bends per edge of every point-set embedding of $G$ on $S$ is $\Omega(n)$.*

## 3 Outerplanar 2-Colored Graphs

Motivated by Theorem 1, in this section we prove that every outerplanar 2-colored graph $G$ admits a point-set embedding with at most 5 bends per edge on any 2-colored set $S$ compatible with $G$.

Let $\Psi$ be a planar embedding of $G$ with all vertices on the external face. We prove that $G$ admits an augmenting 2-colored Hamiltonian cycle constructed on $\Psi$, consistent with $\sigma = seq(S)$, and with at most 2 division vertices per edge. The result then follows from Lemma 2. Since every outerplanar graph can be made biconnected by adding edges while maintaining the outerplanarity, we can assume, without loss of generality, that $G$ is biconnected. In this case the boundary of the external face of $G$ is a simple cycle $\mathcal{C}$ containing all vertices of $G$. The edges of $G$ that are not in $\mathcal{C}$ are called *chords*. We start by proving that every simple cycle $\mathcal{C}$ admits an augmenting 2-colored Hamiltonian cycle $\mathcal{H}$ consistent with $\sigma$ and with at most 1 division vertex per edge. To this aim we describe an algorithm that computes $\mathcal{H}$. We then shall describe how it is possible to obtain from $\mathcal{H}$ an augmenting 2-colored Hamiltonian cycle of $G$ consistent with $\sigma$ and with at most two division vertices per edge. The idea is illustrated in Figure 2.

Let $\sigma = c_0, \ldots, c_{n-1}$. We order the vertices of $\mathcal{C}$ counterclockwise starting from an arbitrary vertex whose color is $c_0$. More formally, let $v_0$ be a vertex

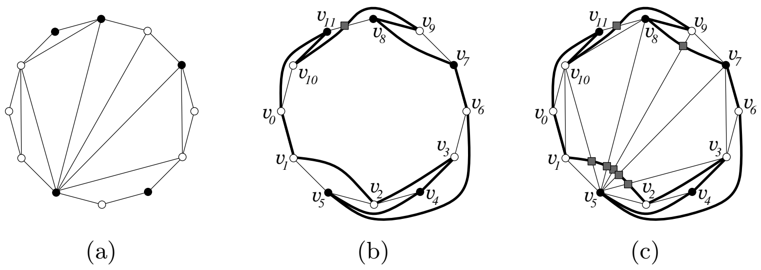


**Fig. 2.** (a) An outerplanar 2-colored graph $G$. (b) An augmenting 2-colored Hamiltonian cycle of the external boundary of $G$, with at most 1 division vertex per edge. (c) An augmenting 2-colored Hamiltonian cycle of $G$ with at most 2 division vertices per edge.

of $\mathcal{C}$ such that $col(v_0) = c_0$; we walk counterclockwise along $\mathcal{C}$ starting from $v_0$ and write $u < v$ if $u$ is encountered before $v$. Also, given a subset of vertices $U \subseteq V(\mathcal{C})$ we write $\text{first}(U)$ to denote the first vertex of $U$ according to $<$ and with $\text{last}(U)$ the last vertex of $U$ according to $<$. Given a vertex $v \in U$, $\text{next}(v)$ denotes the vertex of $U$ that is immediately after $v$ according to $<$.

The construction of $\mathcal{H}$ begins with $v_0$ and adds one vertex of $\mathcal{C}$ per step (plus possibly one division vertex). The vertex of $\mathcal{C}$ added at Step $i$ will be denoted as $v_i$. Also, we denote as $H_i$ the set of vertices added to $\mathcal{H}$ up to Step $i$ and as $G_i$ the augmented graph constructed up to Step $i$, i.e. the graph consisting of $\mathcal{C}$ plus all dummy edges and division vertices possibly added during Steps $0, 1, \ldots, i$. Also we define the following sets:

- $NB_i = \{v \mid v \in V(\mathcal{C}), v \notin H_i, \text{first}(H_i) < v < \text{last}(H_i)\}$
- $F_i^c = \{v \mid v \in V(\mathcal{C}), \text{last}(H_i) < v, col(v) = c\}$, where $c = 0, 1$ (notice that $F_i^c \cap H_i = \emptyset$)

Sets $NB_i$ and $F_i^c$ partition the set of vertices of $\mathcal{C}$ that must be still added to $\mathcal{H}$ at the end of Step $i$. Intuitively, the vertices in $NB_i$ have been already encountered moving counterclockwise on $\mathcal{C}$, while the vertices in $F_i^c$ have yet to be encountered. At the end of Step $i$, the following invariants are maintained:

**Invariant 1.** *All vertices of $NB_i$ have the same color.*

**Invariant 2.** *All vertices of $NB_i$ are on the external face of $G_i$.*

**Invariant 3.** *Vertex $v_i$ is on the external face of $G_i$.*

**Invariant 4.** *If $v_i \neq \text{last}(H_i)$, then for each vertex $u$ such that $v_i < u < \text{last}(H_i)$, we have $u \in H_i$.*

At Step $i + 1$ the algorithm chooses the vertex $v_{i+1}$ of $\mathcal{C}$ to be added to $\mathcal{H}$; the addition of $v_{i+1}$ may require us to add to $\mathcal{H}$ some dummy edges and one division vertex. We say that a dummy edge is added *inside* $\mathcal{C}$ if it is inserted on the left-hand side when walking counterclockwise around $\mathcal{C}$. In order to choose and add $v_{i+1}$, the algorithm distinguishes between the following cases:

**Case 1:** $NB_i \neq \emptyset$ **and** $col(NB_i) = c_{i+1}$**.** The algorithm chooses $v_{i+1} = \text{last}(NB_i)$. If $v_i$ and $v_{i+1}$ are not adjacent in $\mathcal{C}$, a dummy edge $(v_i, v_{i+1})$ is added on the external face of $G_i$ (see, e.g., the addition of vertices $v_4$ and $v_5$ in Figure 2(b)).

**Case 2:** $NB_i = \emptyset$ **or** $col(NB_i) \neq c_{i+1}$**.** The algorithm chooses $v_{i+1} = \text{first}(F_i^{c_{i+1}})$. Vertices $v_i$ and $v_{i+1}$ are connected in $\mathcal{H}$ according to the following sub-cases:

- **Case 2.a:** $v_i = \text{last}(H_i)$**.** If $v_{i+1} = \text{next}(\text{last}(H_i))$, then $v_i$ and $v_{i+1}$ are adjacent in $\mathcal{C}$ and therefore no dummy edge needs to be added (see, e.g., the addition of vertex $v_7$ in Figure 2(b)). If $v_{i+1} \neq \text{next}(\text{last}(H_i))$, a dummy edge $(v_i, v_{i+1})$ is added inside $\mathcal{C}$ (see, e.g., the addition of vertex $v_8$ in Figure 2(b)).
- **Case 2.b:** $v_i \neq \text{last}(H_i)$ **and** $v_{i+1} = \text{next}(\text{last}(H_i))$**.** A dummy edge $(v_i, v_{i+1})$ is added on the external face of $G_i$ (see, e.g., the addition of vertex $v_6$ in Figure 2(b)).
- **Case 2.c:** $v_i \neq \text{last}(H_i)$ **and** $v_{i+1} \neq \text{next}(\text{last}(H_i))$**.** The algorithm splits edge $(\text{last}(H_i), \text{next}(\text{last}(H_i)))$ by means of a division vertex $d$, which is added to $\mathcal{H}$ between $v_i$ and $v_{i+1}$. A dummy edge $(v_i, d)$ is added on the external face of $G_i$ and a dummy edge $(d, v_{i+1})$ is added inside $\mathcal{C}$. For example, in Figure 2(b), the addition of vertex $v_{10}$ is done by inserting a division vertex $d$ that splits $(v_8, v_{11})$, and the two dummy edges $(v_9, d)$, $(d, v_{10})$.

**Lemma 3.** *Let $\mathcal{C}$ be a 2-colored simple cycle and let $\sigma$ be a 2-colored sequence compatible with $\mathcal{C}$. Then $\mathcal{C}$ admits an augmenting 2-colored Hamiltonian cycle consistent with $\sigma$ and with at most 1 division vertex per edge.*

**Lemma 4.** *Let $G$ be an outerplanar 2-colored graph and let $\Psi$ be a planar embedding of $G$ having all vertices on the external face. Let $\sigma$ be a 2-colored sequence compatible with $G$. Then $G$ admits an augmenting 2-colored Hamiltonian cycle constructed on $\Psi$, consistent with $\sigma$, and with at most 2 division vertices per edge.*

*Proof.* Since every outerplanar graph can be made biconnected by adding edges while maintaining outerplanarity, we assume that $G$ is biconnected. Let $\mathcal{C}$ be the boundary of the external face of $G$ in $\Psi$. We remove all the chords from $G$ and compute an augmenting 2-colored Hamiltonian cycle of $\mathcal{C}$ consistent with $\sigma$ and with at most one division vertex per edge, by using the algorithm described above. If we add back the chords of $G$ to the graph $G_{n-1}$, i.e. the augmented graph constructed at the end of the algorithm, these edges will cross the edges of $\mathcal{H}$ that are inside $\mathcal{C}$. For each crossing between an edge $e_{\mathcal{H}}$ of $\mathcal{H}$ and a chord $e_{ch}$ of $G$ we add a division vertex $d$ that splits both $e_{\mathcal{H}}$ and $e_{ch}$. Thus we obtain an augmenting 2-colored Hamiltonian cycle of $G$. To complete the proof we must show that every chord $e_{ch}$ is split by at most two division vertices. To this aim observe that an edge $e_{\mathcal{H}} = (u, w)$ of $\mathcal{H}$ must cross $e_{ch}$ only if an endvertex $v$ of $e_{ch}$ is such that $u < v < w$. We show that for each vertex $v$ there can be at most one edge $e_{\mathcal{H}} = (u, w)$ of $\mathcal{H}$ such that $u < v < w$. Since edge $e_{\mathcal{H}}$ is inside the cycle $\mathcal{C}$, it is a dummy edge added at some Step $i$ $(0 \leq i \leq n-1)$ when

Cases 2.a or 2.c apply. After the addition of this edge, vertex $v$ and all the other vertices between $v$ and $w$ become vertices of $NB_i$. According to the algorithm the vertices of $NB_i$ are added to $\mathcal{H}$ by means of edges that are either edges of $\mathcal{C}$ or dummy edges on the external face of $G_i$ (Case 1). This implies that any other dummy edge incident on a vertex between $v$ and $w$ is not inside $\mathcal{C}$ and therefore edge $e_{\mathcal{H}} = (u, w)$ is the only edge of $\mathcal{H}$ such that $u < v < w$. Since each chord $e_{ch}$ has two endvertices, there is at most one division vertex on $e_{ch}$ for each of them and therefore at most two division vertices for each chord. □

**Theorem 2.** *Every outerplanar 2-colored graph $G$ admits a point-set embedding with at most 5 bends per edge on any 2-colored set compatible with $G$. Such a point-set embedding can be computed in $O(n \log n)$ time, where $n$ is the number of vertices of $G$.*

## 4 Outerplanar 3-Colored Graphs

Since, by Theorem 2, outerplanar 2-colored graphs are 2-colored point-set embeddable with $O(1)$ bends per edge, one may wonder whether the number of bends per edge remains constant for values of $k$ larger than 2. Unfortunately, this may not be the case even for 3 colors.

In this section we describe an infinite family of outerplanar 3-colored graphs such that for any integer $h \geq 0$ there exists a graph in the family and a set of points $S$ such that any point-set embedding of the graph on $S$ contains an edge having more than $h$ bends. Our family of outerplanar 3-colored graphs is parametric with $n$; every member of this family, denoted as $G_n$, has $3n$ vertices and is defined as follows. $G_n$ consists of a simple cycle formed by $n$ vertices of color 0, followed (in the counterclockwise order) by $n$ vertices of color 1, followed by $n$ vertices of color 2 (notice that $G_n$ actually has $3n$ vertices). The vertex of color 1 adjacent in the cycle to a vertex of color 0 is denoted as $v_1$; the vertex of color 2 adjacent in the cycle to a vertex of color 1 is denoted as $v_2$; the vertex of color 0 adjacent in the cycle to a vertex of color 2 is denoted as $v_0$. Also, in $G_n$ every vertex colored $i$ is adjacent to $v_i$ $(i = 0, 1, 2)$ and vertices $v_0, v_1, v_2$ form a 3-cycle. Figure 3(a) is an example of $G_n$ for $n = 12$.

To prove our lower bound, we consider all possible planar embeddings of $G_n$; for each planar embedding $\Psi$ of $G_n$ we will prove that every augmenting $k$-colored Hamiltonian cycle constructed on $\Psi$ and consistent with $seq(S)$ has more than $h$ division vertices for some edge of $G_n$.

For a planar embedding $\Psi$ of $G_n$ and a cycle $C \in G_n$ we say that $C$ *separates* a subset $V'$ from a subset $V''$ of the vertices of $G_n$ if all vertices of $V'$ lie in the interior of the region bounded by $C$ and all vertices of $V''$ are in the exterior of this region.

**Lemma 5.** *Let $h > 0$ be a positive integer and let $G_n$ be such that $n = (h+1)(25h^2+2)+25h^2$. Let $\Psi$ be a planar embedding of $G_n$ and let $C$ be the cycle $v_0, v_1, v_2$. If $C$ does not separate any two vertices, then at least one of the following conditions holds for $\Psi$:*

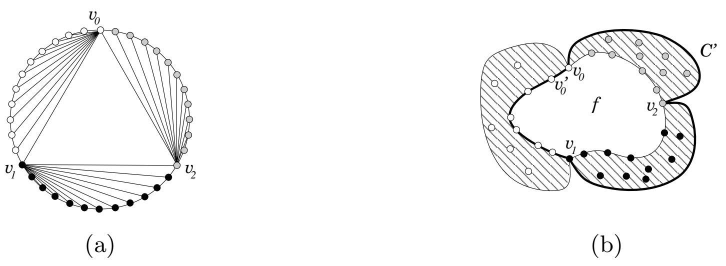


**Fig. 3.** (a)Graph $G_{12}$. (b) An illustration for the proof of Lemma 5. Cycle $C'$ is highlighted in bold.

**C.1** *There exists a face $f$ in $\Psi$ having at least $25h^2$ vertices of each color and containing vertices $v_0$, $v_1$, and $v_2$.*

**C.2** *There exists a cycle $C'$ containing vertices $v_0$, $v_1$, and $v_2$ and such that: (i) $C'$ has at most $25h^2 + 2$ edges, (ii) at least $(h + 1)(25h^2 + 2)$ vertices of a color $i$ $(i = 0, 1, 2)$ are inside $C'$, (iii) every vertex of $V - \{v_0, v_1, v_2\}$ with color different from $i$ is outside $C'$.*

*Sketch of Proof:* We first observe that in every planar embedding of $G_n$ there are exactly two faces whose boundary contains vertices of the three colors. If $C$ does not separate any two vertices, one of these two faces has $C$ as its boundary. Let $f$ be the face of $\Psi$ containing vertices of the three colors whose boundary is not $C$. For each color $i$ in $G_n$, exactly two vertices of color $i$ are adjacent to vertices of different color and each vertex of color $i$ except $v_i$ has degree three and is adjacent to two vertices of the same color; this implies that all vertices of $f$ having color $i$ are consecutive along the boundary of $f$ $(i = 0, 1, 2)$.

If Condition **C.1** does not hold, then the number of vertices of at least one color, say 0, along the boundary of $f$ is $k < 25h^2$. Let $v_0'$ be the vertex of $f$ having color 0 and adjacent to $v_0$; if $v_0$ is the only vertex of $f$ having color 0, let $v_0' = v_1$. Refer to Figure 3(b). Consider the cycle $C' = v_0, v_0', \pi(v_0', v_1), v_1, v_2$, where $\pi(v_0', v_1)$ is the path along the boundary of $f$ that goes from $v_0'$ to $v_1$. Cycle $C'$ has at most $25h^2 + 2$ vertices (and hence edges). By using the argument that $C$ does not separate any two vertices, it follows that $C'$ separates at least $n - 25h^2 = (h + 1)(25h^2 + 2)$ vertices of color 0 from all vertices of color 1 and 2 distinct from $v_1$ and $v_2$. □

**Theorem 3.** *For every $h > 0$, there exists an outerplanar 3-colored graph $G$ with $3 \cdot ((h + 1)(25h^2 + 2) + 25h^2)$ vertices and a set of points $S$ compatible with $G$, such that every point-set embedding of $G$ on $S$ has an edge with more than $h$ bends.*

*Sketch of Proof:* Let $n$ be $((h + 1)(25h^2 + 2) + 25h^2)$ and let $S$ be any set of points compatible with $G_n$ such that $seq(S)$ is an alternating sequence of the 3 colors (i.e., $\sigma = 0, 1, 2, 0, 1, 2, \ldots, 0, 1, 2$). We denote $seq(S)$ as $\sigma$.

We prove that the statement is true for $G_n$ and $S$. To this aim, based on Lemma 1, it suffices to show that, for each planar embedding $\Psi$ of $G_n$, every augmenting 3-colored Hamiltonian cycle constructed on $\Psi$ and consistent with $\sigma$ has more than $h$ division vertices for some edge of $G_n$. Let $\mathcal{H}$ be one such augmenting 3-colored Hamiltonian cycle and let $C$ be the cycle $v_0, v_1, v_2$. Note that in any planar embedding of $G_n$ all vertices having a same color are either inside or outside the region bounded by $C$; hence if $C$ separates any two vertices $u$, $v$, it separates all vertices with the same color as $u$ from all vertices with the same color as $v$ (except $v_0$, $v_1$, and $v_2$). We consider two cases.

**Case 1.** $\Psi$ is such that $C$ separates all vertices of a color $i$, except $v_i$, from all vertices of a different color $j$, except $v_j$.

**Case 2.** $\Psi$ is such that $C$ does not separate any two vertices of $G_n$. Without loss of generality, assume that all vertices of $G_n$ are on or outside $C$.

In Case 1, since $\mathcal{H}$ is consistent with $\sigma$, each vertex of color $i$ is adjacent in $\mathcal{H}$ to a vertex of color $j$. Therefore there are $n-1$ edges of $\mathcal{H}$ that cross $C$ in $\Psi$; it follows that one edge of $C$ has at least $\frac{n-1}{3} > h$ division vertices.

In Case 2, either Condition **C.1** or Condition **C.2** of Lemma 5 holds for $\Psi$. We sketch here the proof when Condition **C.2** holds. The proof for the remaining case is omitted for reasons of space. If Condition **C.2** holds, denote by $U_i$ the set of vertices of color $i$ inside the cycle $C'$ in the statement of Lemma 5. Since $\mathcal{H}$ is consistent with $\sigma$ then each vertex of $U_i$ is adjacent to vertices with color different from $i$. Thus at least $(h+1)(25h^2+2)$ edges cross cycle $C'$. Since $C'$ has at most $25h^2+2$ edges, then at least one edge of $C'$ has more than $h$ division vertices. □

## 5 Outerplanar $k$-Colored Graphs

In contrast with the result of Theorem 3, we prove that given any outerplanar $k$-colored graph $G$ (for any constant $k > 2$), there exist infinite $k$-colored sets compatible with $G$ for which a point-set embedding of $G$ with a constant number of bends per edge is possible.

Let $S$ be a $k$-colored set such that no two points have the same $x$-coordinate; assume that the points are ordered by increasing $x$-values. $S$ is an *ordered $k$-colored set* if, for every color $i$ ($0 \leq i \leq k-1$), all points of color $i$ appear consecutively in the ordering (that is, the colors never alternate in the ordering). The sequence of colors induced by an ordered $k$-colored set is called an *ordered $k$-colored sequence.*

We first describe an algorithm that, given a $k$-colored simple cycle $\mathcal{C}$ and a $k$-colored sequence $\sigma$ compatible with $\mathcal{C}$, computes an augmenting $k$-colored Hamiltonian cycle $\mathcal{H}$ of $\mathcal{C}$ consistent with $\sigma$ and with at most $k$ division vertices. We then use this algorithm to obtain an augmenting $k$-colored Hamiltonian cycle of an outerplanar $k$-colored graph $G$ consistent with $\sigma$ and with at most $2k$ division vertices. Also in this case, the augmenting $k$-colored Hamiltonian cycle of $G$ is constructed on a planar embedding of $G$ having all vertices on the external face.

Let $\mathcal{C}$ be a $k$-colored simple cycle and let $\sigma = c_0, \ldots, c_{n-1}$ be an ordered $k$-colored sequence consistent with $\mathcal{C}$. The algorithm computes $\mathcal{H}$ by performing at most $k$ rounds. At each round, it walks along the cycle $\mathcal{C}$ either in the counterclockwise direction or in the clockwise direction. At each round we refer to the direction followed in that round as the *walking direction* of the round. We order the vertices of $\mathcal{C}$ from a starting vertex and according to the walking direction. For each walking direction, we use the relation $<$ and the notation $\text{first}(U)$, $\text{last}(U)$, and $\text{next}(v)$ defined in Section 3.

At Step 0, the algorithm adds to $\mathcal{H}$ any vertex $v_0$ of color $c_0$. At each Step $i$ ($i = 1, \ldots, n-1$), a new vertex $v_i$ of color $c_i$ is added to $\mathcal{H}$; adding $v_i$ may imply adding some division vertices. Denote by $H_i$ the set of vertices added to $\mathcal{H}$ up to Step $i$ and as $G_i$ the augmented graph constructed up to Step $i$, i.e. the graph consisting of $\mathcal{C}$ plus all edges and division vertices added during Steps $0, 1, \ldots, i$. For each Step $i$ we define the following two sets:

- $F_i^c = \{v \mid v \in V(\mathcal{C}), v \notin H_i, \text{last}(H_i) < v, col(v) = c\}$, where $c = 0, \ldots, k-1$
- $N_i(u, w) = \{v \mid v \in V(\mathcal{C}), v \notin H_i, u < v < w\}$, where $u < w$.

At the end of Step $i$, the following invariants are maintained:

**Invariant 1.** *Any vertex $v \notin H_i$ is on the external face of $G_i$.*

**Invariant 2.** *Vertex $v_i$ is on the external face of $G_i$.*

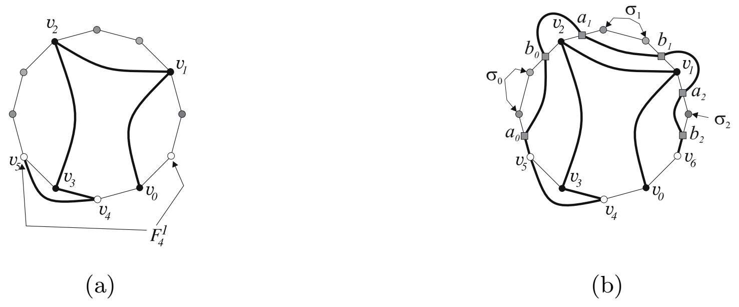


**Fig. 4.** An illustration of different cases of the algorithm to compute $\mathcal{H}$ for a $k$-colored cycle. (a) Case 1 with $v_i = v_4$ and $v_{i+1} = v_5$. Edge $(v_4, v_5)$ is added on the external face because $N_4(v_4, v_5) = \emptyset$. (b) Case 2 with $v_i = v_5$ and $v_{i+1} = v_6$. Sub-sequences $\sigma_0$, $\sigma_1$, and $\sigma_2$ are highlighted; dummy vertices are drawn as small squares.

At Step $i+1$ the algorithm chooses a vertex $v_{i+1}$ of $\mathcal{C}$ to be added to $\mathcal{H}$. The addition of $v_{i+1}$ to $\mathcal{H}$ may require the addition to $\mathcal{H}$ of some dummy edges and some division vertices. We say that a dummy edge is added *inside* $\mathcal{C}$ if it is added on the left-hand side when walking counterclockwise around $\mathcal{C}$. If $F_i^{c_{i+1}} \neq \emptyset$ we choose $v_{i+1} = \text{first}(F_i^{c_{i+1}})$, else invert the walking direction (which starts a new round) and repeat the test. Vertices $v_i$ and $v_{i+1}$ can be connected in $\mathcal{H}$ according to different cases:

**Case 1:** $N_i(v_i, v_{i+1}) = \emptyset$. Refer to Figure 4(a). If $v_i$ and $v_{i+1}$ are not adjacent in $\mathcal{C}$, a dummy edge $(v_i, v_{i+1})$ is added on the external face of $G_i$.

**Case 2:** $N_i(v_i, v_{i+1}) \neq \emptyset$. Refer to Figure 4(b). The vertices of $N_i(v_i, v_{i+1})$ may not form a consecutive sequence along $\mathcal{C}$, because there can be vertices of $H_i$ intermixed with them. Let $\sigma_0, \sigma_1, \ldots, \sigma_{l-1}$ be the maximal sub-sequences of consecutive vertices of $N_i(v_i, v_{i+1})$. For each sub-sequence $\sigma_j$, let $s_j = \text{first}(\sigma_j)$ and $t_j = \text{last}(\sigma_j)$. Also, let $s'_j$ be the vertex (either a "real" vertex of $\mathcal{C}$ or a division vertex) that is immediately before $s_j$ and let $t'_j$ be the vertex (either a "real" vertex of $\mathcal{C}$ or a division vertex) that is immediately after $t_j$. Edges $(s'_j, s_j)$ and $(t_j, t'_j)$ are split by means of the division vertices $a_j$ and $b_j$, respectively. The algorithm connects $v_i$ to $v_{i+1}$ in $\mathcal{H}$ by means of the path $v_i, a_0, b_0, a_1, b_1, \ldots, a_{l-1}, b_{l-1}, v_{i+1}$. Edges $(v_i, a_0)$, $(b_j, a_{j+1})$ $(0 \leq j \leq l-1)$ and $(b_{l-1}, v_{i+1})$ are added on the external face of $G_i$, while edges $(a_j, b_j)$ $(0 \leq j \leq l-1)$ are added inside $\mathcal{C}$.

**Theorem 4.** *Let $G$ be an outerplanar $k$-colored graph and let $S$ be an ordered $k$-colored set compatible with $G$. Then $G$ admits a point-set embedding with at most $4k+1$ bends per edge on $S$. Such a point-set embedding can be computed in $O(n \log n + kn)$ time, where $n$ is the number of vertices of $G$.*

## References

1. P. Bose. On embedding an outer-planar graph on a point set. *Computational Geometry: Theory and Applications*, 23:303–312, 2002.
2. P. Bose, M. McAllister, and J. Snoeyink. Optimal algorithms to embed trees in a point set. *Journal of Graph Algorithms and Applications*, 2(1):1–15, 1997.
3. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing.* Prentice Hall, Upper Saddle River, NJ, 1999.
4. E. Di Giacomo, W. Didimo, G. Liotta, H. Meijer, F. Trotta, and S. K. Wismath. $k$-colored point-set embeddability of outerplanar graphs. Technical Report RT-002-06, Dip. di Ingegneria Elettronica e dell'Informazione, Univ. of Perugia, 2006.
5. E. Di Giacomo, G. Liotta, and F. Trotta. On embedding a graph on two sets of points. *IJFCS, Special Issue on Graph Drawing 2005.* to appear.
6. E. Di Giacomo, G. Liotta, and F. Trotta. How to embed a path onto two sets of points. In *Proc. GD 2005*, volume 3843 of *LNCS*, pages 111–116. Springer, 2005.
7. P. Gritzmann, B. Mohar, J. Pach, and R. Pollack. Embedding a planar triangulation with vertices at specified points. *Am. Math. Month.*, 98(2):165–166, 1991.
8. J. H. Halton. On the thickness of graphs of given degree. *Information Sciences*, 54:219–238, 1991.
9. M. Kaufmann and D. Wagner, editors. *Drawing Graphs*, volume 2025 of *Lecture Notes in Computer Science.* Springer-Verlag, 2001.
10. M. Kaufmann and R. Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *Journal of Graph Algorithms and Applications*, 6(1):115–129, 2002.
11. J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graph and Combinatorics*, 17:717–728, 2001.
12. K. Sugiyama. *Graph Drawing and Applications for Software and Knowledge Engineers.* Word Scientific, 2002.
13. R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Sys., Man, and Cyber.*, SMC-18(1):61–79, 1988.

# Thickness of Bar 1-Visibility Graphs

Stefan Felsner and Mareike Massow

Technische Universität Berlin, Fachbereich Mathematik
Straße des 17. Juni 136, 10623 Berlin, Germany
{felsner,massow}@math.tu-berlin.de

**Abstract.** Bar $k$-visibility graphs are graphs admitting a representation in which the vertices correspond to horizontal line segments, called bars, and the edges correspond to vertical lines of sight which can traverse up to $k$ bars. These graphs were introduced by Dean et al. [3] who conjectured that bar 1-visibility graphs have thickness at most 2. We construct a bar 1-visibility graph having thickness 3, disproving their conjecture. For a special case of bar 1-visibility graphs we present an algorithm partitioning the edges into two plane graphs, showing that for this class the thickness is indeed bounded by 2.

## 1 Introduction

Visibility is a major topic in discrete geometry where a wide range of classes of visibility graphs has been studied, see e.g. [1], [2], [5], [9], [10]. Among the best studied variants are the traditional bar visibility graphs, they admit a complete characterization which has been obtained independently by Wismath [14] and Tamassia and Tollis [13]. On the previous Graph Drawing Symposium, Dean, Evans, Gethner, Laison, Safari and Trotter [3], [4] introduced the class of bar $k$-visibility graphs (B$k$Vs) which 'interpolates' between two classes of graphs with a representation by a family of intervals, namely between bar visibility graphs and interval graphs. Dean et al. are mainly interested in measurements of closeness to planarity of bar $k$-visibility graphs. They prove a bound of 4 for the thickness of bar 1-visibility graphs and conjecture that they actually have thickness at most 2. In Sect. 2 we disprove this conjecture by showing that there are bar 1-visibility graphs with thickness 3. In Sect. 3 we attack the problem from the other side. We consider bars which extend from the $y$-axis to the right, such bars are called *semi bars*. The class of semi bar 1-visibility graphs (SB1Vs) is shown to have thickness 2, the proof is based on an algorithm that partitions the edges of a given SB1V into two planar graphs. In the remainder of this introduction we make the terminology more precise.

### 1.1 Thickness

Thickness is a parameter that measures how far a graph is from being planar: The (graph-theoretic) *thickness* of a graph $G$, denoted by $\theta(G)$, is defined as the minimum number of planar subgraphs whose union is $G$.

Determining the thickness of a graph is NP-hard (see [11]). Exact values are only known for very few classes of graphs. For a survey on theoretical and practical aspects see [12].

Note that in the definition of thickness, the planar embeddings of the subgraphs do not have to coincide. The *geometric thickness* of $G$ asks for the minimum number of subgraphs/colors in the following setting: Choose a straight-line embedding of $G$ and a coloring of the edges such that crossing edges have different colors, the edges of each color then form a plane graph. Geometric thickness was introduced by Dillencourt, Eppstein and Hirschberg in [6]. In [7], Eppstein showed that graph-theoretic thickness and geometric thickness are not even asymptotically equivalent.

## 1.2 Bar $k$-Visibility Graphs

Let a collection of pairwise disjoint horizontal line segments (called *bars*) in the Euclidean plane be given. Construct a graph based on these bars as follows: Take a set of vertices representing the bars. Two vertices are joint by an edge iff there is a line of sight between the two corresponding bars (we then say that the bars *see each other*). A *line of sight* is a vertical line segment connecting two bars and intersecting at most $k$ other bars. A graph is a *bar $k$-visibility graph* (B$k$V) if it admits such a bar representation.

We call the lines of sight that don't intersect any bar *direct*, all others are *indirect* lines of sight; we also use these adjectives for the corresponding edges.

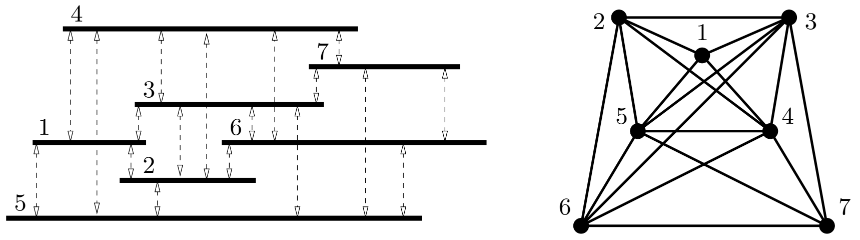


**Fig. 1.** Example of a bar 1-visibility graph

Note that bar visibility graphs can be regarded as bar 0-visibility graphs, and interval graphs as bar $\infty$-visibility graphs. Given a bar representation, we can consider the induced B$k$V for any $k$. In the following, we will – unless otherwise mentioned – only deal with the case $k = 1$. Figure 1 shows an example of a B1V where lines of sight are indicated by dashed lines.

Throughout this paper, we assume that all bars are located at different heights. This can easily be obtained by slightly altering the $y$-coordinates of some bars. We also assume all endpoints of bars to have pairwise different $x$-coordinates by slightly permutating the $x$-coordinates of the endpoints in a given bar representation. (This might result in additional edges, but since we consider problems that only get harder when the number of edges increases, our results extend to general B$k$Vs.) Note that with this assumption, lines of sight can be thougth of as pillars of positive width.

A *semi bar $k$-visibility graph* (SB$k$V) is a bar $k$-visibility graph admitting a representation in which all left endpoints of the bars are at $x = 0$.

Note that for $k = 0$, these graphs have been investigated in [2] where they are identified as the graphs of *representation index* $1 + 1/2$.

For the class of SB$k$Vs the assumption that all endpoints of bars have different $x$-coordinates only refers to the right endpoints. For convenience, we rotate semi bar representations of SB$k$Vs counterclockwise as shown in Fig. 2; in the following we will always think of semi bar representations in this way.

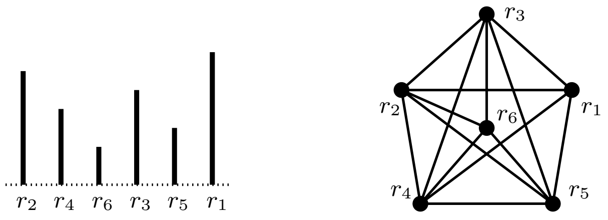


**Fig. 2.** Example of a semi bar 1-visibility graph

Label the bars of a semi bar representation $r_1, r_2, \ldots, r_n$ by decreasing $y$-coordinate of the upper endpoint, i.e., by decreasing height. Reading these labels from left to right we obtain a permutation of $[n]$ which completely determines the graph. The SB1V from the figure is encoded by the permutation $(2, 4, 6, 3, 5, 1)$.

## 2 A Bar 1-Visibility Graph with Thickness 3

In [3], Dean et al. used the Four Color Theorem to show that the thickness of B1Vs is bounded by 4. They conjectured that the correct bound is 2. In this section we construct a B1V with thickness 3. We will often talk about a 2-*coloring* of a graph $G = (V, E)$, meaning a 2-coloring of the edges such that each color class is the edge set of a planar graph on $V$. Given a 2-coloring (with blue and red) we define $G_{\text{blue}}$ and $G_{\text{red}}$ as the graphs on $V$ with all blue and all red edges, respectively.

Here is a brief outline of the construction: First we analyze a quite simple type of graph which has thickness 2 but with the property that every 2-coloring has uniform substructures, so called lampions. Assuming that the original graph is large enough we can assume arbitrarily large lampions. In a second step we introduce a series of slight perturbations into the original graph. It is shown that most of these perturbations have to be incorporated into lampions and the number of perturbations in one lampion is proportional to its size. However a lampion can only absorb a constant number of the perturbations. This yields a contradiction to the assumption that a 2-coloring exists.

We start with an Autobahn where we have heaped up the median strip: Consider the bar representation of the graph $A_n$ shown in Fig. 3. This graph has four outer vertices $A, B, C, D$ and a set $V_{\text{inner}}$ of $n$ inner vertices.

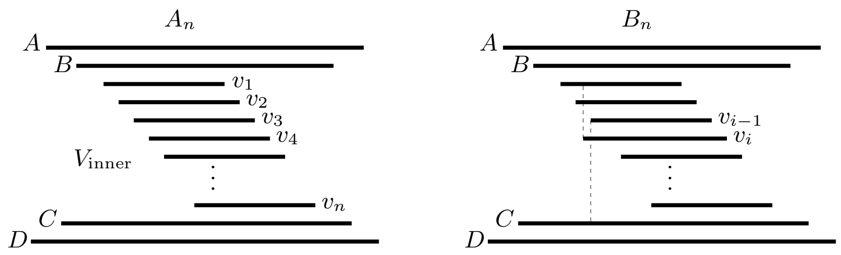


**Fig. 3.** The Autobahn-graph $A_n$ and its modified counterpart $B_n$

Since $A_n$ contains a $K_{4,n}$ as subgraph we know that $A_n$ is non-planar (assuming $n \geq 3$), hence, $\theta(A_n) \geq 2$. To show that $\theta(A_n) = 2$ we let $G_{\text{blue}}$ consist of all direct edges and $G_{\text{red}}$ consist of all indirect edges. Figure 4 shows the partition.

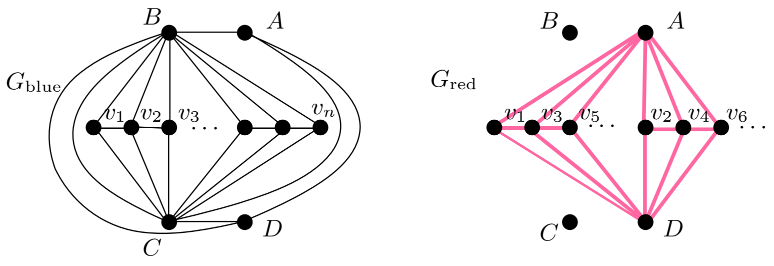


**Fig. 4.** A partition of $A_n$ into two planar graphs

Let $V_{\text{inner}} = \{v_1, v_2, \ldots, v_n\}$, such that the indices represent the order of the right endpoints of the bars from left to right. The inner neighbors of $v_i$ are $v_{i-2}, v_{i-1}, v_{i+1}$ and $v_{i+2}$. The graph $G[V_{\text{inner}}]$ induced by the inner vertices is maximal outerplanar with an interior zig-zag.

A *lampion* in a 2-coloring of $A_n$ consists of a set $W = \{v_i, v_{i+1}, \ldots v_j\}$ of consecutive inner vertices and a partition $\{S_1, S_2\}$, $\{S_3, S_4\}$ of the four outer vertices such that $G_{\text{blue}}$ consists of all zig-zag edges of $G[W]$ and all edges connecting vertices from $W$ with $S_1$ and $S_2$, while $G_{\text{red}}$ consist of the two outer

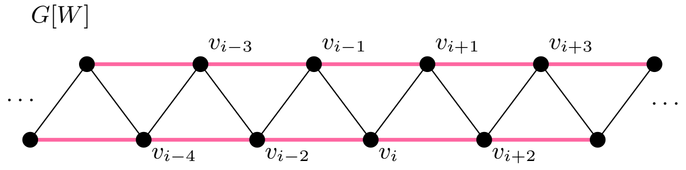


**Fig. 5.** A lampion coloring of $G[W]$

paths of $G[W]$ and all edges connecting vertices from $W$ with $S_3$ and $S_4$ (of course exchanging red and blue again yields a lampion). The set $W$ is the *core of the lampion.* Thus, Fig. 4 shows a lampion with core $V_{\text{inner}}$ together with the additional edges between the four outer vertices.

**Lemma 1.** *For every $k \in \mathbb{N}$ there is an $n \in \mathbb{N}$ such that in every 2-coloring of $A_n$ there is a $W \subset V_{\text{inner}}$ with $|W| \geq k$ such that $W$ is the core of a lampion.*

*Proof.* Each inner vertex has four outer neighbors. Let's call an edge connecting an inner and an outer vertex a *transversal edge.* Consider the blue transversal edges; at each vertex there can be $0, 1, 2, 3$ or $4$ of them. But $G_{\text{blue}}$ is planar and therefore does not contain a $K_{3,3}$. Thus, at most two inner vertices can have the same three outer neighbors in $G_{\text{blue}}$. There are $\binom{4}{3} = 4$ different triples of outer neighbors in $G_{\text{blue}}$, so there can be at most eight inner vertices with more than two outer neighbors in $G_{\text{blue}}$. We might find another eight in $G_{\text{red}}$. These irregular vertices break the sequence $v_1, v_2, v_3, \ldots, v_n$ of inner vertices of $A_n$ into at most 17 pieces. By pigeon-holing, there must be at least one piece with size $n' \geq (n-16)/17$ such that in the induced 2-coloring of $A_{n'}$ all inner vertices are incident to exactly two blue and two red transversal edges.

Considering only the blue transversal edges of $A_{n'}$, the resulting subgraph $G'_{\text{blue}}$ is a subgraph of a blown-up $K_4$ as illustrated in Fig. 6. This subgraph is not arbitrary but has the property that every inner vertex has exactly two incident edges.

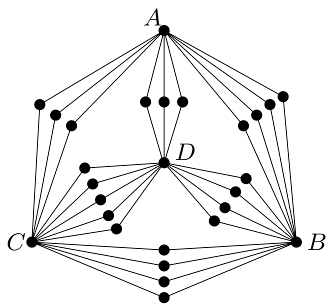


**Fig. 6.** Blown-up $K_4$

Now it remains to planarly embed the *inner edges* of $A_{n'}$, i.e. those of $G[V'_{\text{inner}}]$. To our disposition we have the $\leq 4$ 'large faces' of the blown-up $K_4$, which makes eight faces in total for the two planar graphs. In each of these faces we can embed at most three inner edges. There are other cases with fewer large faces which have in turn more inner vertices at the boundary. In all cases it is impossible to embed more than 12 edges between inner vertices with different outer neighbors in $G'_{\text{blue}}$, the red subgraph may contain another 12 irregular edges. These at most 24 irregularities break the sequence of inner vertices of $A_{n'}$ into at most 25 pieces, we remain with a 2-coloring of $A_{n''}$ with $n'' \geq (n'-24)/25$ such that all inner vertices are incident to the same two outer vertices in $G'_{\text{blue}}$ and to the other two outer vertices in $G'_{\text{red}}$.

We now have $K_{2,n''}$ as a subgraph in both $G''_{\text{blue}}$ and in $G''_{\text{red}}$. The good thing about this is that $K_{2,n''}$ has an (essentially) unique planar embedding. Consequences for the inner edges of $A_{n'}$ are exploited in the following facts.

**Fact 1.** Every inner vertex of $A_{n''}$ has at most two incident inner edges of each color.

It follows that the 2-coloring of $A_{n''}$ induces a 2-coloring of $G[V''_{\text{inner}}]$ such that each color consists of a set of paths and cycles.

**Fact 2.** The set of blue inner edges of $A_{n''}$ contains at most one cycle. The same holds for the red inner edges. If there is a monochromatic cycle, then it is a spanning cycle of $V''_{\text{inner}}$.

There is not much freedom for a 2-coloring of the inner edges of $A_{n''}$ with these properties: We almost have a lampion coloring on $G[V''_{\text{inner}}]$. The exception is that there can be a single *Z-structure* (see Fig. 7) in one color.

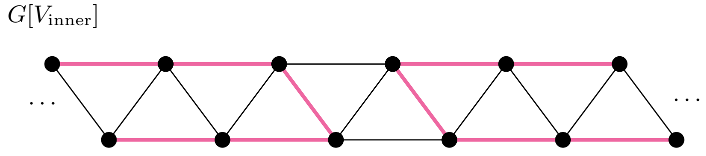


**Fig. 7.** One Z-structure and no monochromatic cycle force all other edge-colors

Removing the Z-structure leaves two consecutive pieces of the sequence of inner vertices. These pieces of $V''_{\text{inner}}$ have a lampion coloring. The size of the larger piece can be estimated as $n''' \geq (n'' - 4)/2$. This proves the lemma. □

Well-prepared we can now look at the variant $B_n$ of $A_n$ in which we have slightly perturbed some of the inner bars (see Fig. 3). To get $B_n$, we have elongated every (say) tenth inner bar by pulling its left endpoint to the left, such that it is further left than the left endpoint of the bar directly above. With this modification we introduce an additional edge between the elongated bar $v_i$ and $v_{i-3}$, but in turn we lose the edge between the bar $v_{i-1}$ and the lowest outer bar $D$. Let's call the vertices corresponding to the elongated bars *modified vertices.*

**Theorem 1.** *The graph $B_n$ is a bar 1-visibility graph with thickness 3, for n large enough.*

*Proof.* We will show that $B_n$ has no 2-coloring. It follows that its thickness is at least 3, and since we can easily use a 2-coloring of $A_n$ and embed the independent additional edges in a third graph, $B_3$ has thickness exactly 3.

Assume that $B_n$ admits a 2-coloring. We first show that any 2-coloring of $B_n$ would have to be very much alike a 2-coloring of $A_n$.

The vertices corresponding to a bar directly above a modified one – let's call them *reduced* – have only three outer neighbors. To avoid a $K_{3,3}$ in one color there can be at most 16 inner vertices incident to more than two transversal edges. In particular, most reduced vertices have to divide their three incident transversal edges into two of one color and one of the other. As in the proof of the lemma we consider a continuous piece in the sequence of inner vertices such that all inner vertices have at most two transversal edges of each color. The graph induced by the largest of these pieces and the outer vertices is $B_{n'}$.

The blue subgraph $G'_{\text{blue}}$ of $B_{n'}$ is a subgraph of a blown-up $K_4$ where at least $9/10$ of the inner vertices have degree 2 and the remaining reduced vertices have degree 1. As in the proof of the lemma it can be argued that there is only a constant number $c$ of edges in $G'_{\text{blue}}$ which join two inner vertices such that there are at least three different outer neighbours of these two vertices, i.e., which join two inner vertices which not belonging to the same blown-up edge of $K_4$. The constant can be bounded as $c \leq 24$. The red graph may contribute another set of $c$ irregular edges. Removing the irregularities will break the sequence of inner vertices into at most $2c+1$ pieces. The graph induced by the largest of these pieces and the special vertices is $B_{n''}$. Assuming that the edges between inner vertices and $D$ are blue in the 2-coloring of $B_{n''}$ the transversal edges of $G''_{\text{blue}}$ and $G''_{\text{red}}$ are shown in Figure 8.

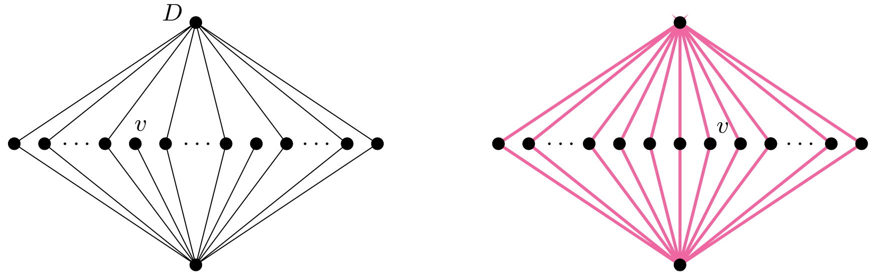


**Fig. 8.** Embedding of a reduced vertex $v$ in $G''_{\text{blue}}$ and $G''_{\text{red}}$

Let $v = v_{i-1}$ be a reduced vertex, the neighbors $v_i$ and $v_{i-3}$ of $v$ both have inner degree 5. In $G''_{\text{red}}$ they have degree (at most) 2, hence, they must have degree 3 in $G''_{\text{blue}}$. This is only possible if $v_i$, $v_{i-1}$ and $v_{i-3}$ form a blue triangle. Since $v_{i-1}$ can have no further blue inner neighbors it follows that the edges $v_{i-1}v_{i-2}$ and $v_{i-1}v_{i+1}$ must be red.

Consider the edge $v_{i-2}v_{i-3}$. Suppose this edge is colored blue. To avoid closing a blue cycle, the edge $v_{i-2}v_i$ must be red. Then to avoid a red cycle the edge $v_iv_{i+1}$ must be blue. Continuing that way the colors of all edges to the right of the blue triangle in Fig. 9 are uniquely determined. To the other side consider the parity of blue and red edges at $v_{i-2}$, this forces $v_{i-2}v_{i-4}$ to be blue, while the parity at $v_{i-3}$ forces two red edges. To avoid a red cycle $v_{i-4}v_{i-5}$ must be blue, whence, parity forces $v_{i-4}v_{i-6}$ to be red. That way the color of edges left of the blue triangle is determined. The complete picture is shown in Fig. 9: We have found a blue Z-structure.

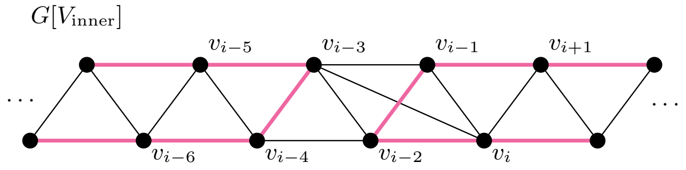


**Fig. 9.** The blue edge $v_{i-3}v_{i-2}$ implies a blue Z

The other case where $v_{i-3}v_{i-2}$ is red leads, by similar arguments, to a red Z-structure.

We have seen that, given a modified vertex in $B_{n''}$, the parity condition and the cycle-freeness of the colored graphs induced by the inner vertices of $B_{n''}$ enforce a Z-structure. The zig-zag emanating from such a Z-structure in one direction has to run into the Z-structure of a second modified vertex. The outer paths of the other color make a turn at a modified vertex – and close a cycle at a second one. This is a contradiction since the blue triangles of the modified vertices are the only monochromatic cycles in $G[V''_{\text{inner}}]$. The contradiction shows that (for $n > 25000$) there is no 2-coloring of $B_n$, hence, the thickness of the graph is 3. □

## 3 Thickness of Semi Bar 1-Visibility Graphs

Let $G = (V, E)$ be an SB1V given by a bar representation, see e.g. Fig. 2. In this section we present an algorithm which 2-colors the edges of $G$ such that each color class forms a plane graph in an embedding induced by the bar representation. Consequently the thickness of an SB1V is at most 2.

Between the full class of B1V graphs and the subclass SB1V there is the class of bar 1-visibility graphs admitting a representation by a set of bars such that there is a vertical line stabbing all bars of the representation. Note that the proof of the previous section implies that already in this intermediate class there are graphs of thickness 3.

### 3.1 One-Bend Drawing

A one-bend drawing of a graph is a drawing in the plane in which each edge is a polyline with at most one bend. Here we introduce a one-bend drawing of semi bar 1-visibility graphs. This drawing is not planar in general, but it will be helpful for the construction and the analysis of the 2-coloring.

Enlarge the bars of each vertex $v$ to a rectangle $B(v)$ with a uniform width. Recall that we assume that the heights of all bars are different and that $B(r_1)$, $B(r_2), \ldots, B(r_n)$ lists the bars by decreasing height. Assign the stripe between the horizontal line touching the top of bar $B(r_i)$ and the horizontal line touching the top of bar $B(r_{i-1})$ to $B(r_i)$; the dotted lines in Fig. 10 separate the stripes. Embed each vertex $v$ at the midpoint of the upper boundary of $B(v)$. We think of the edges as being directed from the longer bar (its *starting bar*) to the shorter bar (its *ending bar*).

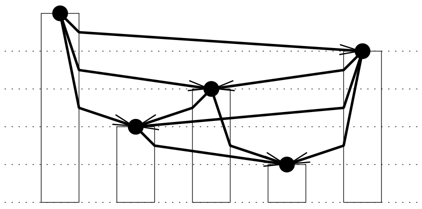


**Fig. 10.** The one-bend drawing of an SB1V

Now draw each edge $e = r_i r_j$ composed of two segments; the first segment is contained in $B(r_i)$, it connects $r_i$ with the inflection point $x_e$, the second segment connects $x_e$ with $r_j$ within the stripe of $r_j$. A good choice for $x_e$ which bewares from crossings between edges emanating from $r_i$ is to place $x_e$ on the vertical boundary of $B(r_i)$ which is closer to $r_j$ with a height which is inside the stripe of $r_j$. We call the segment $(r_i, x_e)$ the *vertical part*, the segment $(x_e, r_j)$ the *horizontal part* of the edge. Note that the stripe associated with $B(v)$ contains the horizontal parts of the incoming edges of $v$. Other edges might cross this stripe, but only with their vertical parts.

## 3.2 2-Coloring Algorithm

Now we present the algorithm 2PLANAR that provides a 2-coloring, i.e., a partition of the edges into two planar graphs (both on the vertex set $V$), using the given embedding. Thus, the algorithm produces planar embeddings of the two graphs such that each edge has only one bend. We think of the partition of the edges as a coloring with blue and red.

The idea of the algorithm is the following: Given the one-bend drawing, start with $r_1$, color all outgoing edges, move on to $r_2$, and so on. The algorithm uses an auxiliary coloring of the bars to determine the color of the edges.

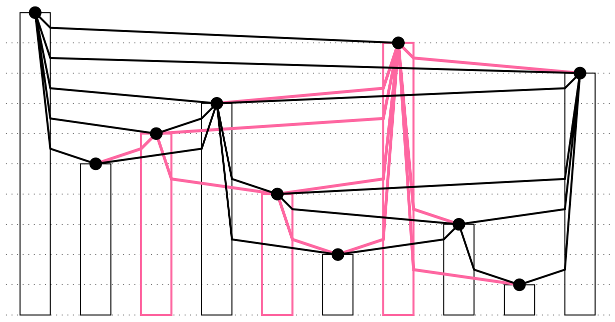


**Fig. 11.** A coloring produced by the algorithm

**Algorithm 2PLANAR**

```
1. Start with $r_1$. Color $B(r_1)$ and all outgoing edges of $r_1$ blue. Whenever such
   an edge traverses another bar, color that bar red.
2. For $i = 2, \ldots, n-1$
   If $B(r_i)$ is uncolored, then color this bar blue.
   For each uncolored edge $e = r_i r_j$
   (a) If $e$ is a direct edge, it obtains the color of its starting bar $B(r_i)$.
   (b) If $e$ is an indirect edge, check if the traversed bar has a color. If so, $e$
       obtains the other color. Otherwise, it receives the color of its starting
       bar $B(r_i)$, and the traversed bar gets the opposite color.
```

Note that 2(b) implies the following:

**Invariant.** Whenever an edge traverses a bar the colors of the edge and the color of the bar are different.

**Theorem 2.** *2PLANAR produces a partition of $E$ into two plane edge sets.*

*Proof.* We have to show that in the 2-coloring computed by 2PLANAR, any two crossing edges have different colors.

The one-bend drawing implies that crossings between edges only appear between the vertical part of one edge and the horizontal part of another edge. Consider a crossing pair $e$, $f$ of edges, assume that the crossing is on the vertical part of $e$ and the horizontal part of $f$. Hence, the crossing point is inside of the starting-bar of $e$, and the edge $f$ is an indirect edge traversing this bar (see Fig. 12).

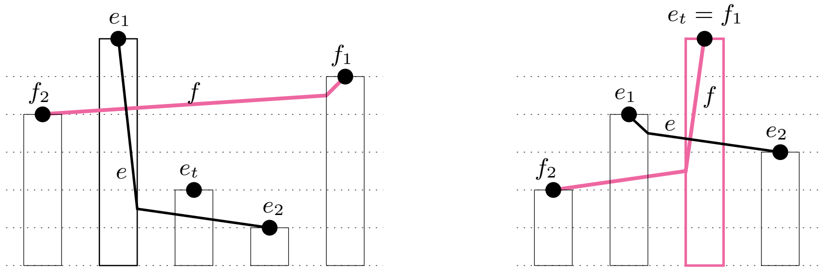


**Fig. 12.** Two crossing edges $e$ and $f$, shown in two possible configurations. Note that there can be many shorter bars between the bars depicted here.

Let the start-vertex of $e$ be $e_1$ and its end-vertex $e_2$. Similarly, let $f$ lead from $f_1$ to $f_2$. Suppose that the color of $f$ is red, then the invariant implies that $B(e_1)$ is blue.

If $e$ is a direct edge it obtains the color of its starting bar, in this case blue. Thus, assume that $e$ is an indirect edge. Then its color depends on the color of the traversed bar, let $B(e_t)$ be this bar. (Note that $e_t = f_1$ or $e_t = f_2$ are possible.)

The key for concluding the proof lies in the following lemma:

**Lemma 2.** *If $B(v)$ is an arbitrary bar, then at most one longer bar can be the starting bar of indirect edges traversing $B(v)$.*

*Proof.* Assume that $x = x_1x_2$ is an edge traversing $B(v)$ with $B(x_1)$ longer than $B(v)$, such that $B(x_2)$ is the shortest ending bar among all such edges. Then we know that between $B(x_1)$ and $B(v)$ in the left-to-right-order there can be no bar longer than $B(x_2)$, else it would block the line of sight corresponding to $x$.

Suppose there is another edge $y = y_1y_2$ starting from a bar $B(y_1)$ which is longer than $B(v)$. The choice of $x$ implies that the horizontal part of $y$ is above the horizontal part of $x$. Since $y$ can traverse only one bar it must connect to a bar $B(y_i)$ which is between $B(v)$ and $B(x_1)$ in the left-to-right-order. Since $y$ is above $x$ the bar $B(y_i)$ is longer than $B(x_2)$. This is in contradiction to the conclusion of the previous paragraph. $\triangle$

Now let's first assume that $B(e_t)$ is shorter than $B(e_1)$. Then by the lemma we know that $B(e_1)$ is the only longer bar sending an edge (e.g. the edge $e$) through $B(e_t)$. Therefore $B(e_t)$ is still uncolored when the algorithm considers $B(e_1)$, therefore, $e$ is colored with the color of $B(e_1)$, which is blue. This shows that the edges $e$ and $f$ have different colors in this case.

If $B(e_t)$ is longer than $B(e_1)$, then we can deduce $e_t = f_1$. For if a bar longer than $B(e_t)$ would be located strictly between $B(f_1)$ and $B(f_2)$ in the left-to-right-order, the line of sight corresponding to $f$ would have to traverse two bars ($B(e_1)$ and this longer bar), which is a contradiction. In addition, we know that $B(f_2)$ is shorter than $B(e_1)$, else $e$ and $f$ would not cross. Thus, we have $e_t = f_1$, and $B(f_1)$ is longer than $B(e_1)$. In this case the lemma tells us that $B(f_1)$ is the only longer bar sending an edge through $B(e_1)$. It follows that $B(e_1)$ was still uncolored when algorithm considered $B(f_1)$. Therefore, the red color of $f$ was chosen equal to the color of the bar $B(f_1)$. The invariant implies that the edge $e$, traversing the red bar $B(f_1)$, is blue. Hence, again $e$ and $f$ have different colors. $\square$

The algorithm shows that SB1Vs have graph-theoretical thickness not more than 2, and it provides a partition of the edges into two planar graphs, providing two plane embeddings. Since the edges are not straight lines, this does not show that the geometric thickness is bounded by 2. We think that any SB1V has a straight-line embedding such that the edges can be partitioned into two plane graphs. For emphasis we state this as:

*Conjecture 1.* Semi bar 1-visibility graphs have geometric thickness at most 2.

## 4 Conclusion and Open Problems

In this paper, we disproved the conjecture of Dean et al. [3] that the tight upper bound on the thickness of bar 1-visibility graphs is 2. We found a B1V with thickness 3, Dean et al. used the Four Color Theorem to show an upper bound of 4 – which still leaves a gap waiting to be closed. Considering that our quite sophisticated structure only yields some independent edges in the third planar layer, we make the following

*Conjecture 2.* The thickness of bar 1-visibility graphs is at most 3.

On the way of getting a better understanding of bar $k$-visibility graphs we considered semi bar $k$-visibility graphs which have a strong combinatorial structure. Here, we proved a tight upper bound on the thickness for the case $k = 1$. We can also use their structure to get tight bounds on the maximum number of edges, the chromatic number and the connectivity of SB$k$Vs. Note for example that the shortest bar in a bar representation of an SB$k$V always corresponds to a vertex with degree at most $2(k + 1)$, which provides a point of attack for inductions on the number of vertices.

These remarks are intended to give an idea of how semi bar $k$-visibility graphs promise to provide more approaches to attack problems about general bar $k$-visibility graphs. The following open questions may serve as a starting point for further research.

1. In [3], it is shown that the thickness of B$k$Vs can be bounded by a function in $k$ (proven is a quadratic one). What is the smallest such function?
2. What is the largest thickness or geometric thickness of SB$k$Vs?
3. What is the largest chromatic number of B$k$Vs? Dean et al. show an upper bound of $6k + 6$.
4. Hartke, Vandenbussche and Wenger [8] found some forbidden induced subgraphs of B$k$Vs. They ask for further characterization of B$k$Vs by forbidden subgraphs.
5. Hartke et al. also examined regular B$k$Vs. Are there $d$-regular B$k$Vs for $d \geq 2k + 3$?
6. What is the largest crossing number of B$k$Vs?
7. What is the largest genus of B$k$Vs?

## References

1. P. Bose, A. M. Dean, J. P. Hutchinson, and T. C. Shermer, *On rectangle visibility graphs*, in Proceedings of Graph Drawing '96, vol. 1353 of Lecture Notes Comput. Sci., Springer, 1997, pp. 25–44.
2. F. J. Cobos, J. C. Dana, F. Hurtado, A. Márquez, and F. Mateos, *On a visibility representation of graphs*, in Proceedings of Graph Drawing '95, vol. 1027 of Lecture Notes Comput. Sci., Springer, 1995, pp. 152–161.
3. A. M. Dean, W. Evans, E. Gethner, J. D. Laison, M. A. Safari, and W. T. Trotter, *Bar k-visibility graphs.* Manuscript, 2005.
4. ———, *Bar k-visibility graphs: Bounds on the number of edges, chromatic number, and thickness*, in Proceedings of Graph Drawing '05, vol. 3843 of Lecture Notes Comput. Sci., Springer, 2006, pp. 73–82.

5. A. M. Dean, E. Gethner, and J. P. Hutchinson, *Unit bar-visibility layouts of triangulated polygons.*, in Proceedings of Graph Drawing '04, vol. 3383 of Lecture Notes Comput. Sci., Springer, 2005, pp. 111–121.
6. M. B. Dillencourt, D. Eppstein, and D. S. Hirschberg, *Geometric thickness of complete graphs*, J. Graph Algorithms & Applications, 4 (2000), pp. 5–17. Special issue for Graph Drawing '98.
7. D. Eppstein, *Separating thickness from geometric thickness*, in Proceedings of Graph Drawing '02, vol. 2528 of Lecture Notes Comput. Sci., Springer, 2002, pp. 150–161.
8. S. G. Hartke, J. Vandenbussche, and P. Wenger, *Further results on bar k-visibility graphs.* Manuscript, November 2005.
9. J. Hutchinson, *Arc- and circle-visibility graphs*, Australas. Journal of Combin., 25 (2002), pp. 241–262.
10. J. P. Hutchinson, T. Shermer, and A. Vince, *On representations of some thickness-two graphs*, Comput. Geom. Theory Appl., 13 (1999), pp. 161–171.
11. A. Mansfield, *Determining the thickness of graphs is NP-hard*, Math. Proc. Camb. Phil. Soc., 9 (1983), pp. 9–23.
12. P. Mutzel, T. Odenthal, and M. Scharbrodt, *The thickness of graphs: A survey*, Graphs and Combinatorics, 14 (1998), pp. 59–73.
13. R. Tamassia and I. G. Tollis, *A unified approach to visibility representations of planar graphs*, Discrete Computational Geometry, 1 (1986), pp. 321–341.
14. S. K. Wismath, *Characterizing bar line-of-sight graphs*, in Proceedings of SCG '85, ACM Press, 1985, pp. 147–152.

# A New Approximation Algorithm for Bend Minimization in the Kandinsky Model*

Wilhelm Barth[1], Petra Mutzel[2], and Canan Yıldız[1]

[1] Institute of Computer Graphics and Algorithms, Vienna University of Technology
Favoritenstraße 9-11, 1040 Wien, Austria
{barth,yildizca}@ads.tuwien.ac.at
http://www.ads.tuwien.ac.at/

[2] Department of Computer Science, University of Dortmund
Otto-Hahn-Str. 14, D-44227 Dortmund, Germany
petra.mutzel@cs.uni-dortmund.de
http://ls11-www.cs.uni-dortmund.de/

**Abstract.** The Kandinsky model has been introduced by Fößmeier and Kaufmann in order to deal with planar orthogonal drawings of planar graphs with maximal vertex degree higher than four [7]. No polynomial-time algorithm is known for computing a (region preserving) bend minimal Kandinsky drawing. In this paper we suggest a new 2-approximation algorithm for this problem. Our extensive computational experiments [13] show that the quality of the computed solutions is better than those of its predecessors [6]. E.g., for all instances in the Rome graph benchmark library [4] it computed the optimal solution, and for randomly generated triangulated graphs with up to 800 vertices, the absolute error was less than 2 on average.

## 1 Introduction

Given a planar graph $G = (V, E, F)$ with a fixed embedding $F$, we consider the problem of finding a region-preserving planar orthogonal drawing with the minimum number of bends. For graphs with maximal degree 4 this problem can be solved in polynomial time [12] if no parallel edge segments leaving a vertex at the same side are allowed. The idea is to set up a network in which each flow corresponds to an orthogonal drawing and vice versa. A bend-minimal orthogonal drawing can thus be obtained from a flow of minimum cost in this network [12], which can be solved in polynomial time [2,10].

Several extensions have been suggested in order to deal with graphs of higher vertex degree. The Kandinsky model has been suggested by Fössmeier and Kaufmann [7]. In this model the vertices are represented by squares of equal size placed on a coarse vertex grid on the plane. The edges consist of continuous sequences of horizontal and vertical line segments routed on a finer edge grid. Thus more than one edge can leave a vertex from the same side, forming a

* Full paper, submitted to GD 2006.

so called 0°-*angle*. Moreover, faces are not allowed to be represented by empty regions (empty faces)[1]. This model has the great advantage that vertices may have prescribed sizes and do not grow arbitrarily as it is the case, e.g., in the GIOTTO model [3], or with respect to the number of edges leaving on one side as in [9].

So far no polynomial time algorithm for computing a bend-minimal drawing in the Kandinsky model, which we refer to as the KMCF-problem, is known[2]. Fössmeier and Kaufmann have suggested the *Kandinsky network*, which can be seen as an extension of the Tamassia network. Unfortunately, a minimum-cost flow in the Kandinsky network must satisfy additional constraints in order to correspond to a bend-minimal Kandinsky drawing of the underlying graph. While there does not exist a polynomial time algorithm to find such a flow, the network can be used for setting up an integer linear program (ILP) for the KMCF-problem.

Bertolazzi et al. [5] have suggested a restriction of the Kandinsky model, the *simple-podevsnef* model, which can be solved in polynomial time. For the KMCF-problem, Eiglsperger [6] has presented a polynomial time algorithm that guarantees to compute a solution with at most twice as many bends as the optimal solution. The algorithm is based on the idea to first compute a minimum-cost flow in the Kandinsky network ignoring the additional constraints, and then to 'repair' the obtained infeasible solution.

In this paper we present the *Cyclic Shift Algorithm* (CS), an alternative 2-approximation algorithm for the KMCF-problem. The idea of our new algorithm is to first solve the LP-relaxation of the corresponding integer linear program (i.e., to ignore the integer constraints), and then to 'repair' the obtained infeasible solution with augmenting cycles. We also consider a variation of the CS algorithm, the *Successive Cyclic Shift Algorithm* (CSS), which performs the cycle augmentations in successive iterations. Extensive computational experiments (see also [13]) have shown that the quality of the computed solutions is better than those of the approximation algorithm in [6]. Moreover, our CS algorithm finds optimal solutions for all instances of the Rome graph benchmark library.

The remainder of this work is organized as follows: In section 2 we briefly refer to the Kandinsky network, which is slightly modified compared to the network described in [7]. In section 3 we describe both variants of the new Cyclic Shift Algorithm (CS and CSS). In section 4 we show that our algorithms guarantee to find a 2-approximate solution to the KMCF-problem. Finally, in section 5 we present experimental results with our new algorithms, two versions of the algorithm suggested in [6], and an exact ILP approach. The tests were performed on the Rome graph benchmark library and on a large set of randomly generated graphs (belonging to different graph classes).

[1] In [7] the Kandinsky model is referred to as *podevsnef*, which originates from "Planar Orthogonal Drawings with Equal Vertex Size and Non-Empty Faces".

[2] The approach suggested in [7] turned out to be not correct, see also Eiglsperger [6].

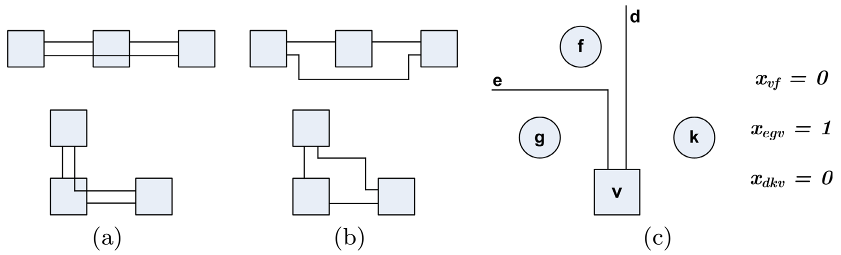


**Fig. 1.** (a) Invalid and (b) valid drawings (c) vertex-bend forced by a 0°-angle

## 2 The Kandinsky Model

Allowing 0°-angles at the vertices leads to several complications which have to be taken into account: Firstly an edge segment may traverse a vertex, secondly an empty face may be generated (Fig. 1a). For preventing this, a bend for each 0°-angle will be forced; such a bend is called *vertex-bend* (see Fig. 1(c)). All other (ordinary) bends are called *edge-bends.* It can be shown [7] that one distinct vertex-bend for each 0°-angle suffices to prevent invalid constructions shown in Fig. 1(a) and to enforce valid Kandinsky drawings (see Fig. 1(b)).

### 2.1 Variables and Constraints for Orthogonal Drawing

We introduce variables $x_{vf} \in \{0, 1, 2, 3, 4\}$ associated with the angle at vertex $v$ between two adjacent edges enclosing the face $f$. One variable unit corresponds to a 90° angle. The sum of all angles at $v$ has to be 360°, i.e.

$$\sum_{f \in F(v)} x_{vf} = 4 \quad \forall v \in V \tag{1}$$

where $F(v)$ denotes all faces adjacent to $v$.

The variables $x_{ef}$ and $x_{eg}$ provide the number of edge-bends on the edge $e \in E$ dividing the two adjacent faces f and g, where $x_{ef}$ represents the convex edge-bends in face $f$ (which at the same time are concave edge-bends in face $g$) and $x_{eg}$ the convex edge-bends in face $g$.

Consider a vertex-bend on an edge $e$ incident to vertex $v$ and separating the faces $f$ and $g$. With the 0/1-variable $x_{egv}$ we associate a vertex-bend forced by a 0°-angle in face $f$ and thus forming a convex bend in $g$ (Fig. 1c). As stated before we have to assure that one of the two edges $d$ and $e$ forming a 0°-angle has a vertex-bend, i.e.

$$x_{vf} + x_{egv} + x_{dkv} \geq 1 \quad \forall v \in V, f \in F(v)\ . \tag{2}$$

Obviously, an edge $e$ is not allowed to have a vertex-bend from the left and the right side in opposite directions. This is forced by the constraint:

$$x_{efv} + x_{egv} \leq 1 \quad \forall v \in V, e \in E(v) \tag{3}$$

where $E(v)$ denotes the set of edges incident with $v$.

To make sure that each face has the correct shape of a rectilinear polygon, it has to be guaranteed that the difference between the number of convex and concave angles is equal to 4 within each inner face and equal to -4 within the outer face. Here angles at vertices (*vertex-angles*) and angles formed by bends (*bend-angles*) must be included in the calculation. This leads to constraints

$$\sum_{e=(v,w)\in E(f)} (x_{ef} - x_{eg} + x_{efv} - x_{egv} + x_{efw} - x_{egw}) - \sum_{v\in V(f)} (x_{vf} - 2) = \mp 4 \tag{4}$$

for all faces $f \in F$, where $g$ is the face on the other side of the edge $(v, w)$ and $E(f)$ (resp. $V(f)$) denotes the set of edges (resp. vertices) on the boundary of $f$. It can be shown that each valid set of variables satisfying (1)-(4) defines a valid Kandinsky shape of the underlying graph [7].

## 2.2 The Corresponding Kandinsky Network

With each graph $G = (V, E, F)$ we associate a network $\mathcal{N} = (N, A)$ with additional constraints, such that the cost of a flow $x$ in $\mathcal{N}$ is equal to the number of bends in the corresponding Kandinsky shape of $G$. The network is a directed graph with the **node set** $N = N_V \cup N_F \cup N_H$, where

| | |
|---|---|
| – $N_V$: | contains a node for each vertex $v \in V$, |
| – $N_F$: | contains a node for each face $f \in F$, |
| – $N_H$: | contains a wreath of artificial nodes around each vertex $v \in V$, |

and the **edge set** $A = A_{VH} \cup A_{HF} \cup A_{FH} \cup A_{FF}$, where an edge in one of these four subsets has the following capacity constraints on the flow and cost:

| | | | |
|---|---|---|---|
| – $A_{VH}$: | $[0 : 4]$ | 0 | represents a vertex-angle, |
| – $A_{HF}$: | $[1 : 4]$ | 0 | lower bound forces a vertex-bend, |
| – $A_{FH}$: | $[0 : 1]$ | 1 | represents a vertex-bend, |
| – $A_{FF}$: | $[0 : \infty]$ | 1 | represents an edge-bend. |

The flow variables in the network correspond to the variables introduced in section 2.1. Before we provide the details for the flow variables in the Kandinsky network, we consider the classical Tamassia network. There, the nodes consist of $N_V \cup N_F$ only, and the edges of $A_{VF} \cup A_{FF}$, where each edge in $A_{VF}$ represents a vertex-angle and has lower bound 1. Fig. 2(a) shows a part of the Tamassia network, which corresponds to the neighbourhood of a vertex with three adjacent edges (dotted lines are edges of the corresponding graph and not of the network). The variables in the figure correspond to flows on the network-edges. The flow has to satisfy the constraints (1) and (4) of section 2.1, hence each vertex-node is a source with a supply of 4 and each face-node a target with a demand of $4 - 2|f|$ (resp. $4 + 2|f|$ if $f$ is the outer face). In [12] it is shown that each minimum-cost flow in this network corresponds to an orthogonal drawing with the minimum number of bends and vice versa.

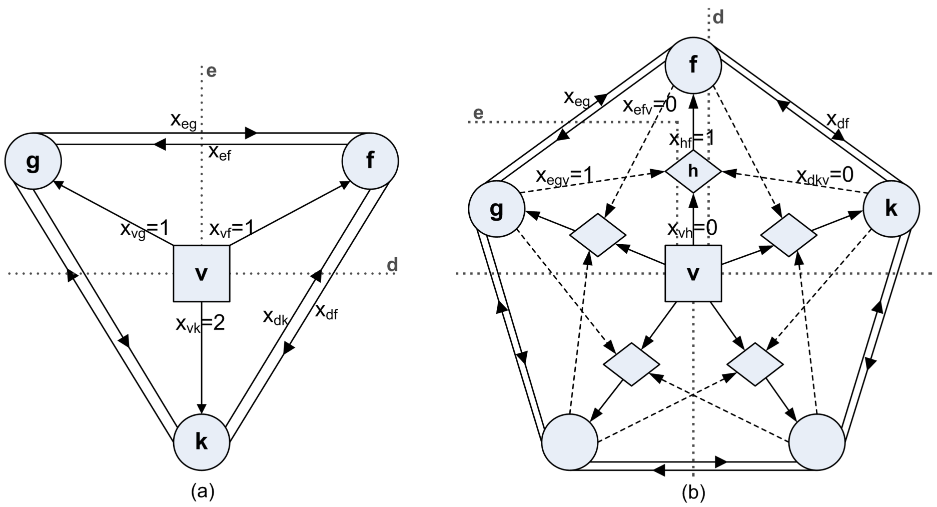


**Fig. 2.** Part of the Tamassia network (a) and the Kandinsky network (b)

In the Kandinsky model 0°-angles (i.e., 0-flow on $vf$-edges) are allowed and we have to guarantee that there is a distinct vertex-bend for each one of them. In the network this can be achieved by introducing an artificial structure as shown in Fig. 2(b). Here each $vf$-edge in the Tamassia network is split into two parts by introducing an artificial node $h$, where the new $vh$-edge represents the vertex-angle (formerly represented by the $vf$-edge). Note that Equation (1) changes in this case to $\sum_{h \in H(v)} x_{vh} = 4$, where $H(v)$ is the wreath of artificial nodes around $v$. A similar change takes place in Equation (4).

Furthermore, new $fh$-edges are introduced, representing the vertex-bends. The new $hf$-edge has a lower bound of 1. So in case of a 0°-angle ($x_{vh} = 0$) one of the two $fh$-edges ending in node $h$ has to carry a flow of 1 unit in order to satisfy the capacity constraint on the $hf$-edge. In this way a wreath of artificial nodes and $fh$-edges is formed around each vertex $v$ such that inequality (2) is always satisfied by a valid flow.

We call a pair of $fh$-edges, each one corresponding to a vertex-bend on the same edge but in opposite directions (pair of intersecting dashed lines in Fig. 2b), **bundles**. At most one of the two edges of a bundle is allowed to carry positive flow. This cannot be achieved by means of network flow techniques. Therefore the flow has to satisfy the additional Constraint (3).

It can be shown that each valid minimum-cost flow satisfying (1), (3), (4) and the capacity constraints in the Kandinsky network corresponds to a bend minimal Kandinsky drawing of the underlying graph $G$ [7,6,13].

## 2.3 The Integer Linear Program

The following ILP is equivalent to the KMCF-problem.

$$\min z = \sum_{e=(v,w)\in E} (x_{ef} + x_{eg} + x_{efv} + x_{egv} + x_{efw} + x_{egw}) \quad \text{(a)}$$

subject to

$$\sum_{h \in H(v)} x_{vh} = 4 \quad \forall v \in V, h \in H(v) \quad \text{(b)}$$

$$\sum_{e=(v,w)\in E(f)} (x_{ef} - x_{eg} + x_{efv} - x_{egv} + x_{efw} - x_{egw}) - \sum_{v \in V(f)} (x_{vh} - 2) = \mp 4 \quad \forall f \in F \quad \text{(c)}$$

$$x_{efv} + x_{egv} \leq 1 \qquad \forall e = (u,v) \in E \quad \text{(d)}$$

$$\begin{array}{ll} 0 \leq x_{vh} \leq 4,\ 1 \leq x_{hf} \leq 4 & \forall v \in Vh \in H(v) \\ 0 \leq x_{ef} \leq \infty,\ 0 \leq x_{efv} \leq 1 & \forall e \in E, f \in F(e), v \in V(e) \end{array} \quad \text{(e)}$$

All variables integer (f)

**Fig. 3.** The integer linear program (ILP) of the KMCF-problem

## 3 The Cyclic Shift (CS) Algorithm

The ILP-formulations of the KMCF-problem and the classical minimum-cost flow (MFC) problem differ only by the *bundle capacity constraints* (Fig. 3.(d)). In section 3.1 we first describe the basic variant of our Cyclic Shift Algorithm (CS), and section 3.2 then describes the *successive* variant (CSS).

### 3.1 The Basic Cyclic Shift (CS) Algorithm

**CS1:** We drop the integer constraint and solve the resulting LP-relaxation. In general the optimum value $z_{CS1}$ of this relaxed problem is smaller than the optimum $z_{opt}$ of the ILP and its solution may contain non-integral values. In particular there may exist bundles, both of whose edges have fractional flows. We call such bundles *critical*, whereas bundles with at least one edge having a zero flow value are called *non-critical.* If the solution is integral, the optimal solution of the ILP is found ($z_{CS1} = z_{opt}$) and the algorithm terminates.
**CS2:** All critical bundles in the solution obtained in step CS1 will be transformed into non-critical ones by augmenting the flow along *correcting cycles.* Note that the constraints (b),(c) and (d) are not violated by these augmentations and additional costs arise only while traversing edges in $A_{FH}$ and $A_{FF}$. We use three different types of cycle-corrections. For details we refer to [13]. In the following we denote with $x(e)$ the flow in an edge $e$ of the Kandinsky network.

*Type 1* (Fig. 4): If the left hand neighbour of a critical bundle $A = (a^L, a^R)$ is a non-critical one $B = (b^L, b^R)$ with $x(b^L) = 0$, then we augment the flow along the correcting cycle $(-a^L, fk, a^R, -vh, vh')$ by $x(a^L)$, so that $A$ becomes non-critical. Note that the capacities of $fk$ and $a^R$ will not be exceeded by this transformation, since $fk$ has infinite capacity and we have $x(a^L) + x(a^R) \leq 1$. Furthermore, the lower bound 1 of edge $hf$ ensures that the flow over $vh$ before

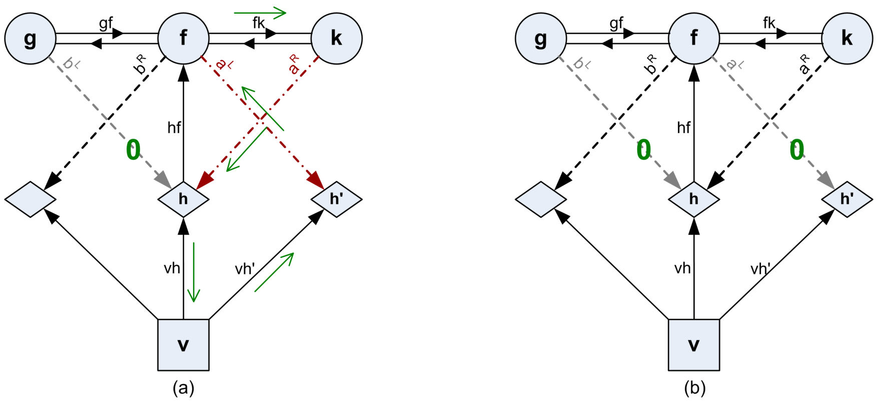


**Fig. 4.** Cycle-correction of Type 1

our transformation is at least $x(a^L)$, since we have $x(vh) \geq 1 - x(a^R) \geq x(a^L)$. The additional cost of this correction is $x(a^L)$.

Now we can apply the same cycle-correction to the right-hand neighbour of bundle $A$ in case that it is critical, too. We continue with these procedure until we reach a non-critical bundle. Thus we transform a chain $I$ of critical bundles between two non-critical ones, into non-critical bundles. The total additional cost is $\sum_{A \in I} x(a^L)$.

We can also perform these transformations in the same way but in the opposite direction beginning with the rightmost bundle of the chain, if its right-hand neighbour is a bundle B with $x(b^R) = 0$. The total cost would be $\sum_{A \in I} x(a^R)$.

*Type 2* (Fig. 5): If $x(b^R) = 0$, a similar left to right correction can be applied. We first augment the flow along the cycle $(-a^R, kf, -hf)$ (Fig. 5a) by $\min(x(a^R), x(hf) - 1)$. Note that this can be done with 0 cost. After this either $a^R$ has zero flow (then $A$ is already non-critical) or $hf$ has flow 1. In the latter case $b^L$ and $a^R$ have a total flow of no more than 1 and we can augment the flow along the correcting cycle $(-a^R, kf, fg, b^L)$ (Fig. 5b) to obtain 0 flow on $a^R$ without violating the capacity constraint of $b^L$.

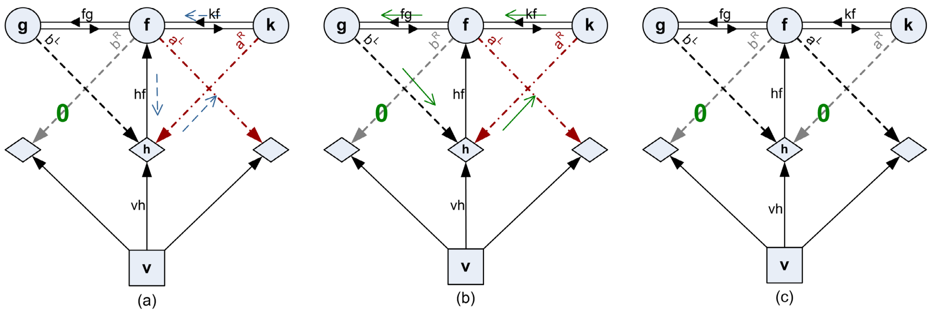


**Fig. 5.** Cycle-correction of Type 2

The bundle $A$ becomes non-critical and we can continue applying the same cycle-corrections on the remaining bundles of the chain $I$. The additional cost is bounded by $2x(a^R)$ for a single bundle and thus by $\sum_{A \in I} 2x(a^R)$ for the entire chain. If $x(b^L) = 0$ holds for the right neighbour of the chain the correction may be performed beginning with the rightmost bundle. In this case, the additional cost is bounded by $\sum_{A \in I} 2x(a^L)$.

It follows that all chains of contiguous critical bundles enclosed by two non-critical bundles can be transformed into non-critical ones by applying cycle-corrections either from right to left or from left to right. In the CS algorithm we always choose the direction with cheaper cost.

*Type 3*: If all bundles in the wreath of bundles around a vertex $v$ are critical, we begin with two adjacent bundles $A$ and $B$ (Fig. 6a) and apply the same cycle-corrections as described in Type 2, so that $A$ becomes non-critical. $B$ not only remains critical, but may even exceed its bundle capacity (Fig. 3.(d)). But the edge capacities are not violated. If we continue applying counter clockwise the same cycle-correction on $B$ and $C$, $B$ becomes non-critical, $C$ eventually exceeds its bundle capacity and so forth. After the cycle-correction is applied on the last critical bundle and $A$ (Fig. 6b), all bundles have become non-critical and $A$ does not exceed its capacity. The total additional cost is bounded by $\sum_{A \in I} 2x(a^R)$. If we process the bundles in the opposite (clockwise) direction the cost is bounded by $\sum_{A \in I} 2x(a^L)$. We choose the cheaper direction.

We will denote the set of all critical bundles of the network by $I_c$ and the cost for correcting all of them by $\text{cost}(I_c)$. Then we have $z_{CS2} := z_{CS1} + \text{cost}(I_c)$.

**CS3:** From each bundle we choose an edge with a zero flow value and set its upper bound to zero, i.e. we *lock* the bundle. After that at most one edge of each bundle can carry flow, thus the bundle capacity constraint is redundant and the KMCF-problem is reduced to a MCF problem.

**CS4:** We solve the MCF problem obtained in step CS3. Since capacities and cost are integer, this problem has always an integer optimal solution that can

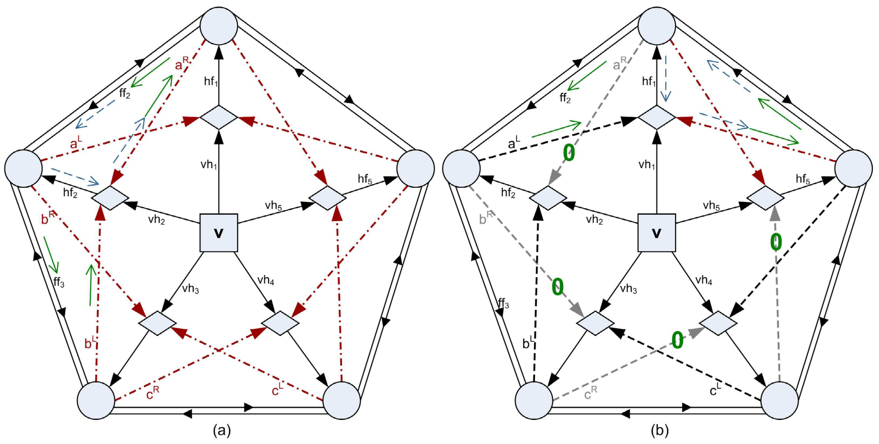


**Fig. 6.** Cycle-correction of Type 3

be found in polynomial time with a standard MCF algorithm. For the obtained objective value $z_{CS}$ the following holds: $z_{opt} \leq z_{CS} \leq z_{CS2}$. If $z_{CS} < z_{CS1} + 1$, the solution found by CS is optimal.

### 3.2 The Successive Cyclic Shift (CSS) Algorithm

Altough the basic variant yields very good practical results, we also experimented with the following version. Here, we lock only the bundles around one certain vertex $v$ in each iteration instead of locking all bundles at once, and solve the LP again. Thus the remaining bundles can adapt their values to this restriction. Bundles once locked, remain locked in following iterations. Each iteration of this algorithm consists of the following steps:

**CSS1:** We solve the LP-relaxation as in CS1. Note that all bundles locked in previous iterations still stay locked.
**CSS2:** Same as CS2. Additionally, we determine for each vertex $v \in V$ the cost $c(v)$ caused by correcting the critical bundles around this vertex.
**CSS3:** We compute a feasible solution by solving the MCF-problem obtained by locking all bundles. If this solution $z_{CSS}$ is better than the ones of the previous iterations, we take it. If $z_{CSS} < z_{CSS_{(1)}} + 1$, where $z_{CSS_{(1)}}$ is the solution of the LP-relaxation in the first iteration, the optimal solution is found and CSS terminates.
**CSS4:** We only lock the bundles around the vertex $v$ with the highest cost $c(v)$ and go to CSS1.

We can go on with the iterations until the solution in step CSS1 is integer. This will happen at the latest when all bundles are locked. But consider that in the third step of each iteration a feasible solution is computed, so we can stop iterating at some point and take the best solution found so far as the final one. In our experiments, we restricted the number of iterations to 5.

## 4 Worst Case Analysis for Quality and Runtime

The objective value of the LP-relaxation in step CS1 is in general smaller than the objective value $z_{opt}$ of the ILP: $z_{CS1} \leq z_{opt}$.

In step CS2 the additional costs cost$(I)$ which arise during the correction of a chain $I$ of critical bundles is, if the correction is performed from left to right

$$\sum_{A \in I} x(a^L) \text{ in case of Type 1 and } \sum_{A \in I} 2x(a^R) \text{ in case of Type 2;}$$

if performed from right to left

$$\sum_{A \in I} x(a^R) \text{ in case of Type 1 and } \sum_{A \in I} 2x(a^L) \text{ in case of Type 2.}$$

Since we choose the cheaper direction, provably the following holds:

$$\text{cost}(I) \leq \sum_{A \in I} (x(a^L) + x(a^R)) \tag{5}$$

Thus for the cost caused by the correction of all the chains we obtain:

$$z_{CS2} \leq z_{CS1} + \sum_{I \in I_C} \sum_{A \in I} (x(a^L) + x(a^R)) \leq 2z_{CS1} \leq 2z_{opt} \tag{6}$$

In the following two steps we lock bundle-edges which are not used anyway and reoptimize, thus the new objective value $z_{CS}$ can not be greater than $z_{CS2}$:

$$z_{CS} \leq z_{CS2} \leq 2z_{opt} \tag{7}$$

The basic variant CS has therefore the approximation factor 2. Since the same solution is obtained in the first iteration of CSS and will be replaced in later iterations only by better ones, CSS does not give worse results.

The running time of step CS2 is linear in $|E|$ as there are two bundles for each edge $e \in E$, and each critical bundle is processed only once with constant expense. Thus the total running time is $O(LP) + O(|E|) + O(MCF)$, hence polynomial. With its constant number of iterations (five in our case) CSS has also a polynomial running time.

## 5 Experimental Results

We have compared the solutions of our algorithms CS and CSS with two versions TE and TES of the algorithm by Eiglsperger [6], and the optimal solutions obtained by the ILP. A short description of TE and TES is as follows[3]:

| | |
|---|---|
| TE1: | Solve the KMCF-problem without the bundle capacity constraint |
| TE2: | Correct overfilled bundles |
| TE3: | Lock each bundle and solve the resulting MCF-problem. |
| TES1,TES2: | same as TE1 and TE2 |
| TES3: | Lock each bundle and solve the resulting MCF-problem; keep the solution, if it is better than the previous. |
| TES4 | Lock only those bundles, which have flow value 1 before TES2 and zero flow afterwards. Go to TES1. |

Besides the Rome graphs [4] (11,529 graphs with up to 100 nodes) we used a randomly generated data set consisting of about 14,000 planar graphs of several classes with up to 800 nodes. The non-planar Rome graphs were planarised with the *Subgraph Planarizer* of the AGD-Library [1]. The test runs were performed on an Intel Pentium IV 2.8GHz with 2GB RAM using ILOG CPLEX 8.1 [11]. We used the *Mixed Integer Optimizer* for the ILP, the *Network Optimizer* for the minimum-cost flow problems (CS4,CSS3,TE), and the standard settings for the LP (CS1/CSS1).

All instances of the Rome graphs have been solved to optimality by our basic CS algorithm. The optimality has been proven for 99.9% of them by the algorithm itself, i.e., $z_{CS} < z_{CS1} + 1$; only for 10 graphs we needed to calculate the

[3] For a detailed description we refer to [6]. We have slightly modified TE and TES to be conform with the implementations of CS and CSS, improving their results by doing so. For details, see [13].

**Table 1.** Experimental results for (random) maximal planar graphs

| **row** | $\vert V\vert$ | **100** | **200** | **300** | **400** | **500** | **600** | **700** | **800** |
|---|---|---|---|---|---|---|---|---|---|
| **1** | $z_{TE} - z_{opt}$ | 17.83 | 36.97 | 56.81 | 93.27 | 112.61 | 138.37 | 160.26 | 182.75 |
| **2** | $z_{TES} - z_{opt}$ | 7.54 | 16.08 | 24.35 | 30.68 | 39.40 | 45.61 | 52.82 | 62.61 |
| **3** | $z_{CS} - z_{opt}$ | 0.30 | 1.24 | 1.36 | 2.02 | 3.19 | 3.67 | 4.84 | 5.41 |
| **4** | $z_{CSS} - z_{opt}$ | 0.03 | 0.23 | 0.36 | 0.42 | 0.83 | 0.86 | 1.62 | 1.62 |
| **5** | $z_{opt}$ | 234.52 | 474.61 | 715.66 | 956.08 | 1196.06 | 1437.62 | 1678.63 | 1918.34 |
| **6** | $z_{CS1} - z_{opt}$ | −0.01 | −0.01 | −0.03 | 0 | −0.03 | −0.01 | −0.04 | −0.06 |
| **7** | $z_{TE1} - z_{opt}$ | −0.02 | −0.03 | −0.07 | −0.03 | −0.07 | −0.2 | −0.2 | −0.23 |
| **8** | $\#(z_{opt} = z_{CS})$ | 80 | 44 | 35 | 19 | 13 | 6 | 4 | 3 |
| **9** | $\#(z_{opt} = z_{CSS})$ | 97 | 82 | 76 | 68 | 47 | 49 | 30 | 26 |
| **10** | $t_{TE}$ | 0.05 | 0.11 | 0.19 | 0.30 | 0.42 | 0.57 | 0.73 | 0.93 |
| **11** | $t_{CS}$ | 0.09 | 0.32 | 0.55 | 0.89 | 1.36 | 1.89 | 2.57 | 3.30 |
| **12** | $t_{CSS}$ | 0.1 | 0.41 | 0.77 | 1.32 | 2.19 | 3.23 | 4.61 | 5.91 |
| **13** | $t_{TES}$ | 0.73 | 2.21 | 4.37 | 7.52 | 11.68 | 16.45 | 22.07 | 28.70 |
| **14** | $t_{ILP}$ | 0.24 | 1.57 | 3.95 | 8.02 | 13.38 | 23.24 | 41.35 | 49.91 |

optimal ILP-value for confirmation. In [6], Eiglsperger has described his experimental results with the successive variant of his algorithm which was able to solve most of the Rome graphs to optimality. However, on 392 instances it has produced 1 additional bend, and on 13 instances 2 additional bends.

Table 1 shows our experimental results for the maximal planar graphs that turned out to be the most difficult instances in our randomly generated test set (see [13]). Rows 1-5 show the average absolute errors of the four algorithms and the average number of bends in the optimal solution, each averaged over 100 instances of the same size. Expectedly the successive variants perform much better. Note that the average relative error of our CSS algorithm does not exceed 0.10%, while it is between 3.15% and 3.40% for TES. Moreover, the average absolute error of CSS is less than 2 for all tested instances. Thus the question arises if it is worth the effort of solving the ILP for getting the exact optimum. Though the ILP has a tolerable running time for small instances, it becomes unacceptable for larger ones because of its exponential increase (row 14).

The objective values obtained by the infeasible solutions of the first steps CS1 and TE1, resp., provide the starting values for the cycle correction steps of the approximation algorithms. Therefore, we were interested in the underestimation for the number of bends obtained by CS1 and TE1 (see rows 6 and 7). It can be observed that the objective value $z_{CS1}$ is very close to the optimal value $z_{opt}$, and always closer than $z_{TE1}$. The latter observation is always true, since for computing $z_{CS1}$ we only omit the integer constraints whereas for computing $z_{TE1}$, the bundle constraints are omitted and with them automatically the integer constraints. Rows 8 and 9 show how often our algorithms CS and CSS have found the optimal solutions (out of 100 instances per size), and rows 10-14 give the running times of the four algorithms and the ILP approach.

## References

1. *AGD User Manual*, 1999. `http://www.ads.tuwien.ac.at/AGD/`
2. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows. *Prentice-Hall*, 1993.
3. Batini, C., Nardelli, E., Tamassia, R.: *A Layout Algorithm for Data Flow Diagrams.* IEEE Trans. Softw. Eng., SE-12(4), pages 538-546, 1986.
4. Di Battista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., Vargiu, F.: *An Experimental Comparison of Three Graph Drawing Algorithms.* Proc. 11th Ann. ACM Symp. Comput. Geom., pages 306-315, 1995.
5. Bertolazzi, P., Di Battista, G., Didimo, W.: *Computing orthogonal drawings with the minimum number of bends.* IEEE Trans. Comput., 49(8), pages 826-840, 2000.
6. Eiglsperger, M.: *Automatic Layout of UML Calss Diagrams: A Topology-Shape-Metrics Approach.* PhD thesis, Eberhard-Karls-Universität zu Tübingen, 2003.
7. Fößmeier, U., Kaufmann, M.: *Drawing high degree graphs with low bend numbers.* In F.J. Brandenburg, editor, Proc. 3rd Int. Symp. on Graph Drawing (GD'95), volume 1027 of LNCS, pages 254-266. Springer, 1996.
8. Fößmeier, U.: *Orthogonale Visualisierungstechnicken für Graphen.* PhD thesis, Eberhard-Karls-Universitä t zu Tübingen, 1997.
9. Fößmeier, U., Kaufmann, M.: *Algorithms and Area Bounds for Nonplanar Orthogonal Drawings.* In G. Di Battista, editor, Proc. 5th Int. Symp. on Graph Drawing (GD'97), volume 1353 of LNCS, pages 134-145. Springer, 1997.
10. Garg, A., Tamassia, R.: *A New Minimum Cost Flow Algorithm with Applications to Graph Drawing.* In S.C. North, editor, Proc. 4th Int. Symp. on Graph Drawing (GD'96), volume 1190 of LNCS, pages 201-216. Springer, 1997.
11. ILOG CPLEX 8.1: `http://www.ilog.com/products/cplex/`
12. Tamassia, R.: On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3), pages 421–444, 1987.
13. Yildiz, C.: *Knickminimales Orthogonales Zeichnen Planarer Graphen im Kandinsky Modell.* PhD thesis, Vienna University of Technology, 2006.

# Radial Drawings of Graphs: Geometric Constraints and Trade-Offs*

Emilio Di Giacomo, Walter Didimo, and Giuseppe Liotta

Dip. Ingegneria Elettronica e dell'Informazione - Università degli Studi di Perugia
{digiacomo, liotta, didimo}@diei.unipg.it

**Abstract.** This paper studies how to compute radial drawings of graphs by taking into account additional geometric constraints which correspond to typical aesthetic and semantic requirements for the visualization. The following requirements are considered: vertex centrality, edge crossings, curve complexity, and vertex radial distribution. Trade-offs among these requirements and efficient drawing algorithms are presented.

## 1 Introduction

The *readability* of a drawing of a graph describes its effectiveness in conveying the information associated with the graph itself. The set of geometric requirements related to the readability of a drawing are called *aesthetic requirements* and those related to the semantics are called *semantic requirements* (see, e.g., [6,11,18]). While aesthetic requirements are usually expressed as geometric optimization goals for a graph drawing algorithm, semantic requirements express constraints that must be satisfied in the output visualization and are provided to the algorithm as an additional input. Taking into account more than one aesthetic requirement typically translates into a multi-objective optimization problem, which is inherently characterized by trade-offs. Many such trade-offs have received attention in the literature, including area vs. angular resolution, area vs. aspect ratio, edge crossings vs. number of bends (see also [6,17]).

This paper is devoted to the study of aesthetic and semantic requirements which occur when computing radial drawings of graphs. A *radial drawing* of a graph is such that every vertex is drawn on one of $k$ concentric circles and the edges are polygonal chains. Radial drawings arise in all those applications where it is important to display a graph with the constraint that some vertices are drawn "more central" than others. Examples of such applications include social networks analysis (visualization of policy networks and co-citation graphs), cybergeography (visualization of Web maps and communities), and bioinformatics (visualization of protein-protein interaction diagrams). See, e.g., [9,10].

In spite of their importance in practice, the study of visualization algorithms and systems that compute radial drawings of graphs by taking into account different aesthetic and semantic requirements has not yet received enough attention. Namely, there exist visualization systems that compute radial drawings

* Research partially supported by MIUR under Project "ALGO-NEXT".

that satisfy the semantic requirement of placing each vertex on a given circle corresponding to its centrality [3,4]; however, these systems often disregard some basic aesthetic requirements, for example they give rise to drawings with several edge crossings. On the other hand, the problem of computing radial drawings of planar graphs with no edge intersections and no bends along the edges has been studied in [7]; however, the drawing algorithm does not take into account any semantic requirements. The problem of testing whether a graph admits a crossing-free radial drawing that satisfies the assigned centrality of the vertices and where the edges are monotone Jordan curves has been studied by Bachmaier et al. [1,2]; however, only a planar embedding (not a drawing) is returned and furthermore the number of bends along the edges is not considered.

We describe different algorithms that compute radial drawings of graphs and present trade-offs among typical aesthetic and semantic requirements. We consider the following requirements: *vertex centrality* (each vertex should be placed on the circular level corresponding to its centrality), *edge crossings* (the number of edge crossings should be small), *curve complexity* (the number of bends along each edge should be small), *radial distribution of the vertices* (the vertices should be uniformly distributed in a radial fashion on a polar grid [16]). An outline of the main results in this paper is as follows.

- We show that in general it is not possible to compute radial drawings of planar graphs where no two edges cross, the number of bends is zero, the radial distribution of the vertices is uniform and the vertex centrality is satisfied. Since vertex centrality is a semantic requirement and hence it must be satisfied and since reducing the number of crossings is recognized as one of the most important aesthetics in graph drawing applications (see, e.g., [14,15]), we give precedence to these two requirements over the others.
- A consequence of a result by Pach and Wenger [13] is that radial drawings of planar graphs having zero edge crossings, uniform radial distribution, and respecting vertex centrality can be computed at the price of a high curve complexity (there can be a linear number of bends per edge). We show how to achieve low curve complexity by describing a linear-time algorithm that computes radial drawings of planar graphs with no edge crossings, uniform radial distribution, and at most three bends per edge.
- Trade-offs between the number of bends per edge and the uniform radial distribution are studied. We describe linear-time algorithms that can further reduce the number of bends at the expenses of a non-uniform radial distribution. We show that every planar graph with assigned vertex centrality admits a radial leveled planar drawing with at most one bend per edge if the radii of the circles are not given as part of the input and with at most two bends per edge if the radii are fixed in advance.

The results in this paper are based on a combination of geometric and graph theoretic techniques. Three bends per edge and uniform radial distribution are achieved by exploiting properties of star-shaped polygons and Hamiltonian augmentation. Radial drawings with at most two bends per edge and with at most one bend per edge are computed by using canonical ordering [5] and curve embedding [8], respectively. Figure 1 shows different radial drawings of the same graph with assigned centrality for the vertices: the white vertices are assigned to the external circle, the black ones to the inner circles, and the grey vertices to the mid circle. The drawing of Figure 1(a) has curve complexity zero but has a high number of crossings and poor radial distribution of the vertices. The drawing of Figure 1(b) is optimal in terms of crossings and radial distribution of the vertices and has a small number of bends per edge. The number of bends per edge is reduced in Figure 1(c) at the expenses of a sub-optimal radial distribution.

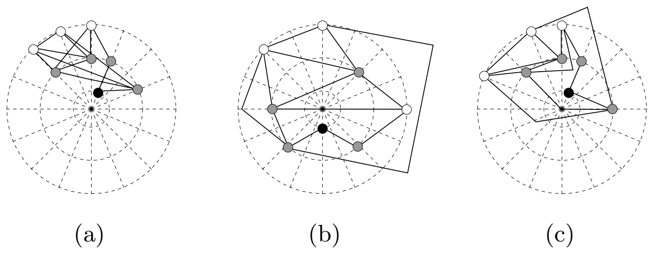


**Fig. 1.** Three radial drawings of the same graph with assigned centrality for the vertices. The three drawings present different trade-offs among aesthetic requirements.

## 2 Preliminaries

We assume familiarity with basic concepts of graph drawing [6]. Let $G$ be a planar graph and let $\Gamma$ be a drawing of $G$. If the edges of $G$ are represented in $\Gamma$ as a polygonal chain we say that $\Gamma$ is a *polyline drawing*. The intersection between two consecutive straight-line segments in the polygonal chain representing an edge $e$ is called a *bend* of $e$. If a drawing $\Gamma$ is such that each edge is represented with a polygonal chain with at most $b$ bends we say that $\Gamma$ is a *$b$-bend drawing* of $G$. A 0-bend drawing is also called a *straight-line drawing*. A *radial drawing* of a graph $G$ is a polyline drawing of $G$ such that each vertex is drawn as a point of one among a set $\mathcal{C}$ of concentric circles. We assume that the center of the circles of $\mathcal{C}$ is the origin $o$ of the Euclidean plane. A *ray* is a half-line with origin the point $o$. Given a ray $\varrho$ we denote by $\angle\varrho$ the counterclockwise angle required to bring the positive $x$-axis into correspondence with $\varrho$. Given two rays $\varrho_a$ and $\varrho_b$, we define $\angle\varrho_a\varrho_b = \angle\varrho_b - \angle\varrho_a$, where angles are measured modulo $2\pi$. A *star-shaped polygon* $\mathcal{P}$ is a polygon in which there exists an interior point $p$ such that all the boundary points of $\mathcal{P}$ are visible from $p$. The set of all points $p$ satisfying this property is called the *kernel* of $\mathcal{P}$.

## 3 Semantic and Aesthetic Requirements

As anticipated in the introduction, the semantic requirement that we take into account is vertex centrality, that associates the vertices in the input graph

$G(V, E)$ to radial levels. For each vertex $v \in V$, its centrality (radial level) is specified as part of the input and the algorithm computes a drawing such that $v$ is drawn on a circle corresponding to the given centrality. More formally, a *leveled graph* $G = (V, E, \phi)$, consists of a set of vertices $V$, a set of edges $E$ and a function $\phi : V \rightarrow \{0, 1, \ldots, k-1\}$ that maps each vertex to an integer between 0 and $k - 1$, which represents its centrality. A *radial leveled drawing* of $G = (V, E, \phi)$ is a radial drawing of $G = (V, E, \phi)$ on a set of $k$ concentric circles $\mathcal{C} = \{C_0, \ldots, C_{k-1}\}$ (with the radius of $C_i$ greater than the radius of $C_{i+1}$) such that each vertex $v \in V$ is drawn as a point of circle $C_{\phi(v)}$. The value $k$ will be also called the *level number* of $G$, and will be denoted as $\lambda(G)$. Since in this paper we only study radial leveled drawings of leveled graphs, from now on we will often call them simply "radial drawings", omitting the term "leveled".

Concerning the visual appeal of radial drawings, we focus on the following aesthetic requirements:

– CROSSINGS. A *crossing* between two edges occurs if the two edges share a point different from their end-vertices. A drawing should have as few crossings as possible, ideally 0 if the graph is planar.
– CURVE COMPLEXITY. The *curve complexity* is the maximum number of bends per edge. A readable drawing typically has low curve complexity (see, e.g., [6]).
– RADIAL DISTRIBUTION. In a radial drawing it is desirable that the vertices are uniformly distributed on a polar grid (see, e.g., [16]). Namely, the difference between the radii of any two consecutive circles should be constant and equal to the radius of the smallest circle; also, the angular distance between any two consecutive vertices encountered with a radial sweep of the drawing should be constant. More formally, we shall measure the radial distribution of the vertices in terms of:
  • RADIAL DISTANCE RATIO (RDR). Denote by $r_i$ the radius of $C_i$ ($i = 0, \ldots, k-1$) and set $r_k = 0$. Define $\Delta r_i = r_i - r_{i+1}$ ($i = 0, \ldots, k-1$), $\Delta r_{min} = \min_i\{\Delta r_i\}$, and $\Delta r_{max} = \max_i\{\Delta r_i\}$. The *Radial Distance Ratio* is defined as RDR$=\frac{\Delta r_{max}}{\Delta r_{min}}$.
  • ANGULAR DISTANCE RATIO (ADR). Let $v$ be a vertex of $G$ and let $\varrho_v$ be the ray passing through $v$. Let $\rho_0, \rho_1, \ldots, \rho_{h-1}$ ($h \geq 1$) be the distinct elements of the set $\{\varrho_v \mid v \in V\}$, ordered so that $\angle\rho_0 < \angle\rho_1 < \cdots < \angle\rho_{h-1}$. If $h > 1$, define $\alpha_i = (\angle\rho_{i+1} - \angle\rho_i)$ (the indices are taken modulo $h$ and the angles are measured modulo $2\pi$), $\alpha_{min} = \min_i\{\alpha_i\}$ and $\alpha_{max} = \max_i\{\alpha_i\}$. If $h = 1$ we define $\alpha_{min} = 0$ and $\alpha_{max} = 2\pi$. The *Angular Distance Ratio* is defined as ADR$= \frac{\alpha_{max}}{\alpha_{min}}$. Notice that, when $h = 1$ we have ADR $= +\infty$.

We say that a radial drawing is *optimal in terms of* CROSSINGS if it is a planar drawing; it is *optimal in terms of* CURVE COMPLEXITY if it is a straight-line drawing; it is *optimal in terms of* RADIAL DISTRIBUTION if both RDR $= 1$ and ADR $= 1$. Ideally, one would like to produce radial drawings that satisfy the semantic requirements and are optimal in terms of all the aesthetic requirements described above. Unfortunately, this is not always possible, as showed by the next result.

**Lemma 1.** *There exists a planar leveled graph that does not have a radial leveled drawing optimal in terms of* CROSSINGS *and* CURVE COMPLEXITY *and* RADIAL DISTRIBUTION.

*Proof.* Let $G = (V, E, \phi)$ be the planar leveled graph defined as follows (refer to Figure 2). $V = \{u_0, u_1, \dots, u_{h-1}\} \cup \{v_0, v_1, \dots, v_{h-1}\} \cup \{w_0, w_1, \dots, w_{h-1}\}$, $E = \{(u_i, u_{i+1}), (v_i, u_i), (v_i, u_{i+1}), (w_i, u_i), (w_i, u_{i+1}), (w_i, v_i) \mid 0 \leq i \leq h-1\}$ (indices are taken modulo $h$), $\phi(u_i) = 0$, $\phi(v_i) = 0$, and $\phi(w_i) = 1$ ($i = 0, \dots, h-1$). Consider the cycle induced by vertices $\{u_0, u_1, \dots, u_{h-1}\}$; since the drawing must be optimal in terms of CROSSINGS, each edge $(u_i, u_{i+1})$ must be drawn as a chord of $C_0$. Vertices $v_i$ ($i = 0, \dots, h-1$) must also be drawn on $C_0$ and outside the polygon representing the cycle. The counterclockwise order of the vertices along $C_0$ must be $u_0, v_0, u_1, v_1, u_2, \dots, u_{h-2}, v_{h-2}, u_{h-1}, v_{h-1}$ or the opposite one. Let $\varrho_{u_i}$ be the ray passing through $u_i$ ($i = 0, \dots, h-1$) and let $\alpha_i = (\angle\varrho_{u_{i+1}} - \angle\varrho_{u_i})$. The angle $\alpha_{min} = \min_i\{\alpha_i\}$ is at most $\frac{2\pi}{h}$ (if we want an optimal ADR all $\alpha_i$ must be equal to $\frac{2\pi}{h}$). Each vertex $w_i$ ($i = 0, \dots, h-1$) must be drawn inside the triangle representing the cycle $u_i, u_{i+1}, v_i$ in order to have a planar drawing, and on $C_1$ in order to satisfy the vertex centrality requirement. This implies that the $C_1$ must cross every segment $\overline{u_i u_{i+1}}$, i.e. the radius $r_1$ must be greater than the minimum distance $d_{min}$ of any of these segments from the center of the circles. The value of $d_{min}$ is equal to $r_0 \cos(\pi/h)$. Thus we have $r_1 > r_0 \cos(\pi/h)$, i.e. $\frac{r_1}{r_0} > \cos(\pi/h)$. In order to have RDR $= 1$ it must be $r_0 = 2r_1$, i.e. $\frac{1}{2} > \cos(\pi/h)$. This inequality is never satisfied for $h \geq 3$, and therefore, for any $h \geq 3$ it is not possible to obtain RDR $= 1$.

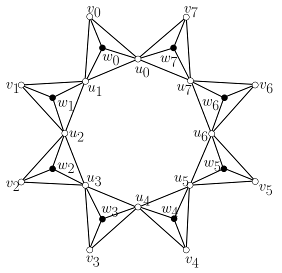


**Fig. 2.** A planar leveled graph that does not admit a radial leveled drawing optimal in terms of CROSSINGS and CURVE COMPLEXITY and RADIAL DISTRIBUTION. White vertices have centrality 0; black vertices have centrality 1.

Lemma 1 naturally raises the question about whether one can relax one of the requirements in order to have drawings that are optimal for the other two. We consider the cases in which either CURVE COMPLEXITY or RADIAL DISTRIBUTION are relaxed. For example, it is not difficult to see that the graph used for the proof of Lemma 1 can be drawn on two circles that are sufficiently close with each other and therefore the aesthetic requirement RADIAL DISTRIBUTION is relaxed.

However, the simultaneous optimality of CROSSINGS and CURVE COMPLEXITY cannot always be achieved. The proofs of the next two lemmas are omitted for reasons of space.

**Lemma 2.** *There exists a planar leveled graph that does not have a planar radial leveled drawing optimal in terms of both* CROSSINGS *and* CURVE COMPLEXITY*.*

**Lemma 3.** *Every planar leveled graph with $n$ vertices has a radial leveled drawing optimal in terms of both* CROSSINGS *and* RADIAL DISTRIBUTION*, and such that* CURVE COMPLEXITY *is $\mathcal{O}(n)$.*

Motivated by Lemma 3, we study in the next section whether one can reduce the number of bends in a planar radial drawing while maintaining optimality for RADIAL DISTRIBUTION. We will then consider trade-offs between RADIAL DISTRIBUTION and CURVE COMPLEXITY, showing how the latter can be reduced at the expenses of the former.

## 4 Radial Drawings with No Crossings, Optimal Radial Distribution and Curve Complexity 3

We describe a drawing algorithm that computes a radial drawing of a planar leveled graph $G$ with at most three bends per edge while maintaining optimal radial distribution and no crossings. An outline of our drawing technique is as follows: (i) If $G$ is not Hamiltonian, dummy edges and vertices are added so that the augmented graph is Hamiltonian and planar. (ii) The vertices of the Hamiltonian circuit are drawn as a star-shaped polygon whose vertices have the wanted centrality and have a uniform radial distribution on a polar grid. (iii) Every edge $e$ not belonging to the Hamiltonian circuit is either drawn inside or outside the polygon; if $e$ is drawn inside it has at most one bend, if it is drawn outside it has at most two bends. (iv) The dummy vertices and edges are finally removed; the construction is such that every edge can have at most three bends. Similar techniques have been previously used (see, e.g., [12,13]).

Before giving a more detailed description of the algorithm and analyzing its properties, we recall some useful definitions and results about Hamiltonian augmentation. A graph $G$ is *Hamiltonian* if it has a simple cycle that contains all its vertices; such a cycle is called a *Hamiltonian cycle* of $G$. Suppose that $G$ is planar and that $G$ is not Hamiltonian. One can augment $G$ to a (not necessarily planar) graph $G'$, by adding to $G$ a minimal set of dummy edges such that $G'$ contains a Hamiltonian cycle $\mathcal{H}'$ including all the dummy edges. If $G'$ is not planar, we can apply on $G'$ a planarization algorithm (see, e.g., [6]) with the constraint that only crossings between dummy edges and edges of $G - \mathcal{H}'$ are allowed. The planarization algorithm constructs an embedded planar graph $G''$ where each edge crossing is replaced with a dummy vertex. Graph $G''$ is called the *augmented Hamiltonian form of* $G$. If $G'$ is planar, the *augmented Hamiltonian form of* $G$ is $G'$ itself along with a given planar embedding. The vertices of the augmented Hamiltonian form that are not dummy vertices are called *real* *vertices*. The Hamiltonian cycle $\mathcal{H}$ of the augmented Hamiltonian form of $G$ is called an *augmenting dividing Hamiltonian cycle of* $G$; note that $\mathcal{H}$ is a subdivision of $\mathcal{H}'$ obtained by possibly splitting some edges of $\mathcal{H}'$ with dummy vertices. If every edge $e$ of $G'$ is crossed at most $c$ times, $\mathcal{H}$ is said to be an *augmenting dividing Hamiltonian cycle of* $G$ *with at most* $c$ *dummy vertices per edge*. Several different techniques have been presented in the literature to compute an augmenting dividing Hamiltonian cycle of a planar graph. The following result has been proved in [8].

**Lemma 4.** [8] *Every planar graph* $G$ *with* $n$ *vertices admits an augmenting dividing Hamiltonian cycle* $\mathcal{H}$ *with at most one dummy vertex per edge. An augmented Hamiltonian form of* $G$ *including* $\mathcal{H}$ *can be computed in* $O(n)$ *time.*

## 4.1 Drawing Algorithm

Let $G = (V, E, \phi)$ be a planar leveled graph with $n$ vertices and let $\mathcal{C}$ be a set of concentric circles such that $|\mathcal{C}| = \lambda(G) = k$ and the radius of $C_i$ is $r_i = (k-i)\cdot\Delta$, $i = 0, 1, \ldots, k-1$ and $\Delta > 0$. This choice of the circles guarantees RDR $= 1$.

Let $G''$ be an augmented Hamiltonian form of $G$ computed with the algorithm of Lemma 4 and let $\mathcal{H}$ be the Hamiltonian cycle of $G''$. Let $u_0$ be a vertex of $G''$ that is also a vertex of $G$; visit $\mathcal{H}$ counterclockwise starting at $u_0$ and let $u_0, u_1, \ldots, u_{n''-1}$ be the vertices of $G''$ in the order they are encountered during the visit. We distinguish the real vertices of $G \cap G''$ from the dummy ones by introducing a second notation for the real vertices. Vertex $u_0$ is also denoted $v_0$, vertex $v_i$ is the real vertex $u_j$ of $G \cap G''$ that is encountered after $v_{i-1}$ when visiting $\mathcal{H}$ counterclockwise.

In order to compute a drawing where ADR $= 1$ and the semantic requirement of vertex centrality is satisfied, we proceed as follows. Let $\rho_i$ ($i = 0, \ldots, n-1$) be the ray that forms an angle of $\frac{2\pi\cdot i}{n}$ with the positive $x$-axis. Each vertex $v_i$ is drawn at the intersection point $\rho_i \cap C_{\phi(v_i)}$ ($i = 0, \ldots, n-1$).

The dummy vertices of $G''$ are drawn as follows. Let $v_i$ and $v_{i+1}$ ($0 \le i \le n-1$) be two vertices of $G$ such that $v_i = u_j$ and $v_{i+1} = u_{j+h}$ ($h > 1$), i.e. two "real" vertices such that there are $h-1$ dummy vertices between them in $\mathcal{H}$. We choose $h-1$ arbitrary rays $\varrho_1, \varrho_2, \ldots, \varrho_{h-1}$ such that $\angle\rho_i < \angle\varrho_1 < \angle\varrho_2 < \cdots < \angle\varrho_{h-1} < \angle\rho_{i+1}$ (for example one can choose $h-1$ equi-spaced rays). Vertex $u_{j+l}$ is drawn at point $\overline{v_i v_{i+1}} \cap \varrho_l$ ($1 \le l \le h-1$). Note that there is no semantic requirement on the dummy vertices and they do not need to be drawn on a circle of $\mathcal{C}$. The edges of $\mathcal{H}$ are drawn as straight-line segments between their endvertices. The chosen position of the vertices implies that the drawing of $\mathcal{H}$ is a star-shaped polygon, whose kernel contains the center. In the following we denote with $\mathcal{P}_0$ the polygon representing $\mathcal{H}$. See for example Figure 3(a).

Each edge $e$ of $G''$ not belonging to $\mathcal{H}$ is either inside or outside $\mathcal{H}$ in the planar embedding of $G''$. If $e$ is inside $\mathcal{H}$, it is drawn inside $\mathcal{P}_0$ as a polyline with one bend, else it is drawn outside $\mathcal{P}_0$ as a polyline with two bends. The edges that do not belong to $\mathcal{H}$ are suitably ordered and are inserted in the drawing one at a time in increasing order. The ordering is defined as follows.

Let $e = (u_i, u_j)$ $(0 \leq i < j \leq n'' - 1)$ be an edge that is not in $\mathcal{H}$ and let $\varrho_{u_i}$ and $\varrho_{u_j}$ be the rays through $u_i$ and $u_j$, respectively. We call *span* of $e$ the angle $\alpha_e = \min\{\angle\varrho_{u_i}\varrho_{u_j}, \angle\varrho_{u_j}\varrho_{u_i}\}$. Notice that, for each edge $e$ we have $0 < \alpha_e \leq \pi$. We order the edges inside $\mathcal{H}$ by increasing span; similarly, the edges outside $\mathcal{H}$ are ordered by increasing span. Ties are broken arbitrarily.

Drawing the edges inside $\mathcal{H}$: Let $e_0, e_2, \ldots, e_{h-1}$ be the edges that are inside $\mathcal{H}$ ordered according to their span. The edges are drawn inside $\mathcal{P}_0$ so that no edge goes trough point $o$. The drawing of edge $e_0$ partitions polygon $\mathcal{P}_0$ into two sub-polygons, one of which contains $o$ and is denoted as $\mathcal{P}_1$. Edge $e_l$ is drawn inside $\mathcal{P}_l$ $(l = 1, \ldots, h-1)$ and partitions it into two sub-polygons one of which contains $o$ and is denoted as $\mathcal{P}_{l+1}$. Given one of the polygons $\mathcal{P}_l$ $(l = 0, \ldots, h-1)$, let $\mathcal{V}_l$ be the set of vertices (real or dummy) that are the endvertices of at least one edge $e_g$ with $g > l$, i.e. an edge that is not yet drawn in the drawing that defines $\mathcal{P}_l$. We say that $\mathcal{P}_l$ is *weakly star-shaped*, if all the vertices of $\mathcal{V}_l$ are visible from an internal point of $\mathcal{P}_l$. The set of points of $\mathcal{P}_l$ from which all the vertices of $\mathcal{V}_l$ are visible is called the *weak kernel* of $\mathcal{P}_l$ and will be denoted as $WKer(\mathcal{P}_l)$. When inserting edge $e_l$ inside polygon $\mathcal{P}_l$ the algorithm maintains the following invariants.

**Invariant 1.** *All vertices of $\mathcal{V}_l$ are on the boundary of $\mathcal{P}_l$.*

**Invariant 2.** *$\mathcal{P}_l$ is weakly star-shaped and the weak kernel of $\mathcal{P}_l$ contains a closed disk centered at $o$.*

We now give details about how to draw edge $e_l$ so that it has at most one bend and it does not go through $o$. Let $\overline{C}_l$ be the boundary of the closed disk contained in $WKer(\mathcal{P}_l)$ and let $e_l = (u_i, u_j)$ $(0 \leq i < j \leq n'' - 1)$. Let $\varrho_{u_i}$ and $\varrho_{u_j}$ be the rays through $u_i$ and $u_j$, respectively and let $\alpha_{e_l}$ be the span of $e_l$. We have that either $\angle\varrho_{u_i}\varrho_{u_j} = \alpha_{e_l}$ or that $\angle\varrho_{u_j}\varrho_{u_i} = \alpha_{e_l}$. Assume the first case holds (the other case is analogous). If both $u_i$ and $u_j$ are real vertices, $e_l$ is drawn as a polyline with $u_i$ and $u_j$ as endpoints and one bend at point $b_l = \varrho \cap \overline{C}_l$, where $\varrho$ that is the bisector of the angle $\angle\varrho_{u_i}\varrho_{u_j}$. See, for example, Figure 3(b). Suppose that one of the endvertices of $e_l$ –say $u_i$– is a dummy vertex (in this case $u_j$ is real by Lemma 4), we draw the bend at point $b_l = \varrho_{u_i} \cap \overline{C}_l$. See, for example, Figure 3(c).

Drawing the edges outside $\mathcal{H}$: Let $e_0, e_2, \ldots, e_{h-1}$ be the edges that are outside $\mathcal{H}$ ordered according to their span. The edges are drawn outside $\mathcal{P}_0$ as follows. Let $\varrho_{u_0}, \varrho_{u_1}, \ldots, \varrho_{u_{n''-1}}$ be the rays passing through the points representing $u_0, u_1, \ldots, u_{n''-1}$, respectively and let $\theta_{min} = \min_i\{\angle\varrho_{u_i}\varrho_{u_{i+1}}\}$. For every ray $\varrho_{u_i}$ we define two rays $\varrho^+_{u_i}$ and $\varrho^-_{u_i}$; refer to Figure 3(d). If $u_i$ is a real vertex $\varrho^+_{u_i}$ is a ray such that $\angle\varrho^+_{u_i} = \angle\varrho_{u_i} + \theta_{min}/3$ and $\varrho^-_{u_i}$ is a ray such that $\angle\varrho^-_{u_i} = \angle\varrho_{u_i} - \theta_{min}/3$. If $u_i$ is a dummy vertex, $\varrho^+_{u_i} = \varrho^-_{u_i} = \varrho_{u_i}$.

Let $e_l = (u_i, u_j)$ be the current edge to be drawn $(l = 0, \ldots, h-1)$. Let $\alpha_{e_l}$ be the span of $e_l$; we have that either $\angle\varrho_{u_i}\varrho_{u_j} = \alpha_{e_l}$ or $\angle\varrho_{u_j}\varrho_{u_i} = \alpha_{e_l}$. Assume the first case holds (the other case is analogous). Let $d_{max}$ be the maximum distance from the center $o$ of the circles to any point in the drawing computed before the addition of $e_l$, and let $C^*$ be a circle centered at $o$ and having radius $r > d_{max}$. Let $\varrho$ be the bisector of the angle $\angle\varrho_{u_i}\varrho_{u_j}$ and let $\ell_{max}$ be the straight-line

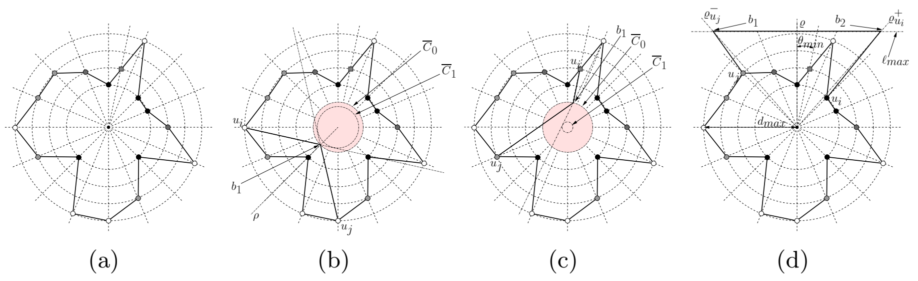


**Fig. 3.** Illustration of the drawing algorithm. (a) The star-shaped polygon $\mathcal{P}_0$ representing $\mathcal{H}$. How to draw an edge inside $\mathcal{H}$: when both endvertices are real (b) and when an endvertex is dummy (c). (d) How to draw an edge outside $\mathcal{H}$.

orthogonal to $\varrho$ passing through the point $\varrho \cap C^*$. The two bends of $e_l$ will be drawn at points $b_1 = \ell_{max} \cap \varrho_{u_i}^+$ and $b_2 = \ell_{max} \cap \varrho_{u_j}^-$. The following theorem describes the performance of the drawing algorithm both in terms of aesthetic requirements and in terms of computational complexity. The proof is omitted due to space limitation.

**Theorem 1.** *Every planar leveled graph has a radial leveled drawing whose* CURVE COMPLEXITY *is* 3 *and that is optimal in terms of both* CROSSINGS *and* RADIAL DISTRIBUTION. *Also, this drawing can be computed in $\mathcal{O}(n)$ time, where $n$ is the number of vertices of the graph.*

## 5 Curve Complexity and Radial Distribution: Trade-Offs

In this section we show how to improve the quality of the drawing in terms of number of bends per edge at the expenses of a lower quality in terms of radial distribution. In Subsection 5.1 we show that at most two bends per edge can be achieved with an optimal RDR but at the price of a suboptimal ADR; Subsection 5.2 shows how to compute radial drawings with at most one bend per edge by also loosing the optimality of RDR. We recall that Lemma 2 implies that one bend per edge may be necessary if the drawing is required to be planar.

### 5.1 Radial Drawings with No Crossings, Optimal Radial Distance Ratio, and Curve Complexity 2

We describe an algorithm to compute a planar radial drawing with RDR $= 1$ and CURVE COMPLEXITY 2. The algorithm computes the drawing by adding at each step a vertex and all its incident edges according to an ordering introduced by de Fraysseix, Pach and Pollack and known as a *canonical ordering* [5].

Let $G$ be a maximal embedded planar graph with external boundary $u$, $v$, $w$. A *canonical ordering* of $G$ with respect to $u, v$ is an ordering of the vertices $v_1 = u, v_2 = v, v_3, \ldots, v_n = w$ of $G$ with the following properties for every integer

$k$ such that $4 \leq k \leq n$: (i) The subgraph $G_{k-1} \subseteq G$ induced by $v_1, v_2, \ldots, v_{k-1}$ is biconnected and the external boundary $B_{k-1}$ of $G_{k-1}$ contains edge $(u, v)$; (ii) $v_k$ is in the external face of $G_{k-1}$, and its neighbors in $G_{k-1}$ form a subpath of the path $B_{k-1} - (u, v)$.

Let $G = (V, E, \phi)$ be a planar leveled graph with $n$ vertices and let $\mathcal{C}$ be a set of concentric circles such that $|\mathcal{C}| = \lambda(G) = k$ and the radius of $C_i$ is $r_i = (k-i) \cdot \Delta$, $i = 0, 1, \ldots, k-1$ and $\Delta > 0$. This choice of the circles guarantees RDR $= 1$. Given a point $p$ in the plane $x(p)$ and $y(p)$ denote the $x$- and $y$-coordinate of $p$, respectively. Also, we denote by $\ell_p$ the straight line $\{(x(p), y) \mid y \in \mathbb{R}\}$ and by $\ell_p^+$ the half-line $\{(x(p), y) \mid y \geq y(p), y \in \mathbb{R}\}$.

The first step of the algorithm draws vertex $v_1$ at the point of coordinates $(-r_{\phi(v_1)}, 0)$. At the second step, $v_2$ is drawn at the point of coordinates $(r_{\phi(v_2)}, 0)$ and edge $(v_1, v_2)$ is drawn as a polyline with one bend of coordinates $(0, -y)$, where $0 \leq y \leq r_{k-1}$. At Step 3, vertex $v_3$ is drawn as the point of coordinates $(0, r_{\phi(v_3)})$ and edges $(v_1, v_3)$ and $(v_2, v_3)$ are drawn as polylines each having one bend. The bend of $(v_1, v_3)$ has coordinates $(-x, y)$, where $0 \leq x, y \leq r_{k-1}$, while the bend of $(v_2, v_3)$ has coordinates $(x, y)$, where $0 \leq x, y \leq r_{k-1}$. At the generic Step $i$ ($i = 4, \ldots, n$) the algorithm adds vertex $v_i$ to the drawing $\Gamma_{i-1}$ of the graph $G_{i-1}$ induced by vertices $v_1, v_2, \ldots, v_{i-1}$. Let $B_i$ be the external boundary of $G_i$ and let $\Pi_i$ be the path obtained by removing edge $(v_1, v_2)$ from $B_i$. The following invariants are maintained (see also Figure 4(a)):

**Invariant A.** *Path $\Pi_i$ is drawn as an $x$-monotone polygonal chain in $\Gamma_i$.*

**Invariant B.** *Every edge $e = (w_0, w_1)$ of $\Pi_i$ is drawn in the half-plane $y > 0$ and as a polyline with at least one bend. Let $w_0$ be encountered before $w_1$ when visiting $\Pi_i$ from $v_1$ to $v_2$. Let $b$ be the leftmost bend of $e$. The half-line $\ell_b^+$ intersects all circles of $\mathcal{C}$ and does not share any point with $\Gamma_i$, except $b$.*

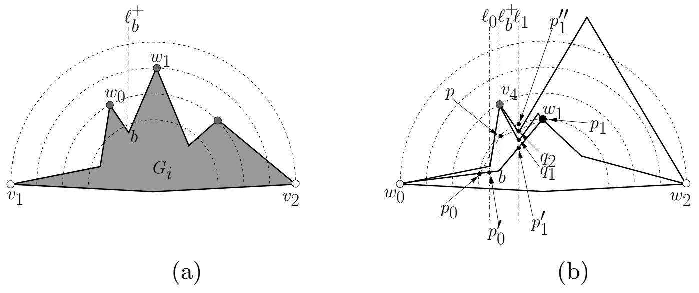


**Fig. 4.** Illustrations for the drawing algorithm that computes radial drawings with no crossing, optimal radial distance ratio, and curve complexity 2. (a) An example of Invariant B. (b) An example of Step $i$ of the drawing algorithm.

The addition of $v_i$ to $\Gamma_{i-1}$ at Step $i$ is computed as follows. Refer to Figure 4(b). Let $w_0, w_1, \ldots, w_{h-1}$ be the vertices on $\Pi_{i-1}$ that are adjacent to $v_i$ and assume that they are encountered in this order when visiting $\Pi_{i-1}$ from $v_1$

to $v_2$. Let $b$ be the leftmost bend of $(w_0, w_1)$. Draw vertex $v_i$ at point $\ell_b^+ \cap C_{\phi(v_i)}$, denote by $p_0$ the point $\overline{bw_0} \cap C_{k-1}$, by $p_1$ the point $\overline{bw_1} \cap C_{k-1}$, and by $p$ the point $\ell_b^+ \cap C_{k-1}$. Let $\ell_0$ be any vertical straight line that intersects segment $\overline{p_0 p}$ and let $p_0' = \ell_0 \cap (w_0, w_1)$. Edge $(w_0, v_i)$ is drawn as a polyline with one bend located at point $(x(p_0'), y)$, with $y(p_0') < y < y(\ell_0 \cap C_{k-1})$. Let $\ell_1$ be any vertical straight line that intersects segment $\overline{pp_1}$, let $p_1' = \ell_1 \cap (w_0, w_1)$ and $p_1'' = \ell_1 \cap C_{k-1}$. Choose $h-1$ points, $q_1, q_2, \ldots, q_{h-1}$, on segment $\overline{p_1' p_1''}$ with $y(q_j) < y(q_{j+1})$. Each edge $(v_i, w_j)$ $(j = 1, \ldots, h-1)$ is drawn as a polyline with two bends; the first bend is at point $q_j$. Let $\sigma$ be the maximum value such that there exists a segment of $\Pi_{k-1}$ with slope either $\sigma$ or $-\sigma$. Choose $h-1$ positive values $\sigma_j$ $(1 \leq j \leq h-1)$ such that $\sigma_{h-1} > \sigma_{h-2} > \cdots > \sigma_1 > \sigma$. The second bend of $(v_i, w_j)$ is placed at the intersection point between the straight line with slope $\sigma_j$ passing trough $q_j$ and the straight line with slope $-\sigma_j$ passing trough $w_j$ $(1 \leq j \leq h-1)$. The proof of the following theorem is omitted for reasons of space.

**Theorem 2.** *Every planar leveled graph with $n$ vertices has a radial leveled drawing whose* CURVE COMPLEXITY *is* 2 *and that is optimal in terms of* RDR *and* CROSSINGS. *Also, this drawing can be computed in $\mathcal{O}(n)$ time.*

## 5.2 Radial Drawings with No Crossings and Curve Complexity 1

It has been proved that every planar graph admits a planar drawing such that all the vertices are drawn on a semi-circle and the CURVE COMPLEXITY is 1 [8]. We call a drawing with these properties a *circle drawing*. Let $\Gamma$ be a circle drawing and let $\overline{p_1 q_1}$ and $\overline{p_2 q_2}$ be two segments of $\Gamma$ that are chords of the semi-circle $C$ hosting the vertices of $\Gamma$ and such that points $p_1$, $p_2$, $q_2$, and $q_1$ are encountered in this order moving clockwise on $C$. We say that segments $\overline{p_2 q_2}$ is *nested* inside segment $\overline{p_1 q_1}$. Segments $\overline{p_1 q_1}$ and $\overline{p_2 q_2}$ are *consecutive nested segments* if $\overline{p_2 q_2}$ is nested inside segment $\overline{p_1 q_1}$ and there is no other segment $\overline{p_3 q_3}$ such that $\overline{p_2 q_2}$ is nested inside $\overline{p_3 q_3}$ and $\overline{p_3 q_3}$ is nested inside $\overline{p_1 q_1}$. Let $\overline{p_1 q_1}$ and $\overline{p_2 q_2}$ be two consecutive nested segments, let $d_a$ be the distance from $p_2$ to $\overline{p_1 q_1}$ and let $d_b$ be the distance from $q_2$ to $\overline{p_1 q_1}$; the *distance between $\overline{p_1 q_1}$ and $\overline{p_2 q_2}$* is the minimum value between $d_a$ and $d_b$.

To compute a planar radial drawing of a planar leveled graph $G = (V, E, \phi)$ with CURVE COMPLEXITY 1, we first construct a circle drawing $\Gamma$ by using the algorithm in [8]. Let $r$ be the radius of the semi-circle $C$ used in $\Gamma$, and let $d$ be the minimum distance between two consecutive nested segments. We choose a set of $k = \lambda(G)$ circles $C_0, C_1, \ldots, C_{k-1}$ that are concentric with $C$ and such that the radius of circle $C_i$ is $r_i = r - (i+1)\frac{\delta}{k}$, where $\delta < d$. Starting from $\Gamma$, we move each vertex $v \in V$ to the point $\varrho_v \cap C_{\phi(v)}$, where $\varrho_v$ is the ray passing through $v$, and we leave the bends at the same locations they have in $\Gamma$. The proof of the following theorem is omitted due to space limitation.

**Theorem 3.** *Every planar leveled graph with $n$ vertices has a radial leveled drawing whose* CURVE COMPLEXITY *is* 1 *and that is optimal in terms of* CROSSINGS. *Also, this drawing can be computed in $\mathcal{O}(n)$ time.*

## 6 Open Problems

We list three open problems related to the subject of this paper: (i) Based on the negative result of Lemma 1, it would be interesting to characterize the family of graphs that admit a radial leveled drawing that is optimal in terms of CROSSINGS and CURVE COMPLEXITY and RADIAL DISTRIBUTION. (ii) Can one compute a radial drawing that satisfies vertex centrality, is optimal in terms of CROSSINGS and of RADIAL DISTRIBUTION, and has CURVE COMPLEXITY less than 3? (iii) We focused on three aesthetic requirements for radial drawings. Studying other aesthetic requirements and trade-offs can be a promising research direction.

## References

1. C. Bachmaier, F. J. Brandenburg, and M. Forster. Track planarity testing and embedding. In *Proc. of SOFSEM'04*, volume 2, pages 3–17, 2004.
2. C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time. *JGAA*, 9(1):53–97, 2005.
3. U. Brandes, P. Kenis, and D. Wagner. Communicating centrality in policy network drawings. *IEEE Trans. on Vis. and Comp. Graph.*, 9(2):241–253, 2003.
4. U. Brandes and D. Wagner. Visone - analysis and visualization of social networks. In M. Jünger and P. Mutzel, editors, *Graph Drawing Software*, Springer Verlag, pages 321–340. 2004.
5. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
6. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, 1999.
7. E. Di Giacomo, W. Didimo, G. Liotta, and H. Meijer. Computing radial drawings on the minimum number of circles. *JGAA*, 9(3):365–389, 2005.
8. E. Di Giacomo, W. Didimo, G. Liotta, and S. K. Wismath. Curve-constrained drawings of planar graphs. *Computational Geometry*, 30:1–23, 2005.
9. M. Dodge and R. Kitchin. *Atlas of Cyberspace*. Addison Wesley, 2001.
10. S. N. Dorogstev and J. F. F. Mendes. *Evolution of Networks, From Biological Nets to the Internet and WWW*. Oxford University Press, 2003.
11. M. Kaufmann and D. Wagner, editors. *Drawing Graphs*, volume 2025 of *LNCS*. Springer, 2001.
12. M. Kaufmann and R. Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *Journal of Graph Algorithms and Applications*, 6(1):115–129, 2002.
13. J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graph and Combinatorics*, 17:717–728, 2001.
14. H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In *Proc. GD '97*, volume 1353 of *LNCS*, pages 248–261. Springer, 1998.
15. H. C. Purchase. Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, 13(2):147–162, 2000.
16. K. Sugiyama. *Graph Drawing and Applications*. World Scientific, 2002.
17. R. Tamassia. Advances in the theory and practice of graph drawing. *Theoretical Computer Science*, 217(2):235–254, 1999.
18. R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst., Man and Cyber.*, SMC-18(1):61–79, 1988.

# Characterization of Unlabeled Level Planar Trees

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

Department of Computer Science, University of Arizona
{aestrell,jfowler,kobourov}@cs.arizona.edu

**Abstract.** Consider a graph $G$ drawn in the plane so that each vertex lies on a distinct horizontal line $\ell_j = \{(x, j) \mid x \in \mathbb{R}\}$. The bijection $\phi$ that maps the set of $n$ vertices $V$ to a set of distinct horizontal lines $\ell_j$ forms a *labeling* of the vertices. Such a graph $G$ with the labeling $\phi$ is called an *n-level graph* and is said to be *n-level planar* if it can be drawn with straight-line edges and no crossings while keeping each vertex on its own level. In this paper, we consider the class of trees that are $n$-level planar regardless of their labeling. We call such trees *unlabeled level planar* (ULP). Our contributions are three-fold. First, we provide a complete characterization of ULP trees in terms of a pair of forbidden subtrees. Second, we show how to draw ULP trees in linear time. Third, we provide a linear time recognition algorithm for ULP trees.

## 1 Introduction

When drawing an $n$-vertex planar graph $G(V, E)$ in the $xy$-plane, a more restrictive form of planarity can be obtained by insisting on a predetermined $y$-coordinate for each vertex. In particular, suppose we have a set of $k$ equidistant horizontal lines or *levels*, namely $\ell_j = \{(x, j) \mid x \in \mathbb{R}\}$ for $j \in \{1, 2, \ldots, k\}$ and each vertex is assigned to one of these $k$ levels. Call this level assignment $\phi$. The tuple $G(V, E, \phi)$ forms a *k-level graph*, and if $\phi$ is bijective so that each vertex is constrained to its own level, i.e., $k = n$, then $G(V, E, \phi)$ is an *n-level graph*. Further, suppose that when drawing $G$, each edge is a straight-line segment (or a continuous $y$-monotone polyline). If a planar drawing of $G$ can be obtained in spite of these restrictions, then $G$ is said to be *level planar* for level assignment $\phi$. If $G$ is an $n$-level graph that is level planar, then we say $G$ is *n-level planar*.

Some level assignments of $G$ do not allow for a level planar drawing. In fact, if $k < n$, then it is NP-hard [9] to determine whether there even exists a $k$-level assignment of $G$ in which $G$ is level planar. If $k = n$, in which $G$ is an $n$-level graph, such a level assignment gives a *labeling* of $V$ since each vertex in $V$ is uniquely numbered. A labeling of $V$ whose level assignment preserves the planarity of $G$ can be easily obtained from a plane drawing of $G$ and a perturbation

[⋆] Partially supported by CONACYT-México.
[⋆⋆] This work is supported in part by NSF grants CCF-0545743 and ACR-0222920.

of the vertices to ensure unique $x$-coordinates [2]. We call a $n$-level tree that is $n$-level planar for all possible labelings of its vertices an *Unlabeled Level Planar* (ULP) tree[1]. We characterize ULP trees in terms of a pair of forbidden subtrees and provide linear time recognition and drawing algorithms.

## 1.1 Background and Related Work

Visualizing hierarchical relationships has historically been a strong motivating factor in the study of the planarity of *level graphs*, i.e., graphs with a predetermined level assignment. Geometric simultaneous embedding has recently led to a new application of $n$-level graphs [2]. Determining which sets of graphs have geometric simultaneous embeddings has proven difficult. For instance, it is unknown whether a path and a tree can aways be simultaneously embedded. When simultaneously embedding an $n$-vertex path $P$ with another $n$-vertex graph $G$, one can relabel the vertices of $P$ sequentially 1 to $n$ from one endpoint to the other. The corresponding relabeling of the vertices of $G$ gives a natural level assignment $\phi$ for $G$. Eades *et al.* [4] have provided an $O(|V|)$ time algorithm for drawing any level planar graph with straight-line segments. Thus, if $G$ is $n$-level planar for $\phi$, it can easily be simultaneously embedded with $P$ since the path merely zig-zags in a $y$-monotone fashion from one level to the next. The ability to characterize $n$-level graphs gives additional insight into open problems in simultaneous embedding.

Jünger *et al.* [11] provide a linear time recognition algorithm for level planar graphs. This is based on the level planarity test given by Heath and Pemmaraju [7,8], which in turn extends the more restricted PQ-tree level planarity testing algorithm of *hierarchies*—level graphs of DAGs in which all edges are between adjacent levels and all the source vertices are on the uppermost level—given by Di Battista and Nardelli [3]. Hierarchies are characterized in terms of level non-planar (LNP) patterns in [3] as well. Jünger and Leipert [10] further provide a linear time level planar embedding algorithm that outputs a set of linear orderings in the $x$-direction for the vertices on each level. However, to obtain a straight-line planar drawing one needs to subsequently run a $O(|V|)$ algorithm given by Eades *et al.* [4] who demonstrate that every level planar embedding has a straight-line drawing. Healy *et al.* [6] use LNP patterns to provide a set of minimum level non-planar subgraph patterns that characterize level planar graphs. These subgraph patterns are somewhat analogous to Kuratowski's result that any minimal non-planar graph is either a subdivided $K_5$ or $K_{3,3}$ [12]. It should be noted that these patterns are specific to a given level assignment and are not based solely on the underlying graph.

## 1.2 Our Contribution

Our contributions are three-fold. First, we provide a forbidden subdivision characterization for unlabeled level planar (ULP) trees in terms of two minimal ULP trees, $T_1$ and $T_2$; see Fig. 1.

[1] A more appropriate name for these types of trees might be "unlabeled $n$-level planar trees" but for simplicity we call them unlabeled level planar or ULP trees.

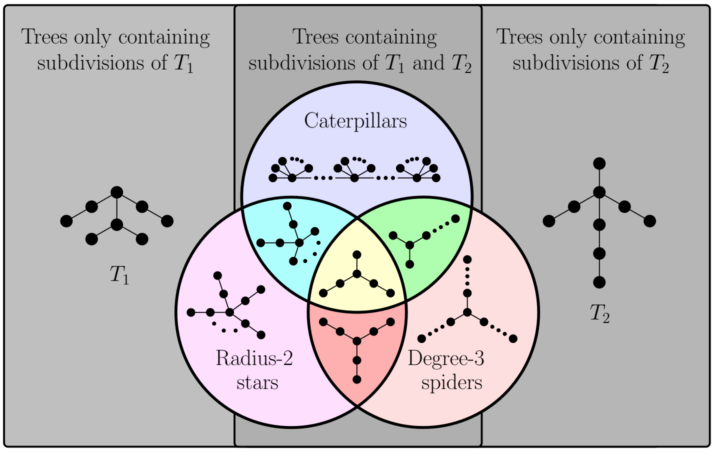

Trees only containing subdivisions of $T_1$
Trees containing subdivisions of $T_1$ and $T_2$
Trees only containing subdivisions of $T_2$
Caterpillars
$T_1$
$T_2$
Radius-2 stars
Degree-3 spiders

**Fig. 1.** A Venn diagram for the universe of trees characterized by the two forbidden subtrees $T_1$ and $T_2$. Graphs that do not contain either a subdivision of $T_1$ or $T_2$ are caterpillars, radius-2 stars, and degree-3 spiders.

Second, we characterize any tree without a subdivision of either $T_1$ or $T_2$ as either (i) a *caterpillar*, a tree in which the removal of all degree-1 vertices yields a path, or (ii) a *radius-*2 *star*, a $K_{1,k}$ in which every edge is subdivided at most once, or (iii) a *degree-*3 *spider*, an arbitrary subdivision of $K_{1,3}$. We show that these three classes are $n$-level planar by virtue of an $O(|V|)$ time algorithm for constructing straight-line $n$-level planar drawings of ULP trees.

Third, if a tree is not a caterpillar, then it must contain a *lobster* (a graph in which the removal of all degree-1 vertices yields a caterpillar). Using minimal lobsters we show that trees that are not radius-2 stars or degree-3 spiders must contain subdivisions of $T_1$ or $T_2$, which completes the characterization. We also provide a $O(|V|)$ time algorithm for testing whether a tree falls into one of these three categories, thus yielding a linear time recognition algorithm for ULP trees.

## 2 Preliminaries

In this paper we try to use the established notation for level graphs whenever possible. The following definitions for levels graphs are predominantly taken from [1,3,6]. A *k-level graph* $G(V, E, \phi)$ on $n$ vertices is a DAG with a level assignment $\phi : V \rightarrow [1..k]$ such that the induced partial order is strict: $\phi(u) < \phi(v)$ for every $(u, v) \in E$. A $k$-level graph is a $k$-partite graph in which $\phi$ partitions $V$ into $k$ independent sets $V_1$, $V_2$, ..., $V_k$, which form the $k$ *levels* of $G$. A *level-j* vertex $v$ is on the $j^{th}$ *level* $V_j$ of $G$ if $\phi(v) = j$ where $V_j = \phi^{-1}(j)$.

If $\phi$ is an injection, each level contains at most one vertex, i.e., $|V_i| \leq 1$ for $i \in [1..k]$, hence, $k \geq n$. W.l.o.g., we can assume in such instances that $k = n$ in which case $\phi$ is a bijection that forms a topological sort of the DAG $G(V, E)$. Unless noted otherwise, an $n$-level graph $G(V, E, \phi)$ is assumed to have a bijective level assignment $\phi$, i.e., $\phi : V \xrightarrow[\text{onto}]{1:1} [1..n]$. Such a bijective level assignment is equivalent to a *labeling* of the vertices from 1 to $n$.

A level graph $G$ has a *level drawing* if there exists a drawing such that every vertex in $V_j$ is placed along the horizontal line $\ell_j = \{(x, j) \mid x \in \mathbb{R}\}$ and the edges are drawn as strictly $y$-monotone polylines. The order that the vertices of $V_j$ are placed along each $\ell_j$ in a level drawing induces a family of linear orders $(\leq_j)_{1 \leq j \leq k}$ along the $x$-direction, which form a *linear embedding* of $G$. A level drawing, and consequently its level embedding, is *level planar* if it can be drawn without edge crossings. A level graph $G$ is level planar if it admits a level planar embedding. The more restrictive definition of level drawings allowing only straight-line segments for edges is equivalent, as shown by Eades *et al.* [4]. A planar graph $H$ is *realize*d if it can be drawn with straight-line edges without crossings. Such a plane graph is a *realization* of $H$. A $n$-level graph $G(V, E, \phi)$ is $n$*-level realize*d if it is *realize*d such that each vertex $v$ lies on its level $\phi(v)$.

A *chain* of a $k$-level graph $G(V, E, \phi)$ is a nonrepeating sequence of vertices $v_1, v_2, \ldots, v_t$ of $V$ such that $t > 1$ and either $(v_i, v_{i+1}) \in E$ or $(v_{i+1}, v_i) \in E$ for every $i \in \{1, 2, \ldots, t-1\}$, i.e., a path in the underlying undirected graph of $G$. If $C$ is a chain, let $\textsc{lower}(C)$ and $\textsc{upper}(C)$ denote the lowermost and uppermost vertices, respectively, with respect to $\phi$. Also let $\textsc{min}(C) = \phi\big(\textsc{lower}(C)\big)$ and $\textsc{max}(C) = \phi\big(\textsc{upper}(C)\big)$ be the minimum and maximum levels of $C$, thus, $\textsc{min}(C) \leq \phi(v) \leq \textsc{max}(C)$ for every $v$ of $C$.

For an $n$-level graph $G(V, E, \phi)$, let $<_Y$ denote the strict linear ordering given by the level assignment $\phi$, i.e., for every $u, v \in V$, we have $u <_Y v$ iff $\phi(u) < \phi(v)$. Let $<_X$ (and $\leq_X$) denote the strict (and weak) linear ordering induced by the $x$-coordinate of the placement of a level-$i$ vertex $u$ and a level-$j$ vertex $v$ along their respective horizontal lines $\ell_i$ and $\ell_j$ in the particular level drawing under consideration. Finally, for vertex subsets $U, W \subseteq V$, let $U <_X W$ (and $U <_Y W$) iff $u <_X w$ (and $u <_Y w$) for every $u \in U$ and $w \in W$. Often we will represent an edge $(u, v) \in E$ as $u - v$ and a chain of vertices, $v_1, v_2, \ldots, v_t$ for some $t > 1$ as $v_1 - v_2 - \cdots - v_t$.

Finally, we recall a few standard graph theory definitions. In a graph $G(V, E)$, *subdividing* an edge $(u, v) \in E$ is the operation of replacing $(u, v)$ with the pair of edges $(u, w)$ and $(w, v)$ in $E$ by adding $w$ to $V$. A *subdivision* of $G$ is a graph obtained by performing a series of successive edge subdivisions of $G$.

## 3 Characterization of Unlabeled Level Planar Trees

First, we introduce the forbidden subdivisions $T_1$ and $T_2$ together with explicit level assignments in which the resulting graphs are level non-planar. Then we show how to compute a $n$-level realization for each of the three remaining types of trees—caterpillars, radius-2 stars and degree-3 spiders—in linear time given a labeling of the vertices. Next, we show that if a tree does not contain a subdivision of $T_1$ or $T_2$, then it must fall into at least one of the three categories of unlabeled level planar trees. Finally, we give a simple $O(|V|)$ time recognition algorithm for ULP trees. Full details are included in the technical report [5].

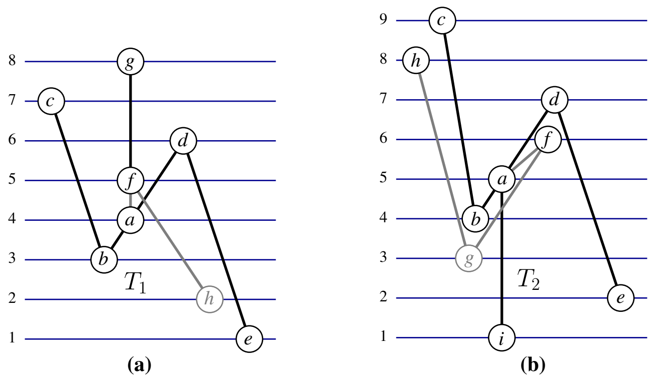


**Fig. 2.** Labelings that prevent $T_1$ and $T_2$ from being ULP

## 3.1 Forbidden Trees

**Lemma 1.** *There exist labelings that prevent $T_1$ and $T_2$ from being level planar.*

*Proof.* Fig. 2(a) gives one of 8 distinct labelings that satisfies the $y$-partial order $\{c,\, g\} >_Y d >_Y f >_Y a >_Y b >_Y \{e,\, h\}$ (or its dual in which the ordering is reversed). Each labeling gives a bijective $n$-level assignment in which $T_1$ does not have a $n$-level realization. This can be seen as follows: To prevent paths $a{-}b{-}c$ and $a{-}d{-}e$ from crossing, one path must go to the left and the other to the right. Assume w.l.o.g. $c{-}b <_X d{-}e$. This forces $c <_X f <_X d$ so that $a{-}f$ does not cross $b{-}c$, or $a{-}f{-}g$ does not cross $a{-}d{-}e$. However, then $f{-}h$ will cross either $a{-}d$ or $a{-}b{-}c$. This concludes the argument for $T_1$.

Next, consider $T_2$. Fig. 2(b) gives one of 8 distinct labelings that satisfies the $y$-partial order $\{c,\, h\} >_Y d >_Y f >_Y a >_Y b >_Y\ g >_Y \{e,\, i\}$ (or its dual in which the ordering is reversed). Each labeling gives a bijective $n$-level assignment in which $T_2$ does not have a $n$-level realization. This can be seen as follows: To prevent paths $a{-}b{-}c$ and $a{-}d{-}e$ from crossing, one path must go to the left and the other to the right. Assume again w.l.o.g. $c{-}b <_X d{-}e$. Then $a{-}i$ must be drawn below and to the right of $a{-}b$ and to the left of $d{-}e$, otherwise it will cross $b{-}c$ or $d{-}e$. To prevent the edge $a{-}f$ from crossing $a{-}b{-}c$ or $a{-}d{-}e$, $f$ must with be drawn (i) so that $a <_X f <_X e$ in which case $f{-}g{-}h$ will then cross $a{-}i$ or $d{-}e$, or (ii) so that $c <_X f <_X d$ in which case $f{-}g$ will cross $a{-}b{-}c$, $a{-}d$, or $a{-}i$. This completes the argument about $T_2$ and the overall claim. □

**Corollary 2.** *If a tree $T(V, E)$ contains a subdivision of $T_1$ or $T_2$, then it cannot be unlabeled level planar.*

*Proof.* Assume that the tree $T$ contains a subdivision of $T_1$ (or $T_2$). Let $T'(V', E')$ be a subtree of $T$ that is a subdivision of $T_1$ (or $T_2$). Label the 8 (or 9) vertices of $V'$ in the same order as shown in Fig. 2. Note that the values of the labels need to be adjusted in order to accommodate any intermediate vertices along a subdivided edge of $T_1$ (or $T_2$). Any extra vertex $w$ along a subdivided edge $(u, v)$

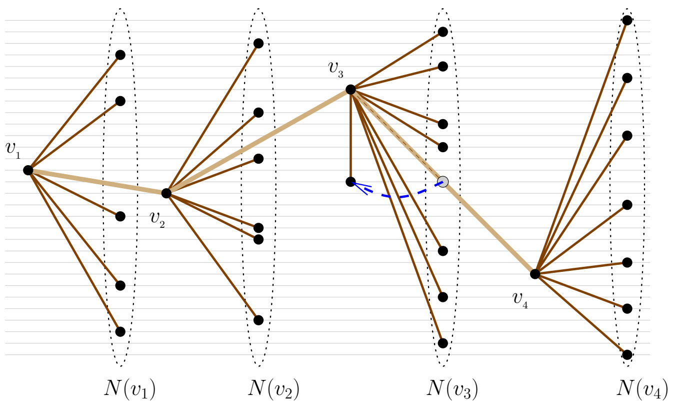


**Fig. 3.** A $n$-level realization of a 30-level caterpillar on a $8 \times 30$ grid

can be assigned to a unique level that preserves the $y$-ordering $u <_Y w <_Y v$. This works since level planarity is defined in terms of $y$-monotone polylines. An edge drawn from level $\phi(u)$ to level $\phi(v)$ is allowed any number of bends so long as the edge proceeds in a $y$-monotone fashion. In particular, it can bend at $w$ on level $\phi(w)$, which is equivalent to subdividing the edge $u$—$v$ into $u$—$w$—$v$.

This gives a labeling of the vertices of $T'$ using the labels $\{1, 2, \ldots, |V'|\}$ such that the 8 (or 9) vertices corresponding to $T_1$ (or $T_2$) satisfy one of the $y$-partial orders from Lemma 1. Hence, the arguments used in Lemma 1 can be directly applied to this $y$-ordering. Thus, $T'$ is not $n$-level planar, and as a consequence, neither is $T$, regardless of the labels for the remaining vertices. □

## 3.2 ULP Trees—Caterpillars, Radius-2 Stars, and Degree-3 Spiders

The following three lemmas explicitly show all the trees that are unlabeled level planar and how to $n$-level realize them in linear time.

**Lemma 3.** *(Brass* et al. *[2]) An $n$-vertex caterpillar $T(V, E)$ with an $m$-vertex spine can be $n$-level realized in $O(n)$ time on a $2m \times n$ grid for any vertex labeling $\phi : V \xrightarrow[onto]{1:1} \{1, 2, \ldots, n\}$.*

*Proof.* The following proof of the claim is a shorter version and is an improvement over the original proof in [3] that spans 3 pages. Note that the original claim had the slightly weaker result of using a $2n \times n$ grid.

Let $T(V, E, \phi)$ be an $n$-level caterpillar with spine $S(V', E')$ such that $S$ is isomorphic to $P_{|V'|}$. In particular, let the vertices of $V'$ be labeled according to their relative distance from the end point $v_1$, and the edges $E' = \big\{(v_1, v_2), (v_2, v_3), \ldots, (v_{|V'|-1}, v_{|V'|})\big\}$. Let the degree-1 leaves of $v$ be denoted by $N(v) = \big\{u \,|\, (u, v) \in E \text{ and } (u, v) \notin E'\big\}$ for each $v \in V'$. Then for each $i \in \{1, 2, \ldots, n\}$ place $v_i \in V'$ at the coordinate $\big(2i - 1, \phi(v_i)\big)$ and place each $u \in N(v_i)$ at the coordinate $\big(2i, \phi(u)\big)$ unless $u$ would lie on the straight-line edge segment

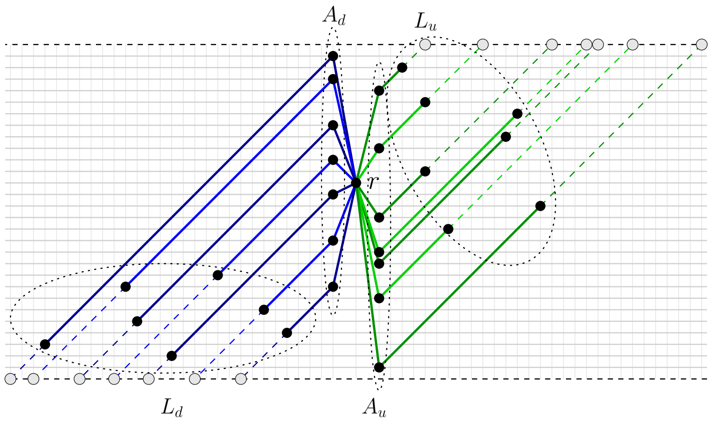


**Fig. 4.** A $n$-level realization of an 29-level radius-2 star on a $61 \times 29$ grid. The small gray circles are the intersection points of slope-1 rays emanating from each vertex in $A_d$ and $A_u$ to the imagined 0-level and imagined $(n+1)$-level, respectively.

$v_i$–$v_{i+1}$ in which case place $u$ directly under $v_i$ at the coordinate $\big(2i-1, \phi(u)\big)$ instead. All this can be done in $O(n)$ time. This drawing is an $n$-level planar since $S$ is drawn in a strictly left to right fashion and each incident edge to the spine is either drawn either directly above or below the spine or immediately to its right. Clearly, this drawing uses only straight-line edge segments in which there are no crossings forming a $n$-level realization. □

**Lemma 4.** *An $n$-vertex radius-2 star $T(V, E)$ can be $n$-level realized in $O(n)$ time on a $(2n+3) \times n$ grid for any vertex labeling $\phi : V \xrightarrow[onto]{1:1} \{1, 2, \ldots, n\}$.*

*Proof.* Let $r$ be the root of an $n$-level radius-2 star $T(V, E, \phi)$ located at the coordinate $\big(n+2, \phi(r)\big)$. For every leaf $\ell$ that is at a distance of 1 from r, place it at the coordinate $\big(n+1, \phi(\ell)\big)$, which is one $x$-coordinate to the left of $r$. For each remaining leaf $\ell$, let $adj(\ell)$ denote its adjacent vertex, and let $L \subseteq V$ denote this set of leaves at a distance 2 from $r$. Then $L_d = \big\{\ell \mid \ell \in L \text{ and } \phi(adj(\ell)) > \phi(\ell)\big\}$ and $L_u = \big\{\ell \mid \ell \in L \text{ and } \phi(adj(\ell)) < \phi(\ell)\big\}$ partition $L$ according to whether the adjacent vertex of the leaf is to be drawn above or below it, i.e., whether the incident edge goes down or up. Let $A_d = \big\{adj(\ell) \mid \ell \in L_d\big\}$ and $A_u = \big\{adj(\ell) \mid \ell \in L_u\big\}$ be the adjacent vertices of degree 2 to the leaf vertices of $T$.

Place each $u \in A_d$ at the coordinate $\big(n+1, \phi(u)\big)$ immediately to the left of $r$, and each $u \in A_u$ at the coordinate $\big(n+3, \phi(u)\big)$, immediately to the right of $r$. For each $\ell \in L_d$, place it at the grid point that corresponds to the intersection of the $\phi(\ell)$-level and the line segment connecting the points $\big(n+1, \phi(adj(\ell))\big)$, the coordinate of its adjacent vertex, and $\big(n - \phi(adj(\ell)) + 1, 0\big)$, the point that an emanating ray from $adj(\ell)$ with a slope of 1 in the *negative* $x$-direction intersects an imagined 0-level; see Fig. 4. Since the ray has slope 1, this intersection will always be an integer grid point. In a similar fashion, place each $\ell \in L_u$ at the grid point that corresponds to the intersection of the $\phi(\ell)$-level and the line segment connecting the points $\big(n+3, \phi(adj(\ell))\big)$, the coordinate of its adjacent vertex, and $\big(n-\phi(adj(\ell))+1, n+1\big)$, the point that an emanating ray from $adj(\ell)$ with a slope of 1 in the *positive* $x$-direction intersects an imagined $(n+1)$-level. All this can be done in linear time since $O(1)$ time is spent locating each vertex.

This produces a $n$-level realization since every vertex that is adjacent to $r$ is either placed immediately to its right or left, and every other leaf is placed so that its incident edge has a slope of 1, which prevents any edge from crossing. □

**Lemma 5.** *An $n$-vertex degree-3 spider $T(V, E)$ can be $n$-level realized in $O(n)$ time for any vertex labeling $\phi : V \xrightarrow[onto]{1:1} \{1, 2, \ldots, n\}$.*

*Proof.* The proof is in three parts. First, we show how to reduce an arbitrary degree-3 spider to one in which all the legs zig-zag between successively lower and higher levels. Second, we skip ahead and show how to place the vertices of the original spider when processing an edge of the reduced spider. Finally, we show an $O(n)$ time algorithm for greedily drawing the reduced degree-3 spider.

**Part 1:** Let $r$ be the root vertex of an $n$-level degree-3 spider $T(V, E, \phi)$. Let $X$, $Y$, and $Z$ be the three subtrees of $r$, each of which forms a chain. Call $T'(V', E', \phi')$ a *strictly expanding* degree-3 spider if the level assignment $\phi'$ on the vertices $r, v_2, \ldots, v_{|C|}$ of each chain $C$ of $T'$ obeys the following two properties:

$$\phi(v_{i-1}) < \phi(v_i) > \phi(v_{i+1}) \ \text{ or } \phi(v_{i-1}) > \phi(v_i) < \phi(v_{i+1}), \tag{1}$$

and

$$\big[\phi(v_{i-1}) < \phi(v_i) \Rightarrow \phi(v_{i-1}) > \phi(v_{i+1})\big] \ \text{ and }$$
$$\big[\phi(v_{i-1}) > \phi(v_i) \Rightarrow \phi(v_{i-1}) < \phi(v_{i+1})\big] \tag{2}$$

for $1 < i < |C|$. We call a chain that satisfies property (1) a *zig-zagging chain* since it cannot have any monotonically increasing or decreasing sequences of

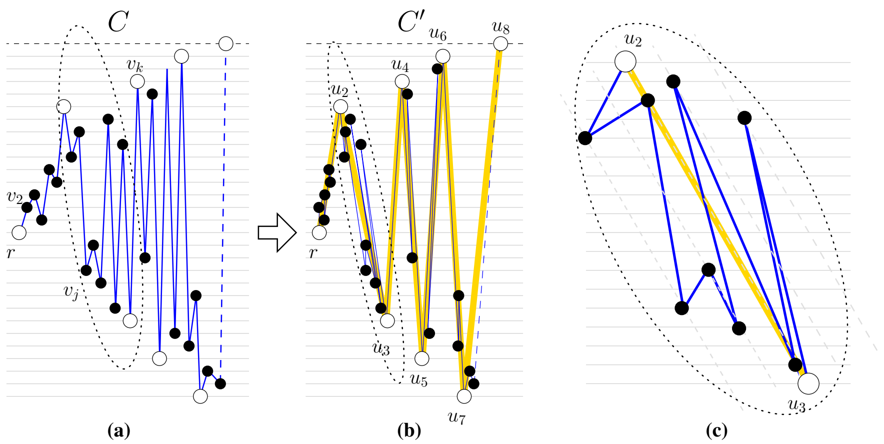


**Fig. 5.** An example of a chain $C$ of (a) in which the strictly expanding zig-zagging subchain $C'$ of white vertices is extracted to give (b). Then (c) shows an example $n$-level realization of the intermediate edges and vertices for the second edge of (b).

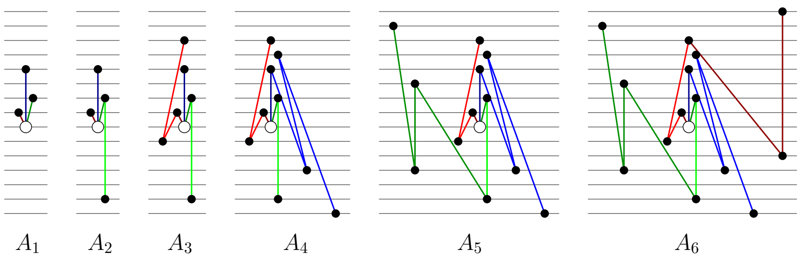


**Fig. 6.** Six iterations of the greedy strategy to $n$-level realize a degree-3 spider

vertices. A zig-zagging chain that also satisfies property (2) is *strictly expanding* as the next level reached by the chain is either greater than any previous level or less than any previous level.

A zig-zagging chain $C$ can be made strictly expanding by keeping track of the minimum and maximum levels encountered by the chain so far. Assume w.l.o.g. that $\phi(r) < \phi(v_2)$, i.e., the chain $C$ begins by going upwards. We extract from $C$, the strictly expanding subchain $C'$, which we will label its vertices as $r - u_2 - u_3 - \cdots - u_{C'}$, by first prepending $r$ to $C'$. Then we set $min_C = \phi(r)$ and find the first vertex $v_j$ along $C$ such that $\phi(v_j) < min_C$. Next we append $u_2 = \text{UPPER}(r - v_2 - v_3 - \cdots - v_{j-1})$ to $C'$, and set $max_C = \phi(u_2)$. Then we look for the next vertex $v_k$ along $C$ such that $\phi(v_k) > max_C$. Afterward, we append $u_3 = \text{LOWER}(v_j - v_{j+1} - \cdots - v_{k-1})$ to $C'$, and set $min_C = \phi(u_3)$, and repeat this process until all the vertices of $C$ are exhausted. If the last vertex encountered is not greater than $min_C$ or less than $max_C$, then we add an extra vertex to the end of $C'$ satisfying this condition; see Fig. 5 for an illustration.

**Part 2:** For each vertex $u_i$ in $C'$, we keep a linked list of the subpath $P = u_i - w_{i_1} - w_{i_2} - \cdots - w_{i_{|P|-2}} - u_{i+1}$ of $C$ that was replaced by the edge $u_i - u_{i+1}$ where $\text{MIN}(P) = \phi(u_i)$ and $\text{MAX}(P) = \phi(u_{i+1})$ (or $\text{MAX}(P) = \phi(u_i)$ and $\text{MIN}(P) = \phi(u_{i+1})$). This linked list will be used to place edges of $T$ as we process edges from $T'$. Fig. 5(c) visually illustrates how this might be done. Here, any particular subpath $P$ of $C$ for a given edge of $C'$ can be drawn arbitrary close to $C$. We omit the details of this particular point, noting only that the intuitive idea of compressing the zig-zagging chain allows us to greedily draw the edges of $T$ without crossings for each edge of $T'$ that is processed. We finish this section of the proof by observing that both the extraction of $C'$ from $C$ *and* the ability to draw the edges of $T$ once we have processed the edges of $T'$ can be done in linear time.

**Part 3:** Now that we have our degree-3 spider in the proper form, we can apply a simple greedy algorithm that can be used to give a $n$-level realization of $T'$. We complete the proof of the lemma by giving the details regarding this linear time algorithm.

Let the vertices of chain $X$ be denoted by $x_0 - x_1 - \cdots - x_{|X|-1}$ in which $x_0 = r$ and $(x_i, x_{i+1}) \in E$ for $0 \leq i < |X| - 2$. Similarly, let the vertices of $Y$ and $Z$ be

$y_0 - y_1 - \cdots - y_{|Y|-1}$ and $z_0 - z_1 - \cdots - z_{|Z|-1}$. Finally, let $A_1 = \{x_1, y_1, z_1\}$ be the first vertices along each chain immediately following $r$.

There exist two possibilities for the strictly expanding degree-3 spider $T'$: Either (i) $A_1 >_Y r$ (or $A_1 <_Y r$) or (ii) $\textsc{min}(A_1) <_Y r <_Y \textsc{max}(A_1)$. We show how to draw $T$ for case (i) assuming that $A_1 >_Y r$. The other case is similar. We start the first iteration by drawing $A_1$ so that the vertex of maximum index with respect to $\phi'$ lies between the other two vertices of $A_1$ along the $x$-coordinate.

At any one point in this greedy strategy, we maintain the invariant that the last vertex along a chain that we placed either lies above or below any of the other vertices that have been drawn so far. Property (2) allows us to do this. If we encounter the end of a chain in which this invariant does not hold for its last vertex, then we can easily draw the remaining two chains without crossings. We do this by drawing one of the two chains monotonically to the right until we reach its end, and do the same for the other chain monotonically to the left.

For iteration $i > 1$, we arbitrarily pick one of the chains whose most recently placed vertex is neither the maximum nor the minimum vertex drawn so far. We greedily extend the chain either to the right or left until we reach a vertex whose level assignment is either above or below all the ones drawn so far. This enlarges the set of processed vertices from $A_{i-1}$ to $A_i$. Note that we can always extend a chain $C$ to the right or left. This follows from the fact that during the previous iteration, *before the vertices of some other chain $C'$ were processed*, the last vertex $v$ of $C$ was either minimum or maximum; see Fig. 6 for an example.

Since we can always greedily place a vertex without introducing a crossing, this strategy succeeds in producing a $n$-level realization of $T$ in $O(n)$ time (constant time per vertex), which shows that $T$ is indeed ULP. Simple geometry can be used to construct such an drawing using only straight-line edge segments for $T'$, which can be used to produce a plane drawing of $T$ as detailed above. □

Now that we have shown which trees are ULP, we need to show that our characterization is complete. First, we show that $T_1$ and $T_2$ are minimal unlabeled level non-planar trees with the following lemma.

**Lemma 6.** *Removing any edge from $T_1$ or $T_2$ yields a forest of ULP trees.*

*Proof.* If removing an edge from $T_1$ decreases the degree of one of the two degree-3 vertices, call them $x$ and $y$, then the resulting graph is a forest consisting of a degree-3 spider and a possible lone edge; see Fig. 7(a). Removing the edge $x - y$ yields two paths. The only possibility (up to isomorphism) in removing an edge without affecting the degree of $x$ and $y$, yields a caterpillar with a spine of length 5. Moving onto $T_2$, if its vertex $z$ of degree 4 maintains its degree after the edge removal, then the resulting graph must be a forest consisting of either a caterpillar, if the removed edge was incident to a leaf vertex at a distance 2 from $z$, or a radius-2 star and a possible lone edge, otherwise. On the other hand, if the degree of $z$ decreases to 3, then the resulting graph is a degree-3 spider and, possibly, a path; see Fig. 7(c). □

The next theorem completes the characterization of ULP trees.

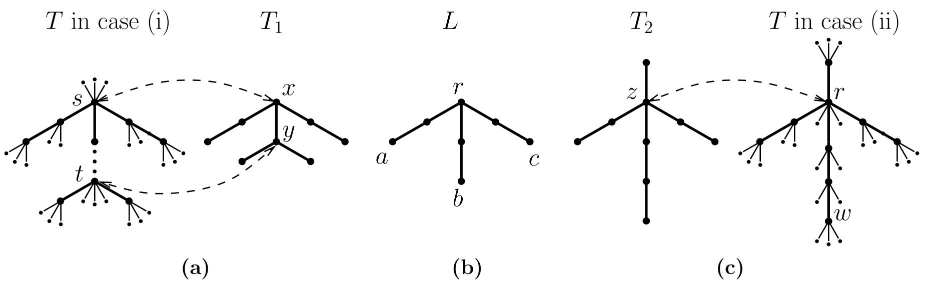


**Fig. 7.** Finding copies of $T_1$ and $T_2$

**Theorem 7.** *Every tree either contains a subdivision of* $T_1$ *or* $T_2$ *in which case it is not* ULP*, or it is a caterpillar, a radius-*2 *star, or a degree-*3 *spider in which case it is* ULP*. Hence,* $T_1$ *and* $T_2$ *give a minimal forbidden subtree characterization of* ULP *trees.*

*Proof.* First, we argue that neither $T_1$ nor $T_2$ is a caterpillar, a radius-2 star, or a degree-3 spider to show that if a tree $T$ contains a subdivision of $T_1$ or $T_2$, it cannot be one of those three. Clearly, both $T_1$ and $T_2$ are lobsters and not caterpillars. The two vertices of degree 3 prevent $T_1$ being a radius-2 star or a degree-3 spider. Since $T_2$ has radius 3 and a vertex of degree 4, it cannot be a radius-2 star or a degree-3 spider either.

Now we show that caterpillars, radius-2 stars, and degree-3 spiders are the only types of ULP trees. We do this by showing that any tree that does not fit into at least one of these categories must contain either a subdivision of $T_1$ or $T_2$. Then by Corollary 2 any tree $T$ that contains a subdivision of $T_1$ or $T_2$ cannot be ULP. By Lemma 6, $T_1$ and $T_2$ are minimal.

Assume that $T(V,E)$ is a tree that is not a caterpillar, radius-2 star, or degree-3 spider. Since $T$ is not a caterpillar, it must contain a minimal lobster $L$, i.e., the unique tree that cannot have any more edges removed without becoming a caterpillar (and possibly a lone edge); see Fig. 7(b). It has one vertex $r$ of degree 3 and three leaf vertices $a$, $b$, $c$ at a distance 2 from $r$, which is the minimal requirement for a tree to be a lobster. Any other lobster can have its edges trimmed away until $L$ is all that remains, which is what makes $L$ minimal.

Since $T$ is not a degree-3 spider, there are two cases to consider: either (i) $T$ has two vertices $s$ and $t$ of degree at least 3 or (ii) $T$ has one vertex of degree $k$ greater than 3.

Assuming case (i) holds, we show how to find a subdivision of $T_1$ in $T$. Let $x$ and $y$ be the two vertices of degree 3 in $T_1$ where $x$ is the one without an adjacent leaf vertex; see Fig. 7(a). At least one of the two vertices $s$ and $t$ of degree at least 3 in $T$ must correspond to the root vertex $r$ in the subtree $L$ that forms the minimal lobster in $T$. Assume w.l.o.g. this vertex is $s$. Then we map $s$ in $T$ to vertex $x$ in $T_1$, and the other vertex of degree at least 3, $t$ in $T$ to vertex $y$ in $T_1$. Since $t$ has degree at least 3, there exists two neighbors of $t$ not along the path from $s$ to $t$, which we can map to the two corresponding leaf vertices in $T_1$ that are adjacent to $y$. Only one of the three leaf vertices $a$, $b$, $c$ of $L$ in $T$ can be contained in the subtree of $s$ containing $t$. Suppose w.l.o.g. it is $a$. Then the other two vertices $b$ and $c$ in $T$ can be mapped to the two remaining leaf vertices in $T_1$. This completes the mapping of vertices of $T_1$.

Next we consider case (ii) in which we show how to find the subtree $T_2$ in $T$. The one vertex of degree $k$ greater than 3 must be the corresponding vertex $r$ of $L$ in $T$; see Fig. 7(c). Otherwise, if there were separate vertices of degree greater than 3, case (i) would apply. Let $r$ be mapped to the degree-4 vertex $z$ of $T_2$. Since $T$ is not a radius-2 star, there exists a vertex $w$ at a distance 3 from $r$, which can be mapped to the leaf vertex in $T_2$ at a distance 3 from $z$. Only one of the three vertices $a$, $b$, $c$ of $L$ in $T$ can be along the path from $r$ to $w$. Suppose w.l.o.g. it is $a$. The other two vertices $b$ and $c$ in $T$ can be mapped to the other two leaf vertices in $T_2$. The remaining leaf vertex of $T_2$ that is directly adjacent to $z$ can be mapped to the endpoint of the fourth edge incident to $r$ in $T$ since it has degree greater than 3. This completes the mapping of vertices of $T_2$. □

## 3.3 Linear Time Recognition of ULP Trees

First, we need a few simple observations regarding the degree sequences of caterpillars, radius-2 stars, and degree-3 spiders, which we state as lemmas whose proofs we omit in this abstract.

**Lemma 8.** *If a tree $T$ has a degree sequence of the form $2, \ldots, 2, 1, 1$ or $1, 1$, i.e., a path, after the removal of all degree-1 vertices, then $T$ must be a caterpillar.*

**Lemma 9.** *If a tree $T$ has a degree sequence of the form $k, 2, \ldots, 2, 1, 1, \ldots 1$ for some $k > 2$, i.e., $T$ is an arbitrarily subdivided $K_{1,k}$, and after the removal of all degree-1 vertices, the degree sequence then becomes $\ell, 1, \ldots, 1$ for some $\ell \leq k$, i.e., $T$ becomes a $K_{1,\ell}$, then $T$ must be a radius-2 star.*

**Lemma 10.** *If a tree $T$ has a degree sequence of the form $3, 2, \ldots, 2, 1, 1, \ldots 1$, i.e., $T$ has maximum degree of 3 with only one vertex of degree 3, then $T$ must be a degree-3 spider.*

Theorem 7 together with the above Lemmas, lead to a simple linear time recognition algorithm for ULP trees summarized in the following corollary:

**Corollary 11.** *The class of ULP trees can be recognized in linear time. That is, given an arbitrary $n$-vertex tree $T$, one can decide in $O(n)$ time whether or not it is always possible to $n$-level realize $T$ for any possible labeling.*

## 4 Conclusion and Future Work

We described a complete characterization of unlabeled level planar trees. We provided a linear time algorithm to $n$-level realize the three classes of ULP trees which can also be used for simultaneously embedding a ULP tree $T$ with any path $P$. Finally, we provided a linear time recognition algorithm for ULP trees. What is missing from the recognition algorithm is a certificate of unlabeled level non-planarity, i.e., the 8 (or 9) vertices corresponding $T_1$ (or $T_2$) if they exist.

Another future task is to provide a forbidden subgraph characterization for general unlabeled level planar graphs as we have done for ULP trees.

## References

1. C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time. *J. Graph Algorithms Appl.*, 9(1):53–97, 2005.
2. P. Brass, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. B. Mitchell. On simultaneous graph embedding. In *8th Workshop on Algorithms and Data Structures*, pages 243–255, 2003.
3. G. Di Battista and E. Nardelli. Hierarchies and planarity theory. *IEEE Trans. Systems Man Cybernet.*, 18(6):1035–1046 (1989), 1988.
4. P. Eades, Q. Feng, X. Lin, and H. Nagamochi. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. *Algorithmica*, 44(1):1–32, 2006.
5. A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. Characterization of ulabeled planar trees. Technical Report TR06-03, University of Arizona, 2006.
6. P. Healy, A. Kuusik, and S. Leipert. A characterization of level planar graphs. *Discrete Math.*, 280(1-3):51–63, 2004.
7. L. S. Heath and S. V. Pemmaraju. Recognizing leveled-planar dags in linear time. In *4th Symposium on Graph Drawing (GD)*, pages 300–311. 1996.
8. L. S. Heath and S. V. Pemmaraju. Stack and queue layouts of directed acyclic graphs. II. *SIAM J. Comput.*, 28(5):1588–1626, 1999.
9. L. S. Heath and A. L. Rosenberg. Laying out graphs using queues. *SIAM J. Comput.*, 21(5):927–958, 1992.
10. M. Jünger and S. Leipert. Level planar embedding in linear time. *J. Graph Algorithms Appl.*, 6:no. 1, 67–113, 2002.
11. M. Jünger, S. Leipert, and P. Mutzel. Level planarity testing in linear time. In *6th Symposium on Graph Drawing (GD)*, pages 224–237. 1998.
12. C. Kuratowski. Sur les problèmes des courbes gauches en Topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

# Drawing Bipartite Graphs on Two Curves*

Emilio Di Giacomo, Luca Grilli, and Giuseppe Liotta

Dipartimento di Ingegneria Elettronica e dell'Informazione
Università degli Studi di Perugia
{digiacomo, liotta, luca.grilli}@diei.unipg.it

**Abstract.** Let $G$ be a bipartite graph, and let $\lambda_e, \lambda_i$ be two parallel convex curves; we study the question about whether $G$ admits a planar straight line drawing such that the vertices of one partite set of $G$ lie on $\lambda_e$ and the vertices of the other partite set lie on $\lambda_i$. A characterization is presented that gives rise to linear time testing and drawing algorithms.

## 1 Introduction

Common requirements for drawing a bipartite graph are that the bipartition is highlighted in the visualization by representing the vertices on two distinct layers, the edges have as few bends as possible, and the number of edge crossings is minimized. A bipartite graph is a *biplanar graph* if it has a straight line crossing-free drawing where the vertices of the same partite set all lie along one of the horizontal layer [5]. Biplanar graphs have been independently characterized in [4,6,8]. Also, the problem of computing straight-line drawings of bipartite graphs with the vertices on two horizontal layers and minimum number of crossings has been well studied; see, e.g. [2,7] for some basic references on this topic.

This paper studies planar drawings of bipartite graphs where vertices are constrained to be on two parallel convex curves, which generalizes the case of horizontal layers. Let $G$ be a bipartite graph, and let $\lambda_e, \lambda_i$ be two parallel convex curves; we want to answer the question about whether $G$ admits a planar straight line drawing such that the vertices of one partite set of $G$ lie on $\lambda_e$ and the vertices of the other partite set lie on $\lambda_i$.

Our interest in this question is in part motivated by the observation that the class of bipartite graphs that admit a planar straight line drawing on two horizontal lines is quite restricted and that one may hopefully enlarge this class by allowing some curvature on the two layers. Indeed, there is already some evidence in the literature that if the vertices in a drawing are not constrained to be collinear but instead can lie on curves, the family of representable graphs for specific drawing conventions can increase significantly; see, e.g. [3] for drawings of planar graphs with at most one bend per edge and vertices constrained to be on a given curve.

The problem addressed in this paper is also related to the study of *radial planarity testing* initiated by Bachmaier, Brandenburg and Forster [1]. In [1] the

* Research partially supported by MIUR under Project "ALGO-NEXT".

input is a $k$-partite graph $G$ and $k$-concentric circles; the question is whether $G$ has a crossing-free drawing where the vertices of the same partite set are points of the same radial level (circle) and the edges are simple Jordan curves in the outward direction. Here, we study radial planarity testing for bipartite graphs with the additional constraint that the edges are straight-line segments (indeed, two concentric circles are a special case of two parallel convex curves).

Our contribution is as follows. The family of bipartite graphs which admit a planar straight-line drawing with the vertices constrained to be on two parallel convex curves and with no two vertices of the same partite set on different curves is characterized. The characterization gives rise to a linear time testing algorithm. The proof of sufficiency uses a linear time (real RAM) drawing algorithm.

## 2 Preliminaries

A graph $G = (V, E)$ is *bipartite* if there exists a partition $V = V_0 \cup V_1$ of the vertices of $G$ such that $E \subseteq V_0 \times V_1$. The two sets $V_0$ and $V_1$ are called *partite sets* of $G$. A bipartite graph with a given planar embedding is *maximal* if every internal face of $G$ consists of four edges.

A simple curve $\lambda$ in the Euclidean plane is a *closed* curve if it partitions the plane into two topologically connected regions; $\lambda$ is an *open* curve otherwise. Curve $\lambda$ is *convex* if any straight line intersects $\lambda$ in at most two points. Note that a circle is a special case of closed convex curve.

Let $p, q$ be two distinct points of $\lambda$. If $\lambda$ is an open curve we say that $p$ *precedes* $q$ on $\lambda$ if $p$ is encountered before $q$ when traversing $\lambda$ in the clockwise direction. If $\lambda$ is a closed curve, let $p$ and $q$ be two distinct points of $\lambda$ such that the portion of $\lambda$ traversed when going from $p$ to $q$ in the clockwise direction is shorter than the portion of $\lambda$ traversed when going from $q$ to $p$; we say that $p$ *precedes* $q$ and that $q$ *follows* $p$ on $\lambda$.

Two convex curves are *parallel* if every normal to one curve is also a normal to the other curve and the distance between the points where the normals intersect the two curves is a constant. In the rest of this paper we denote with $\lambda_e, \lambda_i$ two parallel convex curves such that the curvature of $\lambda_e$ is less than the curvature of $\lambda_i$; $\lambda_e$ is the *external curve*, $\lambda_i$ is the *internal curve* (in the special case of two concentric circles, $\lambda_e$ is the circle with larger radius). Curves $\lambda_e, \lambda_i$ are *paired* if there exist two points $p \in \lambda_e$ and $q \in \lambda_i$ such that the straight-line segment $\overline{pq}$ intersects $\lambda_i$ twice. A straight-line segment with the property of $\overline{pq}$ is said to *cross* curve $\lambda_i$. Observe that two concentric circles are paired. Two curves will be called *non-paired* if they are parallel, convex, but are not paired.

Let $\lambda_e, \lambda_i$ be two parallel convex curves. A bipartite graph $G$ is *curve biplanar* on $\lambda_e, \lambda_i$ if it admits a *curve biplanar drawing*, i.e. a planar straight-line drawing such that all vertices of a bipartite set of $G$ are represented as points on $\lambda_e$ and the vertices of the other bipartite set are represented as points on $\lambda_i$. As the next theorem shows, if $\lambda_e, \lambda_i$ are not paired, the family of curve biplanar graphs on two non-paired curves coincides with the family of biplanar graphs characterized in [4,6,8]. The proof is an easy adaptation of the arguments in [4,6,8] and it has been omitted for space reasons. A graph is a *caterpillar* if deleting all vertices of degree one produces a (possibly empty) path.

**Theorem 1.** *A bipartite graph admits a curve biplanar drawing on two non-paired curves if and only if it is a forest of caterpillars.*

Motivated by Theorem 1, we will investigate the family of bipartite graphs that admit a curve biplanar drawing on two paired curves. We first show how to draw a specific family of graphs, namely *bipartite fans*, and then present a complete characterization of curve biplanar graphs on two paired curves.

## 3 How to Draw a Bipartite Fan

Let $G$ be a biconnected bipartite graph with a given planar embedding. $G$ is a *bipartite fan* if it has a vertex $u$, called *apex*, that is shared by all its faces (including the external one). The edges incident on $u$ are the *radial edges* of the fan. Let $u, v_0, v_1, \ldots, v_{n-2}$ be the vertices of a fan $G$ in the clockwise order they have on the external face. Edges $(u, v_0)$ and $(u, v_{n-2})$ are called *first edge* and *last edge* of the fan, respectively. Any three vertices $v_{2j}, v_{2j+1}, v_{2j+2}$ $(0 \leq j \leq \frac{n-4}{2})$ form a *fan triplet* of $G$. Notice that $v_{2j+1}$ belongs to the same partite set as $u$.

We show how to compute a curve biplanar drawing of a bipartite fan on two paired curves such that the drawing is contained in a suitable region of the plane called a *wedge* and defined as follows. Let $\lambda_e, \lambda_i$ be two paired curves, let $p, q, r$ be three points such that: (i) $p, r \in \lambda_e$ and $p$ precedes $r$ on $\lambda_e$, (ii) $q \in \lambda_i$, (iii) segment $\overline{pq}$ does not cross curve $\lambda_i$, (iv) segment $\overline{qr}$ crosses $\lambda_i$. Let $\lambda_{pr}$ be the portion of $\lambda_e$ consisting of all points $x \in \lambda_e$ such that $x$ follows $p$ and precedes $r$. The closed bounded region delimited by $\overline{pq}$, $\overline{qr}$ and $\lambda_{pr}$ is a wedge of $\lambda_e, \lambda_i$ and is denoted as $W(p, q, r)$.

**Lemma 1.** *Let $G$ be a bipartite fan with $n$ vertices and apex $u$. Let $\lambda_e, \lambda_i$ be two paired curves and let $W(p, q, r)$ be a wedge of $\lambda_e, \lambda_i$. Fan $G$ admits a curve biplanar drawing on $\lambda_e, \lambda_i$ contained inside $W(p, q, r)$ such that: (i) The first and the last edge of $G$ are represented by segments $\overline{pq}$ and $\overline{qr}$, respectively; (ii) For every fan triplet $v_{2j}, v_{2j+1}, v_{2j+2}$ of $G$ $(0 \leq j \leq \frac{n-2}{2})$, the three points representing the triplet define a wedge of $\lambda_e, \lambda_i$.*

*Sketch of Proof:* We assume that $G$ is maximal, i.e. that every internal face consists of four edges; if not, we can split each internal face $f$ having more than four edges by connecting $u$ to all vertices of $f$ that are not adjacent to $u$ and do not belong to the same partite set of $u$ (it is immediate to see that the resulting augmented graph is still a bipartite fan). Let $u, v_0, v_1, \ldots, v_{n-2}$ be the vertices of fan $G$ in the clockwise order they have on its external face.

In what follows refer to Figure 1 for an illustration. The apex of the fan $u$ is drawn at point $q$; vertices $v_0$ and $v_{n-2}$ are drawn at points $p$ and $r$, respectively. Thus, the first edge of $G$ is represented by segment $\overline{pq}$ and the last edge of $G$ is represented by segment $\overline{qr}$. Let $x$ be the point where segment $\overline{qr}$ crosses $\lambda_i$.

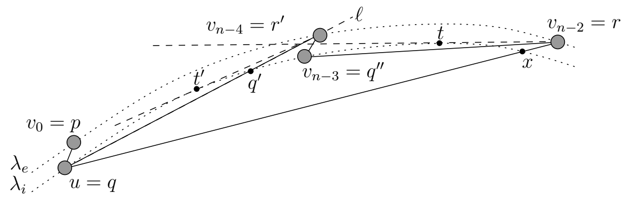


**Fig. 1.** An illustration of the drawing technique for a bipartite fan

For the convexity of $\lambda_i$, the straight line through $r$ and tangent to $\lambda_i$ intersects $\lambda_i$ at a point $t$ such that $t$ follows $q$ and precedes $x$ on $\lambda_i$; since $\overline{pq}$ does not cross $\lambda_i$, then $t$ is inside wedge $W(p,q,r)$. If $n = 4$, i.e. $G$ is a 4-cycle, choose a point $q'$ that follows $q$ and precedes $t$ on $\lambda_i$ and such that segment $\overline{pq'}$ does not cross $\lambda_i$. Draw $v_1$ at point $q'$. Since $q'$ precedes $t$, segment $\overline{q'r}$ crosses $\lambda_i$, while segment $\overline{pq'}$ does not cross $\lambda_i$ and therefore points $p, q'$, and $r$, which represent the vertices of the fan triplet $v_0, v_1, v_2$, define a wedge $W(p,q',r)$.

If $n > 4$ choose a point $q'$ that follows $q$ and precedes $t$ on $\lambda_i$ and let $\ell$ be the straight line through $q$ and $q'$. Draw vertex $v_{n-4}$ at point $r' = \ell \cap \lambda_e$. Draw vertex $v_{n-3}$ at any point $q''$ of $\lambda_i$ such that $q''$ follows $q'$ and precedes $t$ on $\lambda_i$. Segment $\overline{q''r}$ crosses $\lambda_i$ because $q''$ precedes $t$ on $\lambda_i$ and $\lambda_i$ is convex. Let $t'$ be the point where the tangent to $\lambda_i$ through $r'$ intersects $\lambda_i$; $t'$ follows $q$ and precedes $q'$ on $\lambda_i$. It follows that $t'$ precedes $q''$ and hence segment $\overline{r'q''}$ does not cross $\lambda_i$. Therefore points $r', q'', r$, which represent the vertices of the fan triplet $v_{n-4}, v_{n-3}, v_{n-2}$, define a wedge $W(r', q'', r)$. In order to complete the drawing of $G$, we observe that segment $\overline{qr'}$ crosses $\lambda_i$, and therefore points $p, q$, and $r'$ define a wedge $W(p,q,r')$. We recursively draw the fan obtained from $G$ after removing vertices $v_{n-3}$ and $v_{n-2}$ inside wedge $W(p,q,r')$. □

## 4 Curve Biplanar Graphs

We start with a sufficient condition whose proof uses the following definition. Let $G = (V,E)$ be a connected graph. A subset of vertices $S \subset V$ is a *cut-set* if the removal of $S$ disconnects $G$. Let $G_0, \ldots, G_{k-1}$ be the connected components of $G - S$ (possibly isolated vertices). The *$S$-components* of $G$ are the subgraphs of $G$ induced by sets $V(G_j) \cup S$ $(0 \leq j \leq k-1)$.

**Lemma 2.** *Let $G$ be a biconnected bipartite graph with a given planar embedding such that all vertices in one partite set belong to the external face. Then $G$ is curve biplanar on two paired curves.*

*Sketch of Proof:* We describe now how to compute an embedding preserving curve biplanar drawing of $G$. To this aim, we decompose it into subgraphs that are bipartite fans and draw each fan by using the technique described in Lemma 1.

Let $V_0$ and $V_1$ be the two partite sets of $G$ and assume that all vertices of $V_0$ belong to the external face of $G$ in the given embedding. Since $G$ is bipartite, there exists a vertex $u \in V_1$ such that $u$ belongs to the external face of $G$. Let $F_u$ be the subgraph of $G$ induced by all vertices that share an internal face with $u$ ($F_u$ exists because $G$ is biconnected). Note that $F_u$ is a bipartite fan, which we call *the fan of* $u$. Let $\lambda_e$ and $\lambda_i$ be two paired curves, let $W(p, q, r)$ be an arbitrarily chosen wedge of $\lambda_e, \lambda_i$ (such a wedge always exists because $\lambda_e$ and $\lambda_i$ are paired). By Lemma 1, $F_u$ can be drawn inside $W(p, q, r)$ so that its first and last edge are represented by segments $\overline{pq}$ and $\overline{qr}$, respectively.

Let $u, v_0, v_1, \ldots, v_{n-2}$ be the vertices of $F_u$ in the clockwise order they have on the external face of $F_u$. Since $u \in V_1$, vertices $v_{2j}$ $(j = 0, 1, \ldots, \frac{n-2}{2})$ belong to $V_0$ and are on the external face of $G$. This implies that every fan triplet $\tau = \{v_{2j}, v_{2j+1}, v_{2j+2}\}$ is a cut-set for $G$ unless $v_{2j+1}$ is on the external face. Indeed, since $G$ is biconnected, the boundary of its external face is a cycle $C$, and if $v_{2j+1} \in \tau$ is not on the external face, then the vertices of $\tau$ induce a path that splits $C$ in two paths $C'$ and $C''$; since no edge of $G$ can connect a vertex of $C'$ to a vertex of $C''$ (otherwise $G$ would not be planar), then $\tau$ is a cut-set also for $G$. The $\tau$-component of $G$ that does not contain $u$ is a planar bipartite graph that satisfies the condition expressed by the statement and, by Lemma 1, points $p', q', r'$ representing the vertices of $\tau$ in the drawing of $F_u$ define a wedge $W(p', q', r')$. The $\tau$-component of $G$ that does not contain $u$ can be recursively drawn inside $W(p', q', r')$. □

**Theorem 2.** *A bipartite graph $G$ is curve biplanar on two paired curves if and only if it admits a planar embedding such that all vertices in one partite set belong to the external face. Also, if $G$ is curve biplanar on two paired curves, a curve biplanar drawing of $G$ on two paired curves can be computed in $O(n)$ time in the real RAM model of computation, where $n$ is the number of vertices of $G$.*

*Sketch of Proof:* Sufficiency. Let $V_0$ be a partition set of $G$ such that all vertices of $V_0$ belong to the external face of a planar embedding of $G$. If $G$ is biconnected, the sufficiency follows from Lemma 2. Otherwise, $G$ can be augmented by adding dummy vertices and edges such that the augmented graph $G'$ is biconnected and bipartite, one of its partition sets is $V_0$, and has a planar embedding with the vertices of $V_0$ on the external face (for reasons of space we do not describe in more detail this augmentation technique). It follows that $G'$ has a biplanar drawing on two paired curves by Lemma 2 and hence $G$ is curve biplanar on two paired curves.

Necessity. Let $\Gamma$ be a curve biplanar drawing of a graph $G$ on two paired curves $\lambda_e$ and $\lambda_i$. All vertices drawn as points of $\lambda_e$ are on the external face of $\Gamma$ because the curves are convex and the drawing is straight-line. Since all vertices on the same curve are in the same partite set, $G$ admits a planar embedding such that all vertices in one of the partite set belong to the external face.

Time complexity. The augmentation technique can be performed in time proportional to the number of biconnected components of $G$. The fan $F_u$ of $u$ can be computed in time proportional to the number $n_u$ of vertices in $F_u$. It can be proved that the drawing of $F_u$ can be computed in $O(n_u)$ time with the technique of Lemma 1 if the real RAM model of computation is adopted. Since each vertex belongs to at most three fans, the time complexity is $O(n)$. □

**Theorem 3.** *Let $G$ be a bipartite planar graph with $n$ vertices. The curve biplanarity of $G$ on two paired curves can be tested in $O(n)$ time.*

*Sketch of Proof:* A curve biplanarity test can be performed by executing at most two planarity tests with the additional constraint that all vertices in one of the partition sets of $G$ belong to the external face. □

A $k$-partite graph $G = (V_0, \ldots, V_{k-1}, E)$ is *radial $k$-level planar* if it admits a planar drawing on $k$ concentric circles $C_0, \ldots, C_{k-1}$, with the vertices of partite set $V_j$ drawn on circle $C_j$ $(0 \leq j \leq k-1)$ and the edges drawn as strictly monotone curves from inner to outer circles. A linear time algorithm for radial planarity testing and embedding is presented in [1]. Theorems 2 and 3 make it possible to specialize radial 2-level planarity testing to the case that the edges are straight-line segments.

**Corollary 1.** *A bipartite graph $G$ is radial 2-level planar with straight-line edges if and only if it admits an embedding such that all vertices in one partite set belong to the external face. Also, there exists an $O(n)$-time algorithm that tests whether a bipartite graph $G$ is radial 2-level planar with straight-line edges.*

## 5 Open Problems

- Extend the study to $k$-partite graphs and $k$ parallel curves with $k > 2$. In particular, it would be interesting to study radial planarity testing with straight-line edges and more than two concentric circles.
- Study the complexity of the following problem: Let $G$ be a planar bipartite graph and let $c$ be a positive integer. Does $G$ have a curve biplanar subgraph (not necessarily induced) with at least $c$ edges?
- Study the complexity of the edge crossing minimization problem for straight-line drawings of bipartite graphs on two parallel convex curves.

## References

1. C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time. *J. Graph Algorithms Appl.*, 9(1):53–97, 2005.
2. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing.* Prentice Hall, Upper Saddle River, NJ, 1999.
3. E. Di Giacomo, W. Didimo, G. Liotta, and S. K. Wismath. Curve-constrained drawings of planar graphs. *Comput. Geom.: Theory and Appl.*, 30:1–23, 2005.
4. P. Eades, B. D. McKay, and N. C. Wormald. On an edge crossing problem. In *Proc 9th Austr. Comp. Sci. conf.*, pages 327–334. Australian National University, 1986.
5. P. Eades and S. H. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 131(2):361–374, 1994.
6. F. Harary and A. Schwenk. A new crossing number for bipartite graphs. *Utilitas Mathematica*, 1:203–209, 1972.
7. M. Kaufmann and D. Wagner, editors. *Drawing Graphs*, volume 2025 of *Lecture Notes in Computer Science.* Springer-Verlag, 2001.
8. N. Tomii, Y. Kambayashi, and S. Yajima. On planarization algorithms of 2-level graphs. Technical Report EC77-38, Inst. of Elect. and Comm. Eng. Japan, 1977.

# Improved Circular Layouts

Emden R. Gansner and Yehuda Koren

AT&T Labs — Research
Florham Park, NJ 07932, USA
{erg,yehuda}@research.att.com

**Abstract.** Circular graph layout is a drawing scheme where all nodes are placed on the perimeter of a circle. An inherent issue with circular layouts is that the rigid restriction on node placement often gives rise to long edges and an overall dense drawing. We suggest here three independent, complementary techniques for lowering the density and improving the readability of circular layouts. First, a new algorithm is given for placing the nodes on the circle such that edge lengths are reduced. Second, we enhance the circular drawing style by allowing some of the edges to be routed around the exterior of the circle. This is accomplished with an algorithm for optimally selecting such a set of externally routed edges. The third technique reduces density by coupling groups of edges as bundled splines that share part of their route. Together, these techniques are able to reduce clutter, density and crossings compared with existing methods.

## 1 Introduction

Circular layouts are among the most prominent and oldest conventions used to draw graphs. In such layouts, nodes are drawn on a circle, while the edges connecting these nodes are line segments passing within the circle, e.g., Figure 1(a). This drawing convention is often used for the layout of networks and systems management diagrams, where it naturally captures the essence of ring and star topologies. It can be also used for other kinds of graphs, such as social networks and WWW graphs. In particular, a circular layout is appropriate for applications that emphasize the clustering decomposition of a graph, where each cluster is drawn on a separate circle. Much work [1,4,12,13,17,18,20] has been done on these layouts, most of it addressing both the layout of a single circle as well as positioning multiple circles together in order to show the various clusters composing the full graph. Here we concentrate on the former. Circular layouts are highly regularized – nodes placed on a circle – achieving a very clear depiction of each individual node. A node cannot be occluded by another node or by an edge. Moreover, since it is impossible to have three collinear nodes, the problem of two edges obscuring each other is avoided. In general, these layouts can provide a compact presentation, focusing on individual nodes and edges. Additionally, well-designed circular layouts sometimes reveal global properties of the graph such as symmetries and patterns of collective behavior. On the other hand, this strong regularity can obscure other information. For example, these drawings can be very dense, and following paths on them can be difficult.

In this work we suggest methods for improving the clarity of circular layouts through better node placement and edge routing. This is achieved using three contributions. The

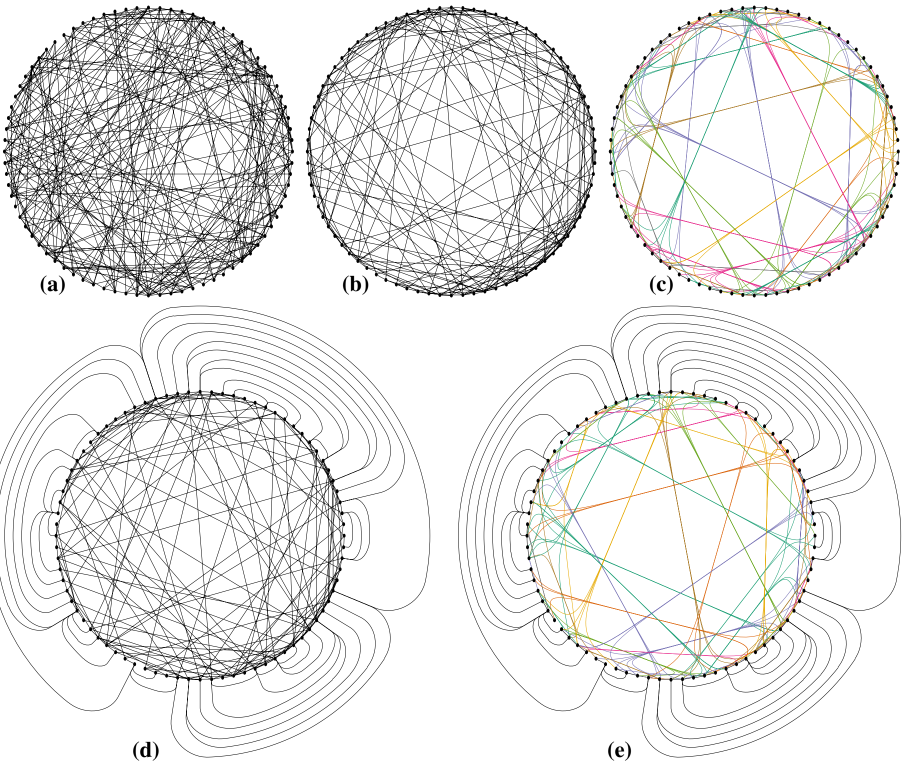


**Fig. 1.** Variations on circular layouts of a random graph ($|V| = 80, |E| = 241$): **(a)** random order; **(b)** edge-length minimizing order; **(c)** bundling edges to save ink and to improve area utilization (colors used to enhance readability); **(d)** exterior routing lessens crossings and alleviates density; **(e)** combining exterior routing with edge bundling

first shows how to adapt traditional energy based node placement considerations in order to shorten edges in circular layouts. This is different from most previous work which concentrated on reducing edge crossings. Experiments show that our method is competitive in terms of edge crossing minimization, while being constantly better in terms of overall edge length. Such a shortening of edge lengths allows the use of less "ink" for drawing the graph, thereby improving clarity. This ink saving paradigm brings us to the second contribution of the paper. We suggest a novel edge routing technique, which uses less ink compared with the common convention of drawing edges as straight lines. This is performed by carefully bundling together line segments between a few edges in a way that frees up drawing area without compromising structural clarity. Considering non-straight line edges opens up even more possibilities for better clarity. Accordingly, our third contribution suggests routing some of the edges through the external face. The externally routed edges are optimally selected in order to minimize certain criteria. In particular, external routing can be very effective in reducing edge crossings.

When used together, one normally performs node placement, followed by exterior routing, then edge bundling. Sections 2-4 consider these techniques in that order. Experimental studies for the techniques are given in Section 5.

## 2 Node Placement

We are looking for the "best placement" of the $n$ nodes of a graph $G(V=\{1,\ldots,n\},E)$. By convention, we assume nodes are equally spaced on the circle, which reduces the problem to finding a *circular ordering* of the nodes. This requirement imposes a certain regularity on the resulting layouts, while having no effect on the number of edge crossings, since only the ordering affects edge crossings.

Some related computational problems are known to be NP-hard. One example is the Minimum Circular Arrangement problem, where nodes are arranged on a circle (with equal gaps) in order to minimize the total angular edge lengths. This problem is reducible from the extensively studied NP-complete problem of Minimum Linear Arrangement [9]. Additional related circular arrangement problems and applications are mentioned in [6,14]. Another NP-hard problem is Circular Crossing Minimization [15], where the goal is to minimize the number of edge crossings in the layout. Given the NP-hardness of the relevant problems, our approach is based on heuristics that cannot guarantee finding an optimal solution.

### 2.1 Mean- and Median-Iterations

While previous work [1,4,12,17] explicitly addressed edge crossings, we prefer to deal with the simpler node-node interactions governing edge lengths. That way we can use ideas developed in other areas of graph drawing, which seek to minimize edge length. The rationale here is that long edges are hard to follow, prone to crossings, and cause unnecessary clutter and density. One such class of methods consists of force-directed algorithms, which define the layout by minimizing a cost function. The methods of Tutte [21] and Hall [10] are probably closest to the one used here. In addition, our technique is closely related to the mean-iteration and the median-iteration heuristics widely used within the crossing minimization phase of Sugiyama-based digraph drawing algorithms [19].

We denote the coordinates of a node $i \in V$ by $(x_i, y_i) \in \mathbb{R}^2$. Assume that the nodes are arranged on the unit circle centered at the origin. We would like to minimize the total squared edge lengths, resulting in the following optimization problem:

$$\begin{aligned} &\min_{x,y} \sum_{\langle i,j\rangle \in E} (x_i - x_j)^2 + (y_i - y_j)^2 \\ &\text{subject to}: \ \ x_i^2 + y_i^2 = 1, \ i = 1, \ldots, n \end{aligned} \tag{1}$$

Tutte [21] and Hall [10] dealt with strategies to minimize the same function, but here we also need to account for the unit circle constraints. Such equality constraints are usually addressed by *Lagrange multipliers*. Therefore, for each node $i$, we introduce a Lagrange multiplier $\lambda_i$, and define the function:

$$f(x, y, \lambda) = \sum_{\langle i,j \rangle \in E} (x_i - x_j)^2 + (y_i - y_j)^2 + \sum_{i=1}^{n} \lambda_i (x_i^2 + y_i^2 - 1) \qquad (2)$$

Any minimum of (1) must be a zero for all partial derivatives of (2). In other words, we require $\partial f / \partial x = 0$, $\partial f / \partial y = 0$, $\partial f / \partial \lambda = 0$. Notice that $\partial f / \partial \lambda = 0$ means that all constraints are satisfied. The other equalities, $\partial f / \partial x = \partial f / \partial y = 0$, imply that for each node $i$:

$$(x_i, y_i) = \frac{1}{1 - \lambda_i} \frac{\sum_{j \in N(i)} (x_j, y_j)}{\|N(i)\|}$$

where $N(i) = \{j \mid \langle i, j \rangle \in E\}$ is the set of neighbors of $i$. Notice that the $\frac{1}{1-\lambda_i}$ multiplier provides the degree of freedom necessary for satisfying the unit circle constraint. In plain words, these equations state that each node on the unit circle should lie on the line connecting the origin and the barycenter of its neighbors. Equivalently, the angular coordinate of each node is the mean of the angular coordinates of its neighbors, while the radial coordinate is always 1.

To solve this problem, we fix the positions of all the nodes but one, giving rise to an iterative optimization process, which we naturally name the *mean iteration*. At each iteration, we sequentially move each single node to the barycenter of its neighbors, and then project it back to the circle:

$$(\mathbf{1}) \;\; (x_i, y_i) \leftarrow \frac{\sum_{j \in N(i)} (x_j, y_j)}{\|N(i)\|} \qquad (\mathbf{2}) \;\; (x_i, y_i) \leftarrow \frac{(x_i, y_i)}{\|(x_i, y_i)\|}$$

A known problem with the mean iteration is that the global minimum of (1) is attained when all nodes are positioned at the same location. Since we are looking for more useful local minima, we avoid such a collapse of the layout by interfering with the process after each few tens of iterations and making the gaps between consecutive nodes uniform. That is, we preserve the current angular order of the nodes, but impose a uniform distribution along the circle. Additionally (or, alternatively), we adopt the anchoring mechanism suggested by Tutte, fixing the positions of three nodes, which prevents the collapse of the layout. During the process we change the anchors to avoid bias toward specific nodes.

While the mean iteration addresses squared edge lengths, a similar *median iteration* addresses non-squared edge lengths. The only difference is the use of the median instead of the mean. Therefore, in this algorithm, the coordinates of a node are iteratively determined by the component-wise median of its neighbors' coordinates, projected back onto the circle. We experienced slightly better results using median iteration over mean iteration in terms of crossing minimization.

The complexity of a single iteration is $O(n + |E|)$. The number of required iterations is less clear. We regularly use $O(n)$ iterations.

## 2.2 Local Refinement Through Dynamic Programming

The median (or mean) iteration is a continuous approximation to the circular ordering problem. We derive the circular order by sorting the nodes according to their angular coordinates. The resulting circular order can be refined by utilizing an algorithm that explicitly considers the discrete nature of the problem. At this stage, we hope that the median iteration already gave us an adequate global positioning of the nodes. Therefore, we opt for using a localized refinement procedure. This refinement procedure considers every sequence of $k$ nodes, and reorders the sequence in a way that minimizes the total edge length.

More formally, assume that the circle contains $n$ equally spaced points named $0, 1, \ldots, n-1$, where point $i$ is located at $(\cos\frac{2\pi i}{n}, \sin\frac{2\pi i}{n})$. In addition, each of the $n$ nodes is uniquely associated with one of the $n$ circle points via the bijection $\mathrm{p}(i) : V \rightarrow \{0, 1, \ldots, n-1\}$. The (angular) distance between two nodes $i$ and $j$ is defined as:

$$d_{ij} = \min\left(\mathrm{p}(i) - \mathrm{p}(j) \mod n,\, \mathrm{p}(j) - \mathrm{p}(i) \mod n\right)$$

Given $k$ nodes $\mathcal{V} = \{v_1, v_2, \ldots, v_k\}$, located consecutively at $\mathrm{p}(v_1), \mathrm{p}(v_1)+1, \ldots, \mathrm{p}(v_1)+k-1$, we would like to reorder $\mathcal{V}$ to minimize $l(\mathcal{V})$, the total length of the edges adjacent to $\mathcal{V}$, which is defined as:

$$l(\mathcal{V}) = \sum_{\langle i,j\rangle \in E, i \in \mathcal{V}} d_{ij}$$

Minimization of $l(\mathcal{V})$ is done by a dynamic programming algorithm which rearranges increasingly larger subsets of $\mathcal{V}$. The pseudocode is given in Figure 2. The complexity of the algorithm is $O(2^k + |E(\mathcal{V})|)$, where $E(\mathcal{V})$ is the set of edges connected to $\mathcal{V}$. Typical values of $k$ are between 5 and 10 (our default is 6). We iteratively run it on each of the $n$ (overlapping) subsequences of length $k$, so the running time of a full sweep optimizing each subsequence is $O(n(2^k + |E|))$. We run a few sweeps until the total edge length cannot be further reduced. Typically, a very low number of sweeps (10 or less) is required for convergence.

Figure 1(b) illustrates the application of these techniques to the initial layout of Figure 1(a).

## 3 Exterior Routing

Node ordering, using the method described in Section 2 (or one of the methods described in the literature [1,4,12,17]), improves the readability of the layout by removing edge crossings and shortening edges. At this stage, further readability improvement can be achieved without altering the node positions. This is accomplished by taking a subset of the edges from the interior of the circle, and routing them around the exterior of the circle, as depicted in Figure 1(d). Importantly, this can be done in an optimal way which maximizes the number of extracted edges or minimizes the number of crossings.

Since exterior routing of an edge is inherently longer than interior routing, we should utilize the exterior routing carefully, and make sure that edges routed externally are readable. Therefore, we do not allow any edge crossing within the external face. Notice that two edges cross in the external face if and only if they cross internally.

We associate weights with the edges (as explained below), and strive to maximize the total weight of the extracted edges. This is carried out using a dynamic programming algorithm. Before describing the algorithm, we make an observation about "edge

```
Function MinCA_DP ($G(V,E), p, \mathcal{V} = \{v_1, v_2, \ldots, v_k\} \subset V, ordering$)
% Given a graph ($G$), circular node positioning ($p$), and a subset of consecutive nodes ($\mathcal{V}$)
% ordered from $v_1$ (leftmost) to $v_k$ (rightmost)
% compute an ordering of $\mathcal{V}$ ($ordering$) that minimizes total edge length
% Data structure: A table $T$ whose entries are indexed by subsets of $\mathcal{V}$
% The function $\mathrm{Cut}(i, S)$ returns the number of edges between $i$ and $S \subset \mathcal{V}$.

  for each $i \in \mathcal{V}$ compute
    $\mathrm{left}(i) \ = \{\langle i,j \rangle \in E \mid d(j, v_1) < d(j, v_k), j \notin \mathcal{V}\}$
    $\mathrm{right}(i) = \{\langle i,j \rangle \in E \mid d(j, v_k) < d(j, v_1), j \notin \mathcal{V}\}$
  end for

  % Initialize table:
  for every $S \subseteq \mathcal{V}$ do
    $table[S].cost \leftarrow \infty$
  end for
  $table[\emptyset].cost \leftarrow 0$
  $table[\emptyset].cut \leftarrow= \sum_{i \in \mathcal{V}} |\mathrm{left}(i)|$

  % Fill table:
  for $i = 1$ to $k$ do
    for every $S \subset \mathcal{V}$, $|S| = i - 1$ do
      $cut^S \leftarrow table[S].cut$
      $new\_cost \leftarrow table[S].cost + cut^S$   % total edge length is a sum of cuts
      for every $j \in \mathcal{V} - S$ do
        if $table[S \cup \{j\}].cost > new\_cost$ then
          $table[S \cup \{j\}].cost \leftarrow new\_cost$
          $table[S \cup \{j\}].right\_vtx \leftarrow j$
          $table[S \cup \{j\}].cut \leftarrow cut^S - |\mathrm{left}(j)| + |\mathrm{right}(j)| - \mathrm{Cut}(j, S) + \mathrm{Cut}(j, \mathcal{V} - S)$
        end if
      end for
    end for
  end for

  % Retrieve optimal ordering:
  $S \leftarrow \mathcal{V}$
  for $i = k$ to $1$ do
    $v \leftarrow table[S].right\_vtx$
    $ordering[i] \leftarrow v$
    $S \leftarrow S - \{v\}$
  end for
end
```

**Fig. 2.** A dynamic programming algorithm for reordering a sequence of nodes in order to minimize total edge length

flipping". Each exterior edge $\langle i, j \rangle$ can be drawn in two ways: either along the short arc connecting $i$ and $j$, or along the complementary long arc connecting $i$ and $j$. Therefore, we assume that all exterior edges are flipped so that no edge is passing over the length-1 arc connecting point $n - 1$ with point 0 on the circle. Note that this flipping will not introduce any crossing into a crossing-free layout. As a consequence, we can cut the circle between point $n-1$ and 0, where no edge passes, and solve an equivalent problem on a line starting at 0 and ending at $n-1$. By solving the problem on a line, we determine which edges should be extracted. Then, each of these edges will be drawn on the exterior of the circle along the shorter of the two possible arcs.

The intuition behind the algorithm for solving the problem on the line is based on likening each edge to parentheses, where the left endpoint of the edge opens the parenthesis, and the right endpoint closes it. Accordingly, a non-crossing set of edges is equivalent to a valid sequence of nested parentheses. This induces the following recurrence relation, where $p_{ij}$ is the maximal weighted sum of edges that can be legally routed between $i$ and $j$:

$$\begin{array}{lll} p_{i,i+1} & = w_{i,i+1} & i = 0, \ldots, n-2 \\ p_{i,j} & = w_{i,j} + \max_{i<k<j}\{p_{ik} + p_{kj}\} & i = 0, \ldots, n-3,\ i+1 < j < n \end{array} \qquad (3)$$

Here, $w_{ij}$ is the weight of $\langle \mathrm{p}^{-1}(i), \mathrm{p}^{-1}(j)\rangle \in E$. Also, $w_{ij} = 0$ if $\langle \mathrm{p}^{-1}(i), \mathrm{p}^{-1}(j)\rangle \notin E$. The target value, $p_{0,n-1}$, is computed in time $O(n^2)$ by dynamic programming. This value indicates the maximal weighted sum of edges that can be extracted. The edges themselves are easily recovered using an auxiliary data structure which enables tracking the computation of $p_{0,n-1}$.

The choice of edge weights ($w_{ij}$) allows flexibility in the optimization goal. Our default is to pick the weights in a way that ensures minimizing the number of edge crossings. To this end, we set $w_{ij}$ to the number of crossings involving $\langle \mathrm{p}^{-1}(i), \mathrm{p}^{-1}(j)\rangle$. In this way, the maximized value $p_{0,n-1}$ is exactly the number of saved edge crossings. Note that there is no problem of double counting, since two extracted edges cannot cross each other.

Our experience shows that exterior routing is a very effective technique, which can remove a significant portion of the edge crossings. The effect is shown in Figure 1(d) and studied in Section 5.

An additional pleasing outcome of exterior routing is that it tends to extract many of the short edges, such as edges of length 2. These edges are often hard to read when drawn as straight lines, as they are almost collinear with the adjacent length-1 edges. Furthermore, collinearity issue of specific edges can be explicitly addressed by increasing their weights, thus encouraging the algorithm to pick them for exterior routing.

## 4 Edge Bundling

After node places are computed and possibly some edges are extracted to be drawn outside the circle, we can further improve the clarity of the drawing by using *edge bundling*. The essence of this technique is a controlled deformation of the edges, such that groups of edges share long common segments, thereby improving the utilization of the drawing area by saving ink. Put differently, while the most economical way to draw a single edge is by using a straight line, when displaying of group of edges, there might be more efficient ways. For illustration, consider Figure 1(c,e).

The idea of bundling edges is related to the work on confluent drawing [3], where edge crossings are eliminated by grouping edges in tracks. Newbery [16] applied bundling to Sugiyama-style layouts to reduce clutter. Additionally, we were inspired by a recent work by Holten and van Wijk [11] that suggested bundling edges based on hierarchical structure associated with the nodes. Our approach is based on a different technique for bundling edges. In the following, we split the description of the technique into two parts. First, we describe how to bundle together a given set of edges in a way that maximizes area utilization and readability. Second, we describe the algorithm for computing the sets of edges that will be bundled.

Consider the case where we are given a set of $m$ lines (“edges”), $Q = \{e_1 = (v_1, u_1), e_2 = (v_2, u_2), \ldots, e_k = (v_m, u_m)\}$, where $v_i, u_i \in \mathbb{R}^2$. In the accompanying example, given in Figure 3, this set includes the 4 edges $(A, E)$, $(B, F)$, $(C, G)$ and $(D, H)$. Our first step is to divide the $2m$ endpoints of the edges into two equally sized sets – $S$ (“sources”) and $T$ (“targets”) – such that for each $(v_i, u_i) \in Q$, either $v_i \in S$, $u_i \in T$, or $u_i \in S$, $v_i \in T$. The intention here is to produce two compact sets, minimizing Euclidean distances between nodes belonging to the same set. We achieve this by a variant of the $K$-means algorithm, where we iteratively assign each point to the set with the closer mean while continually updating the means. Accordingly, in the given example we would choose $S = \{A, B, C, D\}$, $T = \{E, F, G, H\}$.

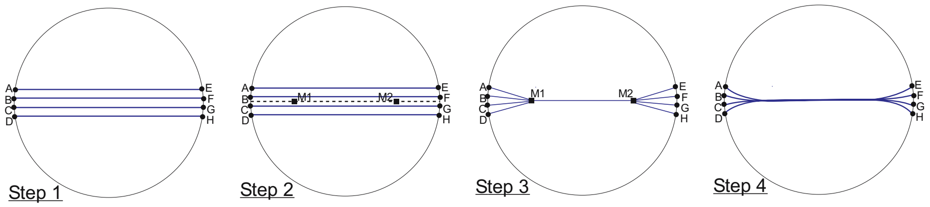


**Fig. 3.** Non-crossing edges are bundled together thereby freeing up drawing area

The next step is to compute the centroids of $S$ and $T$, denoted as $\bar{S}$ and $\bar{T}$, respectively. We denote by $L$ the line containing $\bar{S}$ and $\bar{T}$. The prospective bundling should pass along this line. More specifically, we compute two points – $M_1$ and $M_2$ – on $L$ such that the bundling is carried out by replacing the original line segments by the following line segments: First, a line from each node of $S$ to $M_1$, the meeting point of the “sources”. Then, a line from $M_1$ to $M_2$. Finally, a line from each node of $T$ to $M_2$, the meeting point of the “targets”. See Step 3 in Figure 3. Since we want to reduce the use of ink, the exact positions of $M_1$ and $M_2$ minimize the total line length:

$$(M_1, M_2) = \operatorname*{argmin}_{M_1, M_2} \sum_{p \in S} \|M_1 - p\| + \|M_1 - M_2\| + \sum_{p \in T} \|M_2 - p\|$$

We solve this using a numerical method.

At this stage, we can infer if bundling the lines of $Q$ is profitable, as the ink potentially saved is exactly the difference:

$$\sum_{(v_j, u_j) \in Q} \|v_j - u_j\| - \left( \sum_{p \in S} \|M_1 - p\| + \|M_1 - M_2\| + \sum_{p \in T} \|M_2 - p\| \right)$$

If this difference is positive, we know that we gain area by bundling.

If bundling is $Q$ worthwhile, we recommend depicting each line $(v_i, u_i)$ using a Bézier spline with $M_1$ and $M_2$ as control points. See Step 4 in Figure 3, and Figure 1(c,e). Our experience is that by incorporating Bézier splines, the drawing is smoother and more readable. Also, the readability of edge bundles is improved when each of them is uniquely colored, as can be seen in Figure 1(c,e).

A possible problem when bundling edges is that we might lose the information about which "source" is connected to which "target". For example, in the final picture of Figure 3, it is unclear whether $A$ is connected to $E$ or maybe to $F$, $G$, or $H$. We adopt a simple rule to address this problem: *crossing edges can never be bundled together*. The edges exit the bundle at the same order they entered it, thus avoiding any ambiguity when each source is connected to a unique target.

Now, we turn to the problem of identifying the sets of edges to be bundled. We pick the sets of edges such that, by bundling them, we minimize the amount of ink used. Our choice is to use a bottom-up, agglomerative approach. The process starts with multiple sets, each of which contains a single edge. Then, sets are merged as long as the corresponding bundling improves drawing area utilization; see pseudocode is given in Figure 4.

```
Function BundlingGain ($Q_1, Q_2 \subset E$)
% Return the ink gain by bundling two edge sets (negative value means no gain)
% The function $\mathrm{Ink}(S)$ returns total ink needed for most efficient drawing of $S \subset E$
  if EdgeCrossing($Q_1$,$Q_2$) then
    return -1
  else
    return $\mathrm{Ink}(Q_1) + \mathrm{Ink}(Q_2) - \mathrm{Ink}(Q_1 \cup Q_2)$
  end if
end

Function AgglomerativeBundling ($E = \{e_1, e_2, \ldots, e_m\}$)
% Iteratively, grow edge bundles that improve drawing area utilization
  $sets \leftarrow \{\{e_1\}, \{e_2\}, \ldots, \{e_m\}\}$
  while profitable bundling is possible do
    % Pick two sets generating most gain:
    $(Q_1, Q_2) \leftarrow \mathrm{argmax}_{Q_1, Q_2 \in sets}\, \mathrm{BundlingGain}(Q_1, Q_2)$
    $sets \leftarrow sets \cup \{Q_1 \cup Q_2\} - \{Q_1, Q_2\}$
  end
  return $sets$
end
```

**Fig. 4.** Agglomerative edge bundling algorithm

Concerning computational complexity, this algorithm is essentially a hierarchical clustering algorithm performed on the edges, and therefore it has $O(|E|^2)$ time and space complexity (counting "bundlingGain" calculations), according to Eppstein [5]. The practical situation, however, is better here. First, only a tiny fraction of edge pairs are mergeable since, for most pairs, there is no gain from bundling or the edges cross. Therefore, $O(|E|^2)$ space is unnecessary in practice, and we use a sparse data structure holding only profitable edge pairs. Moreover, when bundling two sets $Q_1$ and $Q_2$, we would consider for potential bundling with $Q_1 \cup Q_2$ only sets that could be bundled with $Q_1$ or $Q_2$. Finally, the $O(|E|^2)$ time complexity needed for evaluating the bundling gain of all possible edge pairs can be significantly alleviated if we initially consider only bundles involving two nearby edges; other bundles can be considered later by transitivity. Here, two edges $e_1, e_2$ are considered "nearby" if one of $e_1$'s endpoints is sufficiently close to one of $e_2$'s endpoints in the given circular ordering.

## 5 Experiments

We evaluated the performance of our methods on the known benchmark set of Rome graphs [2], which contains 11,534 real-life, sparse graphs with 10–109 nodes. In addition, we tested our algorithms on a set of pseudo-random graphs characterized by their average degrees; all these graphs contain 100 nodes.

As a reference, we picked the CIRCULAR algorithm by Six and Tollis [17]. This algorithm finds a circular ordering in two steps. The first step creates an initial ordering based on the largest outerplanar subgraph. Then, the second step iteratively reduces the number of crossings by carefully moving nodes. We used the publicly available implementation *circo*, which is part of the Graphviz package [8].

Another circular ordering algorithm is that of Baur and Brandes [1]. They also explicitly address edge crossings using a two phase process. The reported numbers of crossings are better – by up to 20% – compared with the aforementioned CIRCULAR

**Table 1.** Comparing number of crossings across different circular ordering options, with and without exterior edge routing

| | | No exterior routing | | | Exterior routing | | |
|---|---|---|---|---|---|---|---|
| Name | #graphs | C | M | MC | C | M | MC |
| Rome, 10–19 nodes | 1407 | 2.61 | 3.19 | 2.11 | 0.16 | 0.15 | 0.09 |
| Rome, 20–29 nodes | 839 | 7.18 | 8.01 | 5.51 | 0.83 | 0.68 | 0.46 |
| Rome, 30–39 nodes | 2037 | 21.42 | 22.17 | 16.42 | 4.29 | 3.33 | 2.48 |
| Rome, 40–49 nodes | 1802 | 41.49 | 41.06 | 31.68 | 11 | 8.77 | 6.66 |
| Rome, 50–59 nodes | 1045 | 66.46 | 65.16 | 51.16 | 20.66 | 16.67 | 12.8 |
| Rome, 60–69 nodes | 1172 | 92.76 | 91.3 | 72.51 | 32.4 | 26.93 | 21.33 |
| Rome, 70–79 nodes | 1008 | 123.47 | 120.94 | 96.23 | 47.43 | 39.46 | 31.04 |
| Rome, 80–89 nodes | 788 | 167.29 | 161.84 | 130.41 | 69.84 | 58.53 | 46.73 |
| Rome, 90–99 nodes | 1296 | 209.12 | 205.64 | 165.4 | 92.64 | 80.24 | 64.28 |
| Rome, 100–109 nodes | 140 | 230.1 | 229.52 | 183.83 | 103.45 | 92.74 | 72.97 |
| Random, avg. deg. 3 | 100 | 383.23 | 357.68 | 302.29 | 195.22 | 166.18 | 139.12 |
| Random, avg. deg. 4 | 100 | 1337.68 | 1186.50 | 1048.19 | 838.42 | 714.08 | 627.06 |
| Random, avg. deg. 5 | 100 | 2709.35 | 2489.69 | 2230.24 | 1858.20 | 1678.77 | 1487.14 |
| Random, avg. deg. 6 | 100 | 4437.51 | 4252.31 | 3843.23 | 3192.80 | 3043.01 | 2719.80 |
| Random, avg. deg. 7 | 100 | 6979.42 | 6843.71 | 6210.86 | 5216.34 | 5126.31 | 4594.56 |
| Random, avg. deg. 8 | 100 | 9931.27 | 9808.96 | 8992.90 | 7646.76 | 7545.26 | 6865.73 |

**Table 2.** Comparing total used ink (total length of edges) across different circular ordering options, with and without edge bundling

| | | No edge bundling | | | Edge bundling | | |
|---|---|---|---|---|---|---|---|
| Name | #graphs | C | M | MC | C | M | MC |
| Rome, 10–19 nodes | 1407 | 12.94 | 12.34 | 12.33 | 10.33 | 10.11 | 10.11 |
| Rome, 20–29 nodes | 839 | 17.16 | 15.49 | 15.50 | 13.23 | 12.52 | 12.54 |
| Rome, 30–39 nodes | 2037 | 23.12 | 20.38 | 20.34 | 17.46 | 16.26 | 16.25 |
| Rome, 40–49 nodes | 1802 | 29.08 | 25.26 | 25.19 | 22.26 | 19.73 | 19.68 |
| Rome, 50–59 nodes | 1045 | 34.51 | 29.59 | 29.34 | 24.59 | 22.71 | 22.57 |
| Rome, 60–69 nodes | 1172 | 39.04 | 33.53 | 33.19 | 27.39 | 25.38 | 25.10 |
| Rome, 70–79 nodes | 1008 | 43.57 | 37.28 | 36.58 | 30.30 | 27.98 | 27.56 |
| Rome, 80–89 nodes | 788 | 49.39 | 42.16 | 41.33 | 33.62 | 31.02 | 30.58 |
| Rome, 90–99 nodes | 1296 | 53.99 | 46.21 | 45.14 | 36.31 | 33.63 | 33.03 |
| Rome, 100–109 nodes | 140 | 56.19 | 48.68 | 47.11 | 37.51 | 35.08 | 34.31 |
| Random, avg. deg. 3 | 100 | 72.44 | 62.72 | 61.99 | 46.07 | 43.56 | 42.59 |
| Random, avg. deg. 4 | 100 | 124.08 | 109.29 | 107.96 | 72.04 | 68.20 | 66.77 |
| Random, avg. deg. 5 | 100 | 171.85 | 156.19 | 153.97 | 93.73 | 90.08 | 88.52 |
| Random, avg. deg. 6 | 100 | 220.81 | 206.14 | 202.87 | 114.39 | 111.47 | 109.45 |
| Random, avg. deg. 7 | 100 | 273.35 | 260.37 | 254.79 | 207.10 | 198.32 | 194.59 |
| Random, avg. deg. 8 | 100 | 325.10 | 312.25 | 306.18 | 243.18 | 234.58 | 230.81 |

algorithm. We did not have an implementation of this algorithm, so no direct comparison was performed.

The quality of the drawings was assessed using two aesthetic criteria: number of crossings and total used ink.[1] The results are given in Tables 1 and 2.

The evaluated algorithms are coded in the tables as follows: **C**=CIRCULAR; **M** = Median iteration followed by fine-tuning, as described in Section 2; **MC** = Median iteration followed by fine-tuning and then by the second step of CIRCULAR.

We begin with observations about the circular orderings. In terms of crossings minimization, there is no marked difference between our method (M) and CIRCULAR (C) for the Rome graphs, while M could produce fewer edge crossings than C for the random graphs. As for the edge lengths (Table 2), M consistently achieves better results, which is not surprising as CIRCULAR does not address edge lengths but crossings. Since M does not directly deal with edge crossings, we tried to make it more "crossings aware", by integrating it with the second step of CIRCULAR, obtaining the method coded by MC. As the table shows, MC is consistently the best performer in terms of crossing minimization, outperforming both C and M.

So far, we have compared plain circular orderings. Interestingly, all differences, in terms of number of crossings, are dwarfed by the effect of exterior routing (Section 3). As can be seen in the right columns of Table 1, exterior routing is capable of eliminating a significant portion of the edge crossings. Also, when exterior routing is activated, our method (M) produces fewer crossings than CIRCULAR (C) even for the Rome graphs, whereas the combined method – MC – is still superior. Apparently, our method can

[1] We prefer the term "total used ink" over the more common "total edge length", since when edge bundling is activated they are no longer equivalent.

better benefit from exterior routing because, by producing shorter edges, it allows more non-crossing edges to be routed externally. In fact, for the Rome graphs, M allows an external routing of 23% of the edges (on average, surprisingly uniform for all graph sizes), while C allows external routing of 18% of the edges and MC routes 19% of the edges externally.

We see that M has an advantage in reducing the total used ink. When allowing edge bundling (Section 4), a further significant improvement in drawing area utilization is achieved, as shown in the right columns of Table 2. Our experience shows that this ink saving is helpful in conveying a clearer layout. Notice that a further reduction of drawing density could be obtained by exterior routing, but it was not considered in Table 2, as our intention is to isolate the effect of circular ordering and bundling on ink usage.

Finally, as to running time, the average measured running time on the 100-node graphs is around 1 second on a Pentium 4 machine. This is comparable with the running time of the CIRCULAR algorithm. Almost all running time is dedicated to the computation of the circular ordering. The time needed for computing the edge bundling is 50–200ms (depending on the number of edges), whereas the time for computing the external edges is insignificant.

## 6 Summary

Circular layouts are a rather restrictive layout scheme, offering a simple and highly regularized picture of the graph where nodes cannot be occluded. The limiting nature of circular layouts makes it very important to capitalize on all available degrees of freedom. In this work, we explored new ways for positioning nodes and routing edges in order to maximize the readability of the layouts. In particular, the density of the drawing is alleviated by shortening edge lengths, moving part of the edges to the exterior of the circle, and bundling some edges together. In addition, shortening edges and exterior routing significantly reduce the number of edge crossings.

## References

1. M. Baur and U. Brandes, “Crossing Reduction in Circular Layouts”, *Proc. Graph-Theoretic Concepts in Computer-Science (WG ’04)*, 332-343, 2004.
2. G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari and F. Vargiu, “An Experimental Comparison of Four Graph Drawing Algorithms”, *Comput. Geom. Theory Appl.* **7**, 303–325, 1997.
3. M. Dickerson, D. Eppstein, M. T. Goodrich and J. Meng, “Confluent drawings: Visualizing Non-Planar Diagrams in a Planar Way”, *Proc. Graph Drawing (GD’03)*, 1–12, 2003.
4. U. Doğrusöz, B. Madden and P. Madden, “Circular layout in the Graph Layout Toolkit”, *Proc. Graph Drawing (GD ’96)*, 92–100, 2003, 1996
5. D. Eppstein, “Fast Hierarchical Clustering and Other Applications of Dynamic Closest Pairs”, *Journal of Experimental Algorithmics* **5**, 1–23, 2000.
6. M. K. Ganapathy and S. Lodha, “On Minimum Circular Arrangement”, *Proc. 21st Annual Symposium on Theoretical Computer Science (STACS’04)*, 394–405, 2004.

7. E. R. Gansner, Y. Koren and S. North, "Graph Drawing by Stress Majorization", *Proc. Graph Drawing (GD'04)*, 239–250, 2004.
8. E. R. Gansner and S. North, "An Open Graph Visualization system and its Applications to Software Engineering", *Software - Practice & Experience* **30**, 1203–1233, 2000. Also, www.graphviz.org.
9. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP Completeness*, Freeman, 1979.
10. K. M. Hall, "An $r$-dimensional Quadratic Placement Algorithm", *Management Science* **17**, 219–229, 1970.
11. D. Holten and J. J. van Wijk, "Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data", *Proc. IEEE Information Visualization (InfoVis'06)*, to appear.
12. G. Kar, B. Madden and R. S. Gilbert, "Heuristic Layout Algorithms for Network Management Presentation Services", *IEEE Network* 29–36, 1988.
13. M. Kaufmann and R. Wiese, "Maintaining the Mental Map for Circular Drawings", *Proc. Graph Drawing (GD'02)*, 12–22, 2002.
14. V. Liberatore, "Multicast Scheduling for List Requests", *Proc. IEEE INFOCOM 2002*, 1129–1137, 2002.
15. S. Masuda, T. Kashiwabara, K. Nakajima and T. Fujisawa, "On the NP-Completeness of a Computer Network Layout Problem", *Proc. IEEE International Symposium on Circuits and Systems*, 292–295, 1987.
16. F. J. Newbery, "Edge Concentration: A Method for Clustering Directed Graphs", *Proc. 2nd Intl. Workshop Software Configuration Management*, 76–85, 1989.
17. J. M. Six and I. G. Tollis, "Circular Drawings of Biconnected Graphs", *Proc. Algorithms Engineering and Experimentation (ALENEX'99)*, 57–73, 1999.
18. J. M. Six and I. G. Tollis, "A Framework for Circular Drawings of Networks", *Proc. Graph Drawing (GD'99)*, 107–116, 1999.
19. K. Sugiyama, S. Tagawa and M. Toda, "Methods for Visual Understanding of Hierarchical Systems", *IEEE Trans. Systems, Man, and Cybernetics* **11**, 109–125, 1981.
20. A. Symeonidis and I. G. Tollis, "Visualization of Biological Information with Circular Drawings", *Proc. Biological and Medical Data Analysis (ISBMDA'04)*, 468–478, 2004.
21. W. T. Tutte, "How to Draw a Graph", *Proc. London Mathematical Society* **13**, 743–768, 1963.

# Controllable and Progressive Edge Clustering for Large Networks

Huamin Qu, Hong Zhou, and Yingcai Wu

Department of Computer Science and Engineering,
The Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong
{huamin,zhouhong,wuyc}@cse.ust.hk

**Abstract.** Node-link diagrams are widely used in information visualization to show relationships among data. However, when the size of data becomes very large, node-link diagrams will become cluttered and visually confusing for users. In this paper, we propose a novel controllable edge clustering method based on Delaunay triangulation to reduce visual clutter for node-link diagrams. Our method uses curves instead of straight lines to represent links and these curves can be grouped together according to their relative positions and directions. We further introduce progressive edge clustering to achieve continuous level-of-details for large networks.

## 1 Introduction

Many visualization problems can be modeled using node-link diagrams or networks, where nodes represent data elements and links their relationships. For example, hyperlinks among Internet webpages, citations in scientific papers, traffics between telecommunication switches, and airline routes can all be represented by node-link diagrams. As the amount of data from real world keeps increasing, visual clutter becomes a very serious problem for large networks and greatly affects the effectiveness of networks for conveying information.

Visual clutter is usually caused by an excessive number of nodes and links. Visual clutter for edges caused by too many edge crossings is also called edge congestion [1] [2]. Too many crossings of links will obscure some nodes and links in the graph. Various filtering and clustering methods [3] can be used to effectively reduce the number of nodes and thus the number of links. However, simply reducing the number of nodes is not a practical solution for some applications. For example, in typical airline routes, removing a node will cause the lost of route information for an airport. Clustering of nodes may not be a good solution either. Users may have problems to relate the links coming in or out of virtual clustered nodes to real links.

Edge congestion is a challenging problem and many approaches have been proposed. Edge crossings can be reduced by rearranging the nodes and edges. Various force-based or energy-based node layout algorithms for graphs [4] can generate good layouts for small-size graphs according to some aesthetic criteria including minimum edge crossings. However, for large graphs, edge crossings usually cannot be reduced to a satisfying level. Some researchers try to totally avoid edge crossings by making the graph planar [5]. However, large graphs usually cannot be drawn in a planar way. For some applications such as communication and transportation networks, the semantic meanings of the node positions limit the room of adjustment for nodes.

Edge congestion can be alleviated by merging edges and drawing edges as curves. Phan et al. [6] proposed a *flow map layout* to reduce visual clutter by merging edges. The flow map layout is generated by hierarchical clustering. The node positions are distorted but relative positions are maintained in their algorithm. Their method is mainly designed for single-source node-link diagrams although the authors mentioned that their methods can be extended for multiple-source networks by overlapping multiple flow maps. Carpendale and Rong described an interesting technique to examine edge congestion around node areas by adjusting the edge position if the edge passing through the node [1]. They used an edge-displacement algorithm to curve away all edges from the region of interest. Wong et al. [2] introduced *EdgeLens* to manage edge congestion in graphs. They pointed out that the relation between vertices might be displayed ambiguously if their edges overlap. Their solution is to introduce a lens which displaces edges in a local area with dense edge overlapping and reveals hidden information in that area and clarifies graph structures. While the *EdgeLens* is quite effective to make the edges in a small area more discernable, the visual clutter of the overall graphs is not reduced and the large-scale patterns cannot be effectively revealed by this method. Wong and Carpendale further introduced *Edge Plucking* [7], an interesting interactive technique which allows users to temporarily pluck edges apart to clarify node-edge relationships.

In this paper, we describe a controllable and progressive edge clustering method to address edge congestion for large network visualization. Our research follows the same direction of some previous work [1, 6]. Our method first connects the nodes by Delaunay triangulation and then sets some control points on the Delaunay edges of these triangles. After that, we convert all links into a series of paths consisting of these control points. By adjusting the number and positions of these control points, different levels of edge clustering can be achieved. By setting a minimum distance between these control points and the original nodes, the ambiguity cases in the traditional node-link diagrams can be avoided. By grouping links together, the high level linkage patterns related to the whole node-link diagram can be revealed. Compared with node-clustering methods, our method can reveal the linkage information for real nodes instead of clustered virtual nodes. Compared with force-based or energy-based approaches, our method is geometry based and all computations (e.g., Delaunay triangulation, ray/triangle intersection, and K-Means clustering) can be done very efficiently and can be accelerated by graphics hardware. Our method is easy to implement and the generated layouts are visually appealing. In addition, our approach gives users great flexibility for layout generation. Users can easily and dynamically change the size of "protected" areas around nodes and the levels of clustering for links. We further introduce progressive edge clustering to achieve continuous level-of-details for large networks.

## 2 Controllable Edge Clustering

In this section, we introduce our method for edge clustering given a set of nodes and links. We assume that the positions of nodes have been computed by some other methods such as force-based models [4] and a relatively good initial layout has been obtained. We do not further distort node positions. Therefore, their original layout is preserved. Actually, for certain applications, node positions encode geographic information and any adjustment of node positions may cause confusion for users.

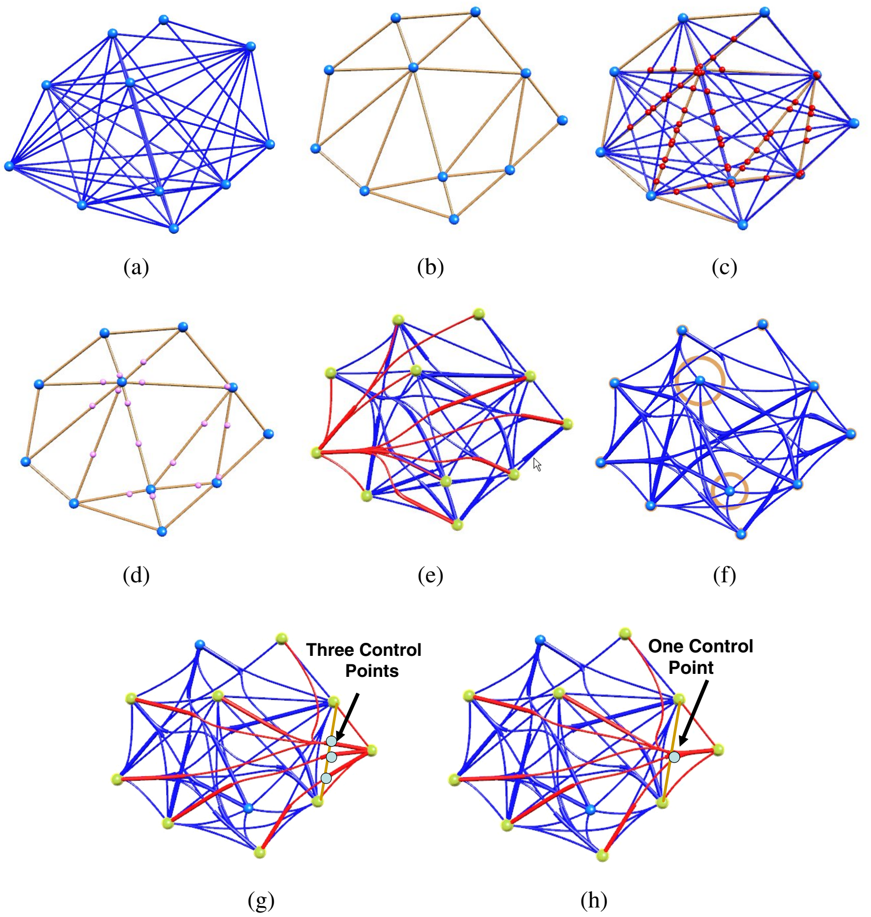


**Fig. 1.** Framework of our method: (a) Original node-link diagram; (b) Delaunay triangulation of the original nodes; (c) The intersection points of all links and Delaunay edges indicated by the red dots; (d) The control points computed by clustering the intersection points; (e) All links are forced to pass through the control points; (f) Protected areas for nodes; (g) Three control points on a Delaunay edge; (h) One control point on a Delaunay edge

Figure 1 shows the framework of our method. Figure 1 (a) shows the original node-link diagram. We first compute a Delaunay triangulation of the points given by the positions of the vertices (See Fig. 1 (b)). Then, for each link, we compute the intersection points of this link with the Delaunay edges (See Fig. 1 (c)). For each Delaunay edge, we cache all the links passing through this Delaunay edge and their intersection points with this edge. Then we assign one or more control points on each Delaunay edge and use these control points to cluster links (See Fig. 1 (d)). The control points can be clustered using the K-Means algorithm. Later, the number and positions of these control points can be adjusted by users to achieve different levels of clustering. After that, we force all the links to pass through these control points and compute a path consisting of these control points for each link (See Fig. 1 (e)). Some overlapping paths will be then grouped together by the system or by users. In this paper, all the paths are drawn as Nurbs curves.

By forcing all links to pass through control points on the Delaunay edges, we can easily solve the ambiguity cases for node-link diagrams. We can set up a **"protected area"** for each node by forcing the control points to be at lease certain distance away from the node. If only one control point is used on each Delaunay edge, the protected area for each node can be as large as half of the minimum distance from this node to its neighboring nodes. If two or more control points are used, then we can easily control the size of the protected areas for nodes. In practice, we can compute this size based on the importance of the nodes if any criterion for importance is given by users. The size of protected areas can also be adjusted by users during the visualization process. Figure 1f shows the protected areas for two nodes.

Because all links have to pass through the control points on Delaunay edges, then some link segments will be automatically clustered and some flow-map-style effect can be achieved. In practice, we can start from the coarse level and gradually refine edge clustering. First, we assign only one control point on each Delaunay edge and then we compute the shortest paths consisting of these control points for all links. Then we examine all incoming and outgoing links for each node. For all paths passing through the same set of control points, we group that parts of links together. After finding all the overlapping path segments for all links, we put more control points on the Delaunay edges so the directions of links will be closer to their original directions.

The level of clustering can be controlled by users. We provide an interface which allows users to dynamically change these parameters so the node-link diagrams can be examined at different level-of-details. There are two major parameters users can adjust: the number of control points on Delaunay edges and their positions. We use the following principles to position control points: For short or unimportant Delaunay edges (i.e., edges with only a few or even zero link passing through), there are fewer control points; The positions of control points will be close to the real intersection points of the links and the Delaunay edges. Figure 1 (g) and (h) show different levels of edge clustering by adjusting the number and positions of control points on a Delaunay edge. From the figure we can see that our layout is visually appealing and our method gives users great flexibility to control the final layout.

## 3 Progressive Edge Clustering

If there are too many nodes and links, even clustering all links on all Delaunay edges cannot solve the visual clutter problem. Inspired by progressive mesh simplification [8], we propose progress edge clustering by collapsing Delaunay edges. As illustrated in

Figure 2, if we collapse one Delaunay edge (or two nodes), then the total number of Delaunay edges will be reduced and the links will become smoother and more space will be available for positioning nodes and links. Progressive edge clustering is based on collapsing Delaunay edges one by one so that at each step only one Delaunay edge disappears. There are two major issues for progressive edge clustering: the order of collapsing for Delaunay edges and the new position of the node(s) after collapsing one Delaunay edge. We use the following criteria to determine the order in which vertices collapse: The length of the Delaunay edges; The number of links passing through the Delaunay edges; After the collapsing, the maximum distance of the newly positioned paths to the original links and the curvatures of these paths. The short Delaunay edges with less links passing through will be clustered first.

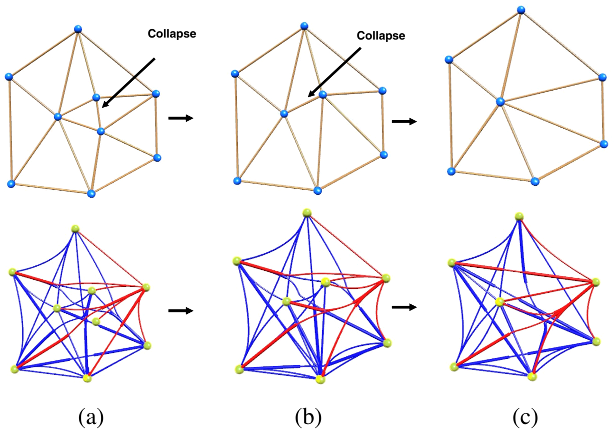


**Fig. 2.** Progressive edge clustering: (a) Traditional node-link diagram; (b) Collapse one Delaunay edge; (c) Collapse another Delaunay edge

For the new position of the collapsed nodes, we consider two choices. First, the nodes are still at their original positions. Only the "road" becomes unavailable so all links passing through this Delaunay edge have to go through the neighboring control points. Second, these two nodes overlap and share one position. For example, these two nodes can assume the position of one of the original nodes or are moved into a new position such as the middle point of the Delaunay edge. We provide an interface to allow users to decide the new position(s) of the nodes associated with the collapsed Delaunay edge. Please notice that our focus is not to cluster nodes, as there are many excellent papers on this topic. Therefore, these nodes only overlap in 2D space and links are still associated to their original nodes instead of one abstract virtual node. We want to group more links together by progressive edge collapsing so that different level-of-details for large networks can be achieved and the visual clutter problem can be alleviated. Figure 2 illustrates progressive edge clustering.

## 4 Conclusions and Future Work

In this paper, we proposed a geometry-based edge clustering method for traditional node-link diagrams. Our method is easy to implement and can generate visually appealing layouts for large networks in real time. By setting control points on Delaunay edges to control the flow of links we can easily obtain different levels of edge clustering. The classic ambiguity cases for node-link diagrams can be easily solved by setting a protected area for each node. We also proposed progressive edge clustering so that continuous level-of-details can be generated for large graphs.

In the future, we plan to further analyze the intersection points cached on Delaunay edges and automatically generate some road-map style layout for large graphs. Our method can be further improved if combined with some node layout adjustment algorithm to make edge merging more effective. We plan to use polylines with very small round corners to display clustered links. More sophisticated progressive edge clustering techniques which take the graph's topology into consideration will also be developed.

## Acknowledgments

This work is partially supported by HKUST grant DAG 04/05 EG02. We would like to thank the anonymous reviewers for their valuable comments.

## References

1. Carpendale, M., Rong, X.: Examining edge congestion. CHI '01 extended abstracts on Human factors in computing systems (2001) 115–116
2. Wong, N., Carpendale, M., Greenberg, S.: Edgelens: An interactive method for managing edge congestion in graphs. IEEE Symposium on Information Visualization 2003 (2003) 51–58
3. van Ham, F., van Wijk, J.J.: Interactive visualization of small world graphs. (2004) 199–206
4. Noack, A.: Energy-based clustering of graphs with nonuniform degrees. Graph Drawing (2005) 309–320
5. Dickerson, M., Eppstein, D., Goodrich, M.T., Meng, J.Y.: Confluent drawings: Visualizing non-planar diagrams in a planar way. J. Graph Algorithms Appl. **9**(1) (2005) 31–52
6. Phan, D., Xiao, L., Yeh, R., Hanrahan, P., Winograd, T.: Flow map layout. IEEE Symposium on Information Visualization 2005 (2005) 219–224
7. Wong, N., Carpendale, S.: Interactive poster: Using edge plucking for interactive graph exploration. Poster in the IEEE Symposium on Information Visualization (2005)
8. Hoppe, H.: Progressive meshes. Proceedings of ACM SIGGRAPH '96 (1996) 99–108

# Biclique Edge Cover Graphs and Confluent Drawings

Michael Hirsch, Henk Meijer, and David Rappaport

School of Computing, Queen's University
Kingston, Ontario, Canada

**Abstract.** Confluent drawing is a technique that allows some non-planar graphs to be visualized in a planar way. This approach merges edges together, drawing groups of them as single *tracks*, similar to train tracks. In the general case, producing confluent drawings automatically has proven quite difficult. We introduce the biclique edge cover graph that represents a graph $G$ as an interconnected set of cliques and bicliques. We do this in such a way as to permit a straightforward transformation to a confluent drawing of $G$. Our result is a new sufficient condition for confluent planarity and an additional algorithmic approach for generating confluent drawings. We give some experimental results gauging the performance of existing confluent drawing heuristics.

## 1 Introduction

In 2003, Dickerson, Eppstein, Goodrich, and Meng introduced confluent drawing, and with it a heuristic able to generate confluent drawings for some graphs [1]. These drawings present a novel way of visualizing non-planar graphs in a planar way, however, producing a *planar confluent drawing* for an arbitrary graph has proven to be quite difficult. Devine speculates that merely deciding whether such a drawing exists is NP-hard for an arbitrary graph [1]. Hui, Schaefer, and Stefankovic also speculate that this problem is NP-complete [2]. In this paper we explore alternate methods of automatically generating confluent drawings. We experimentally evaluate Dickerson et al.'s confluent drawing heuristic, as well as our own heuristics based on the *biclique edge cover graph*.

Francis Newbery proposed a method of merging together edges called *edge concentration* in a 1989 paper [3]. Dickerson et al. first introduced confluent drawings in [3]; they have been subsequently studied in [2,4,1,5].

This paper is organized as follows. Section 2 provides a brief background, Sect. 3 defines the biclique edge cover graph, and Sect. 4 gives a method to transform such a graph into a confluent drawing. Finally, Sect. 5 covers confluent drawing algorithm implementations and their experimental performance.

## 2 Background

We define the relevant concepts in confluent drawing: A *curve* is a continuous map into the plane. A curve is *smooth* if it is continuously differentiable along its length (there are no sharp bends) [6,2]. A drawing $D$ is a *confluent drawing* of an undirected graph $G$ if:

- There is a one to one mapping between vertices of $G$ and $D$.
- There exists a smooth curve between vertices $u$ and $w$ in $D$ if and only if there exists an edge $(u, w)$ in $E$.

Consistent with [4] and [1], we have omitted the planarity constraint found in Dickerson et al.'s definition [6]. We say that a confluent drawing is *planar* if no smooth curve(s) intersect at a single point (they may share overlapping portions). A confluent drawing is *non-planar* if any curve(s) intersect at a single point.

Lastly, we define a switch, and traffic circle, two basic confluent elements. A *switch* is the point where curves converge. A switch $s$ is defined to have degree three [2,1], but we generalize it to have arbitrary degree. A *traffic circle* is a particular confluent representation of a clique such that all smooth curves merge with a central circular track. Figure 1 depicts a switch (left) and a traffic circle (right).

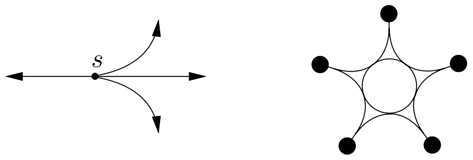


**Fig. 1.** A *switch* of degree four (left), and a confluent drawing of $K_5$ called a *traffic circle* (right)

## 3 Biclique Edge Cover Graph

Let $G = (V, E)$ be a graph. A clique $c$ is a subset of $V$ such that the subgraph induced by $c$ is a complete graph. We say that edge $e$ is in clique $c$ if it is in the subgraph induced by the vertices in $c$.

A biclique $(b_i, b_j)$ is an unordered pair of disjoint subsets $b_i$ and $b_j$ of $V$, such that for all $u \in b_i$ and $w \in b_j$, $(u, w) \in E$. We call each subset $b_i$ and $b_j$ a *b-part*. We say that edge $e$ is in biclique $(b_i, b_j)$ if it is incident to a vertex in each b-part.

Let $C$ be a set of cliques, and let $B$ be a set of bicliques such that each edge of $G$ is in a clique of $C$ or in a biclique of $B$. We say that such sets $B$ and $C$ together *edge cover* $G$. Given a set of bicliques $B$, $B_p$ is the set of b-parts such that for each $(b_i, b_j) \in B$, $b_i, b_j \in B_p$.

Let $G$ be a graph. Let $B$ be a set of bicliques and let $C = \{c_0, c_1, \ldots, c_{m-1}\}$ be a set of cliques that together edge cover $G$. Let b-parts $b_0, b_1, \ldots, b_{l-1}$ denote the elements of $B_p$. Let $\pi_0, \pi_1, \ldots, \pi_{2^{l+m}-1}$ denote the elements in the power set of $B_p \cup C$. We define the biclique edge cover graph $G_b = (V_b, E_b)$ as follows:

– Vertex $v_b = \pi_i$ is in $V_b$ if and only if there exists a vertex $v \in V$ such that $v$ is in every b-part and clique in $\pi_i$ and no others.
– Edge $e_b = (u_b, w_b)$ is in $E_b$ if and only if $u_b \neq w_b$, and $u_b \cap w_b \cap C \neq \emptyset$, or there exists a $b_i \in u_b$ and $b_j \in w_b$ such that $(b_i, b_j) \in B$.

We say that $v$ and $v_b$ are associated if a vertex $v \in V$ is in every b-part and clique in $v_b = \pi_j \in V_b$ and no others. Hereafter, $u, v, w$ and $v_0, v_1, \ldots, v_{n-1}$ denote the elements of $V$. Similarly, $u_b, v_b, w_b$ and $v_{b_0}, v_{b_1}, \ldots, v_{b_{n-1}}$ denote the elements of $V_b$.

## 3.1 Example

The following example illustrates the derivation of a biclique edge cover graph $G_b$ from graph $G$ (Fig. 2). Given a graph $G$, first determine a set of bicliques $B$, and cliques $C$. Note that any sets will suffice, provided that $B$ and $C$ together edge cover $G$. We choose $B = \{(\{v_0, v_1\}, \{v_2, v_3, v_4\})\}$ and $C = \{c_0\} = \{v_2, v_3, v_4, v_5, v_6\}$. Thus $B_p = \{b_0, b_1\} = \{\{v_0, v_1\}, \{v_2, v_3, v_4\}\}$.

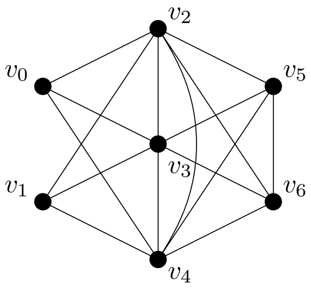


**Fig. 2.** Graph $G$

We construct the vertex set $V_b$: The vertex $v_0 \in b_0$, $v_0 \notin b_1 \cap c_0$. By our definition, $v_0$ establishes $\{b_0\} \in V_b$, and $v_0$ and $\{b_0\}$ are associated. Vertex $v_1$ also establishes $\{b_0\} \in V_b$. Vertices $v_2, v_3, v_4$ each establish $\{b_1, c_0\} \in V_b$. Vertices $v_5$ and $v_6$ establish $\{c_0\} \in V_b$. Thus $V_b = \{v_{b_0}, v_{b_1}, v_{b_2}\} = \{\{b_0\}, \{b_1, c_0\}, \{c_0\}\}$. The edge set $E_b = \{(v_{b_1}, v_{b_2}), (v_{b_0}, v_{b_1})\}$. Figure 3 depicts biclique edge cover graph $G_b$. We use a solid vertex to depict any $v_b \cap C \neq \emptyset$, and an unfilled vertex to depict any $v_b \cap C = \emptyset$.

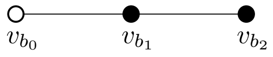


**Fig. 3.** Derived biclique edge cover graph $G_b$

**Lemma 1.** *Let $G$ be a graph. Let $B$ be a set of bicliques and let $C$ be a set of cliques that together edge cover $G$. Let $G_b$ be the resulting biclique edge cover graph. The following properties hold:*

1. *The magnitude $|V_b| \leq |V|$.*
2. *Vertices $v_0, v_1, \ldots, v_{r-1} \in V$ associated with a vertex $v_b \in V_b$ define an independent set in $G$ where $v_b \cap C = \emptyset$; otherwise, they define a clique in $G$.*

*Proof.* Property (1) follows from our definition of a biclique edge cover graph: Each vertex $v \in V$ may only be associated with a single vertex $v_b \in V_b$. Each vertex $v_b \in V_b$ is associated with at least one vertex $v \in V$. It follows that $|V_b| \leq |V|$.

We prove Property (2) by contradiction. Let $u$ and $w$ be adjacent vertices in $V$, such that $u$ and $w$ are associated with $v_b \in V_b$ and $v_b \cap C = \emptyset$. By definition, edge $e = (u, w)$ is in a biclique in $B$ or a clique in $C$. If $e$ is in a clique $c \in C$, then $c$ must be an element of $v_b$. This is contrary to $v_b \cap C = \emptyset$. If $e$ is in a biclique $(b_i, b_j) \in B$, it follows that $u$ must be an element of $b_i$ and $w$ an element of $b_j$. Recall that b-parts $b_i$ and $b_j$ are disjoint. Vertices $u$ and $w$ cannot therefore both be associated with $v_b$. Where $v_b \cap C \neq \emptyset$, Property (2) follows directly from the definition. □

## 4 Generating Confluent Drawings

In this section we show how to construct a drawing $D$ of $G$ from a drawing of its biclique edge cover graph $G_b$. Let $G$ be a graph. Let $B$ be a set of bicliques and let $C$ be a set of cliques that together edge cover $G$. Let $G_b$ be the resulting biclique edge cover graph. Let $D_b$ be a drawing of $G_b$. Note that drawing $D_b$ could be a traditional drawing or a confluent drawing of $G_b$. Replace each vertex $v_{b_i} \in D_b$ by vertices of $G$ as follows:

We will compose a confluent structure. Begin with a single circular track. Join each vertex in $V$ associated with $v_{b_i}$ to the circular track:

– If $v_{b_i} \cap C \neq \emptyset$: Join each vertex in $V$ associated with $v_{b_i}$ to the circular track by means of two smooth curves such that one curve may be followed onto, and around the adjoined circular track in the clockwise direction, and the other in the counterclockwise direction. We call this construction a *traffic circle.*

– If $v_{b_i} \cap C = \emptyset$: Join each vertex in $V$ associated with $v_{b_i}$ to the circular track by means of a smooth curve such that the curve may be followed onto, and around the adjoined circular track in the counterclockwise direction. We call this construction a *counterclockwise traffic circle.*

Remove vertex $v_{b_i}$ from drawing $D_b$, and put the composed confluent structure in its place. Merge all (confluent) edges previously incident to $v_{b_i}$ with the circular track such that each edge may be followed onto, and around the adjoined circular track in the clockwise direction. Figure 4 illustrates our replacement method for a vertex $v_{b_i} \in D_b$ with two incident edges, and three associated vertices in $V$.

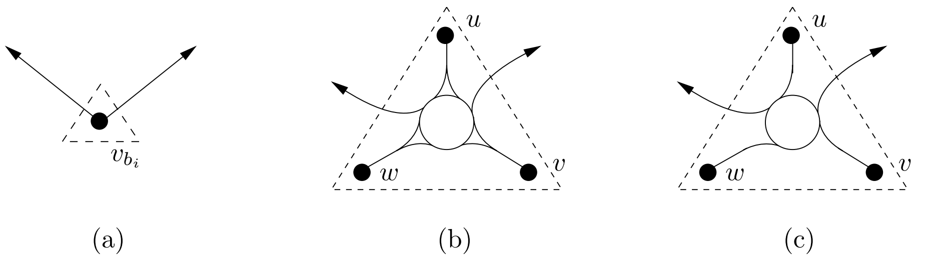


**Fig. 4.** (a) Vertex $v_{b_i}$ in drawing $D_b$ (b) Replacement of $v_{b_i}$ in drawing $D$ where $v_{b_i} \cap C \neq \emptyset$ (c) Replacement of $v_{b_i}$ in drawing $D$ where $v_{b_i} \cap C = \emptyset$

## 4.1 Example

We continue with the example of Sect. 3.1. We generate a confluent drawing of $G$ from a drawing of $G_b$ (Fig. 3). Each vertex of $G_b$ is replaced by all associated vertices of $G$. The result is drawing $D$, Fig. 5.

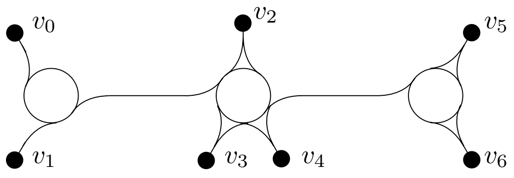


**Fig. 5.** Confluent drawing $D$ generated from a drawing of $G_b$

**Lemma 2.** *Let $D$ be a drawing generated from a drawing of $G_b$ by the method of this section. Drawing $D$ is a confluent drawing of $G$.*

*Proof.* We first show that a one to one mapping exists between vertices of $G$ and $D$. We then show that there exists a smooth curve between vertices $u$ and $w$ in $D$ if and only if there exists an edge $(u, w)$ in $E$. We present this argument in two cases.

Our method replaces each vertex $v_b \in V_b$ by all associated vertices in $V$. Because each vertex in $V$ is associated with a single vertex in $V_b$, the vertex set of $D$ is precisely that of $G$.

*Case I.* We will show that there exists a smooth curve between vertices $u$ and $w$ in $D$ if there exists an edge $(u, w)$ in $E$. Vertices $u$ and $w$ are either associated with the same vertex, or two different vertices in $V_b$. If they are associated with the same vertex $v_{b_i} \in V_b$ and $v_{b_i} \cap C \neq \emptyset$, then our method sees that $u$ and $w$ are in the same constructed traffic circle. A smooth curve therefore exists between $u$ and $w$ in $D$. Otherwise (if $v_{b_i} \cap C = \emptyset$) Lemma 1 precludes $(u, w)$ from being an edge in $E$.

We now show that a smooth curve exists between $u$ and $w$ if they are associated with different vertices in $V_b$: $u$ with $u_b$ and $w$ with $w_b$. By definition, edge $(u, w)$ is in a biclique in $B$ or a clique in $C$. If $(u, w)$ is in a clique $c \in C$, then clique $c$ is an element of both $u_b$ and $w_b$. Vertices $u_b, w_b$ are therefore adjacent. If $(u, w)$ is in a biclique $(b_i, b_j) \in B$, then $b_i$ is an element of $u_b$ and $b_j$ an element of $w_b$. Vertices $u_b, w_b$ are again adjacent. Our method ensures that the (confluent) edge between $u_b$ and $w_b$ is merged with the circular track that replaces $u_b$ and the circular track that replaces $w_b$. This merged edge completes a smooth curve between $u$ and $w$ in $D$.

*Case II.* We will show that there exists an edge $(u, w)$ in $E$ if there exists a smooth curve between vertices $u$ and $w$ in $D$. Our method generates smooth curves in $D$ in two ways. First, constructed traffic circles consist of smooth curves between vertices of $V$. Two vertices $u$ and $w \in V$ are in the same traffic circle only if they are associated with the same vertex in $V_b$. It follows from Lemma 1 that edge $(u, w) \in E$.

Additionally, smooth curves connect traffic circles/circular tracks that have replaced adjacent vertices in $V_b$. Because (confluent) edges are always merged with these confluent structures in the same direction, a smooth curve never connects two structures that have replaced non-adjacent vertices in $V_b$. Smooth curves in $D$ therefore connect vertices $u$ and $w$ that are associated with adjacent vertices in $V_b$. If vertices $u_b, w_b \in V_b$ are adjacent, then there exists a clique $c \in u_b \cap w_b$ or there exist b-parts $b_i \in u_b$ and $b_j \in w_b$ such that $(b_i, b_j) \in B$. If $c \in u_b \cap w_b$, then any vertex associated with either $u_b$ or $w_b$ is in $c$. Otherwise, if $b_i \in u_b$ and $b_j \in w_b$, then $u$ is an element of $b_i$ and $w$ an element of $b_j$. In either case, it follows that edge $(u, w) \in E$. □

## 4.2 Planarity

**Lemma 3.** *Let $D$ be a drawing generated from a drawing of $G_b$ by the method of this section. If the drawing of $G_b$ is confluent planar then $D$ is confluent planar.*

*Proof.* Our method replaces vertices in the drawing of $G_b$ by confluent planar structures to produce $D$. No edge crossings exist in the drawing of $G_b$, and none are introduced by our replacement scheme. Drawing $D$ is therefore a confluent planar drawing of $G$. □

**Corollary 1.** *If the drawing of $G_b$ is planar then $D$ is confluent planar.*

*Proof.* A planar graph satisfies the definition of confluent planarity. □

## 4.3 Prickly Clique

A *prickly clique* consists of a clique and one additional vertex adjacent to each vertex in the clique. More formally, a prickly clique is a graph $G$ on $2n$ vertices

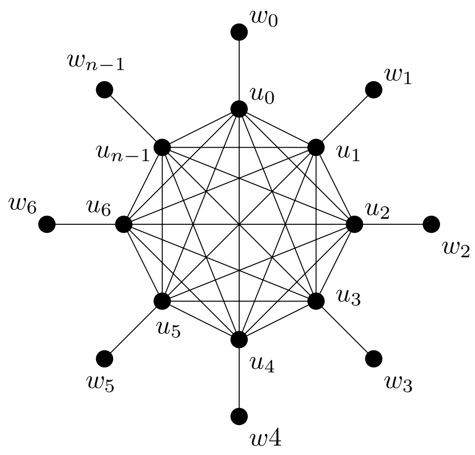


**Fig. 6.** Prickly clique $G$ for $n = 8$

with $n \geq 2$ such that $V = \{u_0, u_1, \ldots u_{n-1}, w_0, w_1, \ldots, w_{n-1}\}$, and $E = E_0 \cup E_1$ where $(u_i, w_i) \in E_0$ for $0 \leq i < n$ and $(u_i, u_j) \in E_1$ for $0 \leq i < j < n$. The prickly clique for $n \geq 5$ is an example of a confluent planar graph that does not have a resulting planar biclique edge cover graph.

**Lemma 4.** *Let $G$ be a prickly clique. Let $B$ be a set of bicliques and let $C$ be a set of cliques that together edge cover $G$. Let $G_b$ be the resulting biclique edge cover graph. Graph $G$ is isomorphic to $G_b$.*

*Proof.* We construct the vertex set $V_b$. Each edge $(u_i, w_i) \in E_0$ is either a clique in $C$ or a biclique in $B$, while each edge $(u_i, u_j) \in E_1$ is in at least one biclique in $b$ or clique in $C$. Thus, each vertex $w_i \in G$ establishes a vertex $w_{b_i} \in V_b$, while each vertex $u_i \in G$ establishes a vertex $u_{b_i} \in V_b$. This defines a bijection $u_i \rightarrow u_{b_i}$, $w_i \rightarrow w_{b_i}$, from $V$ to $V_b$.

We construct the edge set $E_b$. Edge $(u_{b_i}, w_{b_i}) \in E_b$ for $0 \leq i < n$ (if $\{u_i, w_i\} \in C$ then $u_{b_i} \cap w_{b_i} \cap \{u_i, w_i\} \neq \emptyset$; otherwise if $\{\{u_i\}, \{w_i\}\} \in B$ then $\{u_i\} \in u_{b_i}$ and $\{w_i\} \in w_{b_i}$). Moreover, edge $(u_{b_i}, u_{b_j}) \in E_b$ for $0 \leq i < j < n$ (if $(u_i, u_j)$ is in a clique $c \in C$ then $u_{b_i} \cap u_{b_j} \cap c \neq \emptyset$; otherwise if $(u_i, u_j)$ is in a biclique $b = (b_i, b_j) \in B$ then $b_i \in u_{b_i}$ and $b_j \in u_{b_j}$). □

## 5 Implementation and Results

In this section we examine two algorithmic approaches for generating confluent drawings. We will examine the experimental performance of each implemented algorithm, and conclude with some sample outputs.

### 5.1 ConfluentDickerson(*G*)

Algorithm 1 is based on the only previously published confluent drawing algorithm for undirected graphs. Presented by Matthew Dickerson at the 11th International Symposium on Graph Drawing, the heuristic iteratively identifies and replaces large cliques and bicliques (complete and bipartite subgraphs) with equivalent confluent structures [6].

Our implementation uses an $O(nm\mu)$ solution (where $\mu$ is the number of maximal independent sets of a graph) to the maximal independent sets problem by Tsukiyama et al. [7] to enumerate all maximal cliques and bicliques. This solution yields all maximal cliques when applied to a graph's complement, and all maximal bicliques when applied to its *double cover* [8]. Note that Dickerson et al. [6] specified an algorithm by Chiba and Nishizeki [9] for identifying cliques and a second algorithm by Eppstein [10] for identifying bicliques.

**Algorithm 1**: ConfluentDickerson($G$)

```
Input: A connected graph $G = (V, E)$
Output: A confluent drawing of $G$

done ← false;
while !done and G is non-planar do
    $C$ ← all maximal cliques of $G$;
    foreach clique $c \in C$ in order of decreasing size do
        if there exists an edge in $E$ between each pair of vertices in $c$ then
            Remove all edges from $E$ between pairs of vertices in $c$;
            Add a vertex $u$ to $V$; denote it as a traffic circle; Add an edge to
            $E$ between $u$ and each vertex in $c$;
            done ← false;
    $B$ ← all maximal bicliques of $G$;
    foreach biclique $(b_i, b_j) \in B$ in order of decreasing size do
        if there exists an edge in $E$ between each pair of vertices $(v, w)$,
        where $v \in b_i$, $w \in b_j$ then
            Remove each edge $(v, w)$ from $E$ where $v \in b_i$, $w \in b_j$;
            Add vertices $v$ and $w$ to $V$; denote each as a switch;
            Add an edge $(v, w)$ to $E$;
            Add an edge to $E$ between $v$ and each vertex in $b_i$;
            Add an edge to $E$ between $w$ and each vertex in $b_j$;
            done ← false;
Draw $G$;
```

### 5.2 ConfluentHirsch(*G*)

Algorithm 2 is an implementation of the algorithm presented in Secs. 3 and 4. The algorithm first randomly computes a set of cliques and bicliques that together edge cover $G$. It then determines the vertex set of the biclique edge cover graph. Each of these vertices is inserted into $G$, joined to all associated vertices, and denoted as a *traffic circle* or *counterclockwise traffic circle.*

**RecursiveHirsch($G, i$).** Beginning with $G$, this variation of $ConfluentHirsch$ $(G)$ iteratively computes $i$ successive biclique edge cover graphs. A confluent drawing of $G$ is recursively constructed using the algorithm in Sec 4. See [11] for details.

**DiscardHirsch($G$).** This second variation discards cliques $|c| \leq 3$ from $C$ and bicliques $|b_i|, |b_j| \leq 1$ from $B$. Intuitively, including these degenerate cases can only hamper the performance of our algorithm. Any vertices that are no longer in a clique or biclique after the discard are effectively ignored by the algorithm.

**Algorithm 2**: ConfluentHirsch($G$)

```
Input: A connected graph $G = (V, E)$
Output: A confluent drawing of $G$

Let $V$ be an array of vertices of $G$ and let $V_b$ be an array of collections;
Let $C$ be a collection of cliques, and let $B$ be a collection of bicliques;
Let $B_p$ be a collection b-parts such that for all $(b_i, b_j) \in B$, $b_i, b_j \in B_p$;
*A collection is a single object that contains multiple elements.

foreach edge $e \in E$ do
    if edge $e$ is not yet covered by a clique $\in C$ or a biclique $\in B$ then
        Randomly expand $e$ into a maximal clique or biclique and
        accordingly add it to $C$ or $B$;
Remove all edges from $E$;
foreach set $s \in C \cup B_p$ do
    foreach vertex $v \in s$ do
        Add set $s$ to $V_b[V.indexOf(v)]$;
foreach unique collection $u_b \in V_b$ do
    if $u_b \cap C \neq \emptyset$ then
        Add a vertex $u_b$ to $V$; denote it as a traffic circle;
    else if $u_b \cap C = \emptyset$ then
        Add a vertex $u_b$ to $V$; denote it as counterclockwise traffic circle;
    Add an edge to $E$ between vertex $u_b$ and each vertex $v$ where
    $V_b[V.indexOf(v)] = u_b$;
Add an edge to $E$ between any two vertices $u$ and $w$ where $u \cap w \cap C \neq \emptyset$;
Add an edge to $E$ between any two vertices $u$ and $w$ where $u \in b_i$ and
$w \in b_j$ such that $(b_i, b_j) \in B$;
Draw $G$;
```

### 5.3 Experimental Results

We have applied our drawing algorithm implementations to two sets of graphs in order to measure their performance[1]. Table 1 summarizes results for the *Rome graphs* [12], and Table 2 for the *ATT graphs* (http://www.graphdrawing.org).

Because ConfluentHirsch($G$) and DiscardHirsch($G$) are non-deterministic, they were allowed multiple attempts per input to produce a confluent planar output. For a given case, a single confluent planar output was recorded if at least one attempt produced a confluent planar output. Note that RecursiveHirsch($G, i$) is also non-deterministic, however, multiple recursive iterations ensure that its output is not determined by one random set of cliques and bicliques.

**Table 1.** Performance of confluent drawing algorithms on the Rome set

| Algorithm applied | Non-planar inputs | Attempts per input | Confluent planar outputs | Confluent planar outputs not found by ConfluentDickerson($G$) |
|---|---|---|---|---|
| ConfluentDickerson(G) | 8253 | 1 | 210 | - |
| ConfluentHirsch(G) | 8253 | 10 | 9 | 1 |
| ConfluentHirsch(G) | 8253 | 100 | 10 | 1 |
| RecursiveHirsch(G,10) | 8253 | 1 | 10 | 1 |
| RecursiveHirsch(G,100) | 8253 | 1 | 10 | 1 |
| DiscardHirsch(G) | 8253 | 10 | 115 | 22 |

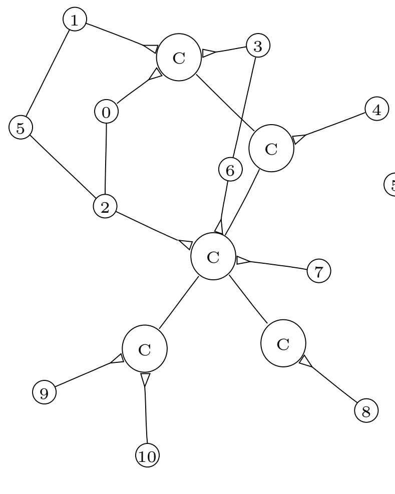
**Table 2.** Performance of confluent drawing algorithms on the ATT set

| Algorithm applied | Non-planar inputs | Attempts per input | Confluent planar outputs | Confluent planar outputs not found by ConfluentDickerson($G$) |
|---|---|---|---|---|
| ConfluentDickerson(G) | 423 | 1 | 166 | - |
| ConfluentHirsch(G) | 423 | 10 | 48 | 0 |
| ConfluentHirsch(G) | 423 | 100 | 53 | 0 |
| RecursiveHirsch(G,10) | 423 | 1 | 57 | 1 |
| RecursiveHirsch(G,100) | 423 | 1 | 61 | 2 |
| DiscardHirsch(G) | 423 | 10 | 129 | 5 |

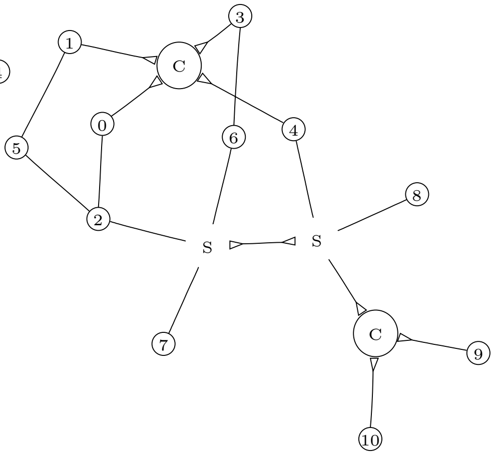
Figures 7 and 8 were output by our implementation.[2] Switches are denoted $S$, with an arrowhead marking the incident edge along which the other incident edges converge. Traffic circles are denoted $C$:

[1] Planarity was determined using the Lempel-Even-Cederbaum planarity test implementation included as part of GTL, the *Graph Template Library* (http://www. infosun.fmi.uni-passau.de/GTL).

[2] Our implementation uses the *Graphviz* [13] package to produce layouts and drawings, as well as the *Grappa* [14] package for working with graphs in Java.



(a) DiscardHirsch($G$)



(b) ConfluentDickerson($G$)

**Fig. 7.** Confluent drawings of an example given by Dickerson et al. in [6]. Spring embedder layout computed using *Graphviz* [13].

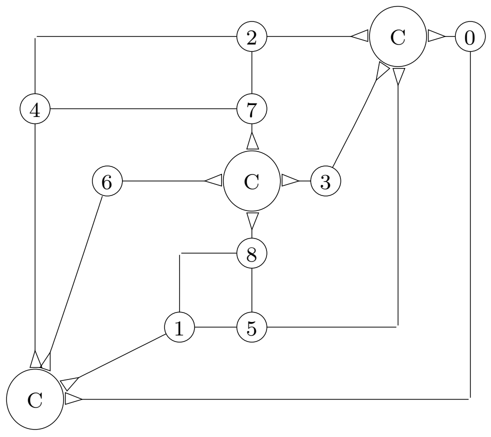


**Fig. 8.** The smallest known confluent non-planar graph is the Peterson graph with one vertex and adjacent edges removed [12]. Above, a confluent planar drawing of its complement generated by ConfluentDickerson($G$). Layout computed using the dominance-polyline method for general undirected planar graphs in [15].

## 6 Conclusion

The performance of the algorithms varied, with ConfluentDickerson($G$) producing the greatest number of confluent planar drawings for both sets of graphs.

Each variation of ConfluentHirsch($G$) was however able to produce confluent planar results for some inputs where ConfluentDickerson(G) was not. Our results seem to confirm that confluent drawings offer a valid means for drawing non-planar graphs in a planar way for some inputs. Confluent drawings can however be more difficult to read than traditional drawings. This holds true even for cases where a confluent drawing is planar and the original graph is not.

## References

1. Devine, J.: Confluent graphs. Master's thesis, Queen's University (2005)
2. Hui, P., Schaefer, M., Štefankovič, D.: Train tracks and confluent drawings. In Pach, J., ed.: Proc. 12th Int. Symp. Graph Drawing (GD 2004). Number 3383 in Lecture Notes in Computer Science, Springer-Verlag (2004) 318–328
3. Newbery, F.J.: Edge concentration: a method for clustering directed graphs. In: Proc. 2nd Int. Works. on Soft. configuration management, ACM Press (1989) 76–85
4. Eppstein, D., Goodrich, M.T., Meng, J.Y.: Confluent layered drawings. In Pach, J., ed.: Proc. 12th Int. Symp. Graph Drawing (GD 2004). Number 3383 in Lecture Notes in Computer Science, Springer-Verlag (2004) 184–194
5. Eppstein, D., Goodrich, M.T., Meng, J.Y.: Delta-confluent drawings. In Healy, P., Nikolov, N.S., eds.: Proc. 13th Int. Symp. Graph Drawing (GD 2005). Number 3843 in Lecture Notes in Computer Science, Springer-Verlag (2006) 165–176
6. Dickerson, M.T., Eppstein, D., Goodrich, M.T., Meng, J.Y.: Confluent drawings: visualizing non-planar diagrams in a planar way. In: Proc. 11th Int. Symp. Graph Drawing (GD 2003). Lecture Notes in Computer Science, Springer-Verlag (2003)
7. Tsukiyama, S., Ide, M., Ariyoshi, H., Shirakawa, I.: A new algorithm for generating all the maximal independent sets. In: SIAM Journal on Computing. Volume 6. (1977) 505–517
8. Alexe, G., Alexe, S., Crama, Y., Foldes, S., Hammer, P.L., Simeone, B.: Consensus algorithms for the generation of all maximal bicliques. Discrete Applied Mathematics **145**(1) (2004) 11–21
9. Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM J. Comput. **14**(1) (1985) 210–223
10. Eppstein, D.: Arboricity and bipartite subgraph listing algorithms. Information Processing Letters **51**(4) (1994) 207–211
11. Hirsch, M.: Generating confluent drawings: Theory and practice. Master's thesis, Queen's University (2006)
12. Di Battista, G., Garg, A., Liotta, G.: An experimental comparison of three graph drawing algorithms (extended abstract). In: Proc. 11th Annual Symp. Computational Geometry (SCG '95), New York, NY, USA, ACM Press (1995) 306–315
13. Gansner, E.R., North, S.C.: An open graph visualization system and its applications to software engineering. Softw. Pract. Exper. **30**(11) (2000) 1203–1233
14. Barghouti, N.S., Mocenigo, J., Lee, W.: Grappa: A graph package in java. In Di Battista, G., ed.: Proc. 5th Int. Symp. Graph Drawing (GD '97). Volume 1353 of Lecture Notes in Computer Science., Springer (1997) 336–343
15. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, NJ, USA (1998)

# Schnyder Woods and Orthogonal Surfaces

Stefan Felsner and Florian Zickfeld

Technische Universität Berlin, Fachbereich Mathematik
Straße des 17. Juni 136, 10623 Berlin, Germany
{felsner,zickfeld}@math.tu-berlin.de

**Abstract.** In this paper we study connections between Schnyder woods and orthogonal surfaces. Schnyder woods and the face counting approach have important applications in graph drawing and dimension theory. Orthogonal surfaces explain the connections between these seemingly unrelated notions. We use these connections for an intuitive proof of the Brightwell-Trotter Theorem which says that the face lattice of a 3-polytope minus one face has dimension three. Our proof yields a companion linear time algorithm for the construction of the three linear orders that realize the face lattice.

Coplanar orthogonal surfaces are in correspondance with a large class of convex straight line drawings of 3-connected planar graphs. We show that Schnyder's face counting approach with weighted faces can be used to construct all coplanar orthogonal surfaces and hence the corresponding drawings. Appropriate weights are computable in linear time.

## 1 Introduction

In two fundamental papers [15,16] Schnyder developed a theory of Schnyder labelings and Schnyder woods for planar triangulations. The second paper deals with grid drawings of planar graphs and contains the first of numerous applications of Schnyder woods in the area of graph drawing. For example, the results in [1], [2], [5], [13] use Schnyder woods, and more references can be found in [6].

In [15], Schnyder presented a characterization of planar graphs in terms of order dimension. We briefly introduce the terminology needed for the statement of this result: With a graph $G = (V, E)$, associate an order $P_G$ of height two on the set $V \cup E$. The order relation is defined by setting $x < e$ in $P_G$ if $x \in V$, $e \in E$ and $x \in e$. The order $P_G$ is called the *incidence order* of $G$.

The *dimension* of an order $P$ is the least $k$ such that $P$ admits an order preserving embedding in $\mathbb{R}^k$ equipped with the *dominance order*. In the dominance order we have that $u \leq v$ if and only if $u_i \leq v_i$ holds for each component $i$. For more on order dimension see [17], [3] or [7].

**Theorem 1 (Schnyder's Theorem).** *A graph is planar if and only if the dimension of its incidence order is at most three.*

In the same paper Schnyder also shows that the incidence poset of vertices, edges and faces of a planar triangulation has dimension four, but the dimension drops to three upon removal of a face. Brightwell and Trotter [4] extended Schnyder's

Theorem to the general case of embedded planar multigraphs. The main building block for the proof is the case of 3-connected planar graphs.

**Theorem 2 (Brightwell-Trotter Theorem).** *The incidence order of the vertices, edges and faces of a 3-connected planar graph $G$ has dimension four. Moreover, if $F$ is a face of $G$, then the incidence order of the vertices, edges and all faces of $G$ except $F$ has dimension three.*

Note that, by Steinitz's Theorem, the incidence poset of vertices, edges and faces of a 3-connected planar graph is just the face lattice of the corresponding 3-polytope with **0** and **1** removed.

The original proof of Theorem 2 in [3] was long and technical, and Felsner gave a simpler one in [7]. Consecutively, Felsner [8] showed that every Schnyder wood of a 3-connected planar graph is supported by a rigid orthogonal surface (Theorem 5). An orthogonal surface is called rigid if it supports a unique graph, see Figures 3 and 4. By a result of Miller [14], Felsner's result implies Theorem 2. In Section 3 we present an intuitive proof of Theorem 5, that leads to a simple linear time algorithm for the computation of the rigid surface. The reduction to topological sorting it uses is simpler and more efficient than the constructions that are implicit in the other proofs. The idea is to start with the orthogonal surface $\mathfrak{S}$ obtained from a Schnyder wood $S$ by face counting. If this surface is non-rigid it is possible to make some local adjustments at a non-rigid edge by moving some of the flats up or down in the direction of their normal vector, see Figure 4. The nontrivial point is to show that these adjustments can be combined in such a way that the whole surface becomes rigid.

The rest of the paper is organized as follows. In Section 2 we give definitions and a brief introduction into the structural properties of Schnyder woods and orthogonal surfaces which are required for the discussion in the later parts of this paper. For a more detailed introduction we refer the reader to [9].

As mentioned above, Section 3 deals with rigid orthogonal surfaces. Section 4 is concerned with coplanar surfaces, that is orthogonal surfaces with the property that all generating minima lie on some plane. The interest in this class originates from their close connection to planar straight line drawings. Connecting the minima of a coplanar surface by straight line segments yields a plane and convex straight line drawing of the graph. Similar approaches for non-coplanar surfaces fail as the drawings need not be crossing-free. We show that all coplanar surfaces supporting $S$ can be obtained using Schnyder's original construction with appropriately weighted faces. At the end of the section, we give an example of a Schnyder wood that has no supporting orthogonal surface which is simultaneously rigid and coplanar. We conclude with a related open problem.

Some proofs are omitted in this paper, others are considerably shortened. Complete proofs can be found in the full version [10].

## 2 Basics on Schnyder Woods and Orthogonal Surfaces

All the proofs omitted in the this section can be found in [9], [8] or [7]. A *planar map $M$* is a simple planar graph $G$ together with a fixed planar embedding of

$G$. Let $a_1, a_2, a_3$ be three vertices occurring in clockwise order on the outer face of $M$. A suspension $M^\sigma$ is obtained by attaching a half-edge that reaches into the outer face to each of these *special vertices*.

Let $M^\sigma$ be a suspended 3-connected planar map. A *Schnyder wood* rooted at $a_1, a_2, a_3$ is an orientation and coloring of the edges of $M^\sigma$ with the colors 1, 2, 3 (alternatively: red, green, blue) satisfying the following rules[1].

(W1) Every edge $e$ is oriented in one directin or in two opposite directions. The directions of edges are colored such that if $e$ is bidirected the two directions have distinct colors.

(W2) The half-edge at $a_i$ is directed outwards and colored $i$.

(W3) Every vertex $v$ has outdegree one in each color. The edges $e_1, e_2, e_3$ leaving $v$ in colors 1, 2, 3 occur in clockwise order. Each edge entering $v$ in color $i$ enters $v$ in the clockwise sector from $e_{i+1}$ to $e_{i-1}$.

(W4) There is no interior face the boundary of which is a directed monochromatic cycle.

We will sometimes refer to the Schnyder wood of a planar map, without choosing a suspension explicitly. Let $M$ be a planar map with a Schnyder wood. Let $T_i$ denote the digraph induced by the directed edges of color $i$. Every inner vertex has outdegree one in $T_i$. Therefore, every $v$ is the starting vertex of a unique $i$-path $P_i(v)$ in $T_i$. The next lemma implies that each of the digraphs $T_i$ is acyclic, and hence the $P_i(v)$ are simple paths.

**Lemma 1.** *Let $M$ be a planar map with a Schnyder wood $(T_1, T_2, T_3)$. Let $T_i^{-1}$ be obtained by reversing all edges from $T_i$. The digraph $D_i = T_i \cup T_{i-1}^{-1} \cup T_{i+1}^{-1}$ is acyclic for $i = 1, 2, 3$.*

By the rule of vertices (W3) every vertex has out-degree one in $T_i$. Disregarding the half-edge at $a_i$, this makes $a_i$ the unique sink of $T_i$. Since $T_i$ is acyclic and has $n - 1$ edges we obtain:

**Corollary 1.** *$T_i$ is a directed tree rooted at $a_i$, for $i = 1, 2, 3$.*

The $i$-path $P_i(v)$ of a vertex $v$ is the unique path in $T_i$ from $v$ to the root $a_i$. Lemma 1 implies that for $i \neq j$ the paths $P_i(v)$ and $P_j(v)$ have $v$ as the only common vertex. Therefore, $P_1(v), P_2(v), P_3(v)$ divide $M$ into three regions $R_1(v)$, $R_2(v)$, and $R_3(v)$, where $R_i(v)$ denotes the region bounded by and including the two paths $P_{i-1}(v)$ and $P_{i+1}(v)$, see Figure 1.

**Lemma 2.** *If $u$ and $v$ are vertices with $u \in R_i(v)$, then $R_i(u) \subseteq R_i(v)$. The inclusion is proper if $u \in R_i(v) \setminus (P_{i-1}(v) \cup P_{i+1}(v))$.*

**Lemma 3.** *If the directed edge $e = (u, v)$ is colored $i$, then $R_i(u) \subset R_i(v)$, $R_{i-1}(u) \supseteq R_{i-1}(v)$ and $R_{i+1}(u) \supseteq R_{i+1}(v)$. At least one of the latter two inclusions is proper.*

[1] We assume a cyclic structure on the colors so that $i+1$ and $i-1$ are always defined.

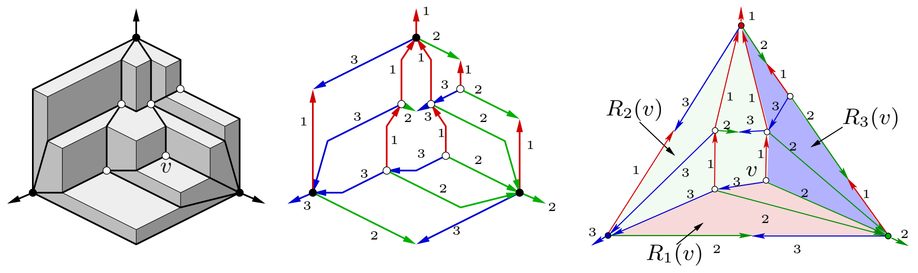


**Fig. 1.** An orthogonal surface, the induced Schnyder wood and the regions of a vertex $v$ in this Schnyder wood. The small numbers correspond to edge colors.

These lemmas are crucial for the applications of the *face-count vector* $(v_1, v_2, v_3)$ of a vertex $v$ with respect to a Schnyder wood which is defined by

$$v_i = \text{ the number of faces of } M \text{ contained in region } R_i(v).$$

Later we will use this vector to construct orthogonal surfaces supporting a given Schnyder wood. In that context $\{(v_1, v_2, v_3) \mid v \in V\}$ will be the generating set for the surface.

Another tool needed from the theory of Schnyder woods is the *edge split*. The following lemma from [2] describes the generic face in a Schnyder wood.

**Lemma 4.** *Given a Schnyder wood $S$ let $F$ be an interior face. The edges on the boundary of $F$ can be partitioned into six sets occurring in clockwise order around $F$. The sets are defined as follows (in case of bidirected edges the clockwise color is noted first): One edge from the set* {red-cw, blue-ccw, red-blue}*, any number (possibly 0) of edges* green-blue*, one edge from the set* {green-cw, red-ccw, green-red}*, any number of edges* blue-red*, one edge from the set* {blue-cw, green-ccw, blue-green}*, any number of edges* red-green*. The three edges from the first, third, and fifth set are the* special edges *of the face.*

Given a Schnyder wood $S$ let $e$ be a bidirected edge such that one of its directions is colored $j$ and $F$ be the incident face to which $e$ is not special. Choose a vertex $w$ of $F$ such that the angle of $w$ in $F$ is labeled $j$. To *split $e$ towards $w$* is to divide the bidirected edge $e$ into two uni-directed copies and to move the head of the $j$ colored copy to connect to $w$. Figure 2 illustrates the operation.

**Lemma 5.** *Let $S$ be a Schnyder wood and $e$ a bidirected edge of $S$. Then, splitting $e$ yields a Schnyder wood on the resulting graph.*

We now introduce orthogonal surfaces and review some facts that we will need in the sequel. Consider $\mathbb{R}^3$ equipped with the dominance order. We write $u \vee v$ and $u \wedge v$ to denote the *join* (component-wise maximum) and *meet* (component-wise minimum) of $u, v \in \mathbb{R}^3$. Let $\mathcal{V} \subset \mathbb{R}^3$ be an antichain, i.e., a set of pairwise incomparable elements. The *filter* generated by $\mathcal{V}$ in $\mathbb{R}^3$ is the set

$$\langle \mathcal{V} \rangle = \{\alpha \in \mathbb{R}^3 \mid \alpha \geq v \text{ for some } v \in \mathcal{V}\}.$$

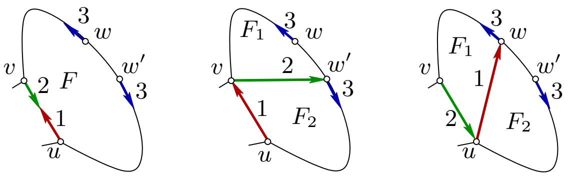


**Fig. 2.** The two possible types of splits of a non-special bidirected red-green edge $uv$ in $F$. The numbers in the figure correspond to the edge colors.

The boundary $\mathfrak{S}_{\mathcal{V}}$ of $\langle \mathcal{V} \rangle$ is the *orthogonal surface* generated by $\mathcal{V}$, see Figure 1.

If $u, v \in \mathcal{V} \subset \mathfrak{S}_{\mathcal{V}}$ and $u \vee v \in \mathfrak{S}_{\mathcal{V}}$, then $\mathfrak{S}_{\mathcal{V}}$ contains the union of the two line segments joining $u$ and $v$ to $u \vee v$; we refer to such arcs as *elbow geodesics* in $\mathfrak{S}_{\mathcal{V}}$. The *orthogonal arc* of $v \in \mathcal{V}$ in direction of the standard basis vector $e_i$ is the piece of the ray $v + \lambda e_i$, $\lambda \geq 0$, which follows a crease of $\mathfrak{S}_{\mathcal{V}}$. Clearly every vector $v \in \mathcal{V}$ has exactly three orthogonal arcs, one parallel to each coordinate axis. Some orthogonal arcs are unbounded while others are bounded. Observe that $u \vee v$ shares two coordinates with at least one (and perhaps both) of $u$ and $v$, so every elbow geodesic contains at least one bounded orthogonal arc.

Let $M$ be a planar map. A drawing $M \hookrightarrow \mathfrak{S}_{\mathcal{V}}$ is a *geodesic embedding* of $M$ into $\mathfrak{S}_{\mathcal{V}}$, if the following axioms are satisfied:

(G1) *Vertex axiom.* There is a bijection between the vertices of $M$ and $\mathcal{V}$.

(G2) *Elbow geodesic axiom.* Every edge of $M$ is an elbow geodesic in $\mathfrak{S}_{\mathcal{V}}$, and every bounded orthogonal arc in $\mathfrak{S}_{\mathcal{V}}$ is part of an edge of $M$.

(G3) There are no crossing edges in the embedding of $M$ on $\mathfrak{S}_{\mathcal{V}}$.

An orthogonal surface $\mathfrak{S}_{\mathcal{V}} \subset \mathbb{R}^3$ is called *axial* if contains exactly three unbounded orthogonal arcs. Figure 1 shows an axial orthogonal surface. These definitions have been proposed by Miller [14] who, essentially, also observed the following theorem.

**Theorem 3.** *Let $\mathcal{V}$ be axial and $M \hookrightarrow \mathfrak{S}_{\mathcal{V}}$ be a geodesic embedding, then the embedding induces a Schnyder wood of $M^\sigma$, which is suspended at the unbounded orthogonal rays. Conversely, every Schnyder wood of a suspended map $M^\sigma$ induces an axial geodesic embedding of $M^\sigma$.*

An embedding of a Schnyder wood into an orthogonal surface is shown in Figure 1. A proof of the theorem can be found in [9].

Since every orthogonal arc leaving a vertex is occupied by an edge, every angle is completely contained in a *flat*. Basically, flats are the connected regions of constant gray-value in our drawings of orthogonal surfaces. To make this precise, let $H$ be the plane $x_i = h$ and $\tilde{F}_1, \ldots, \tilde{F}_\ell$, the connected components of the interior of $H \cap \mathfrak{S}$. The topological closures $F_1, \ldots, F_\ell$ of these components are $i$-flats of height $h$. The $i$-flat of $v \in V$ is denoted by $F_i(v)$.

Given Theorem 3, it is natural to ask questions about *existence* and *uniqueness* of geodesic embeddings. A surface with three orthogonal arcs meeting in a single

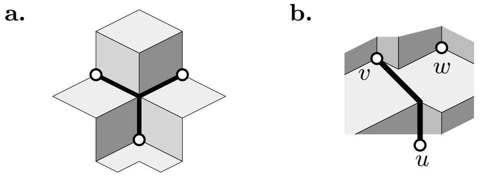


**Fig. 3. a.** A degenerate pattern. **b.** A non-rigid edge $(u, v)$.

point does not support a Schnyder wood, see Figure 3.a. We call surfaces with a such a pattern degenerate. All other orthogonal surfaces support a Schnyder wood. In Figure 3.b. the edge $(u, v)$ can be replaced by the edge $(u, w)$. Hence the surface supports two different graphs and also two different Schnyder woods. The existence of such a choice for an edge is caused by a non-rigidity in the sense of the following definition. An elbow geodesic connecting vertices $u$ and $v$ is *rigid*, if $u$ and $v$ are the only vertices in $\mathcal{V}$ dominated by $u \vee v$. An orthogonal surface $\mathfrak{S}_{\mathcal{V}}$ is a *rigid surface* if all its elbow geodesics are rigid. For an example, see the left part Figure 4, where the inner blue edge is not rigid.

## 3 Rigid Orthogonal Surfaces Via Flat Shifting

We set Theorem 5 into context before we give a new proof. Miller [14] observed that a rigid orthogonal surface supports exactly one Schnyder wood and proved:

**Theorem 4.** *Every suspended 3-connected planar map $M^\sigma$ has a geodesic embedding $M^\sigma \hookrightarrow \mathfrak{S}$ on some rigid orthogonal surface $\mathfrak{S}$.*

Together with the following proposition from [14] (see also [8]) this implies the Brightwell-Trotter Theorem (Theorem 2).

**Proposition 1.** *Let $\mathfrak{S}_{\mathcal{V}}$ be a rigid orthogonal surface. Let $M^\sigma \hookrightarrow \mathfrak{S}_{\mathcal{V}}$ be a geodesic embedding and $F$ a bounded region of $M$. If $\alpha_F$ is the join of the vertices of $F$, then $w \in F \Leftrightarrow w \leq \alpha_F$.*

We give a new proof of the following result by Felsner [8], who answered a question by Miller with this extension of Theorem 4.

**Theorem 5.** *If $S$ is a Schnyder wood of a map $M^\sigma$, then there is a rigid axial orthogonal surface $\mathfrak{S}$ and a geodesic embedding $M^\sigma \hookrightarrow \mathfrak{S}$. In particular $S$ is the unique Schnyder wood supported by $\mathfrak{S}$.*

We will now give a sketch of the proof. Lemmas 6 and 7 are part of this sketch. Let $S$ be a Schnyder wood on a 3-connected planar map $M = (V, E)$ and let $\mathfrak{S}$ be the orthogonal surface obtained from $S$ via face counting. Let $\mathcal{F}_i$ be the set of $i$-flats of $\mathfrak{S}$. On the set $\mathcal{F}_i$ we define a relation $\Gamma_i$ by three rules, Figure 4 shows an example.

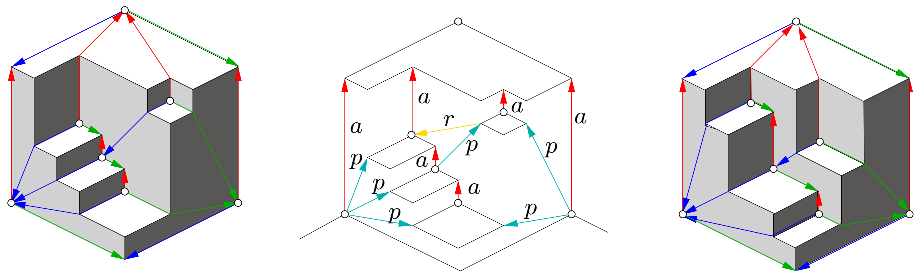


**Fig. 4.** A Schnyder wood $S$ on a non-rigid surface, the corresponding relation $\Gamma_1$, and $S$ on a rigid surface. For $\Gamma_1$ $a$-relations correspond to red arrows, $p$-relations to cyan arrows, and the only $r$-relation to a magenta arrow.

$(a)$ If $(u, v)$ is an edge of color $i$, then $F_i(u) < F_i(v)$ in $\Gamma_i$.

$(p)$ If $(v, u)$ is unidirected in color $i-1$ or $i+1$, then $F_i(u) < F_i(v)$ in $\Gamma_i$.

$(r)$ If $(v, u)$ is unidirected in color $j \neq i$ and there is a vertex $w \in V$ such that $F_j(w) = F_j(u)$ and $w_i > u_i$, then $F_i(v) < F_i(w)$ in $\Gamma_i$.

The pairs in $\Gamma_i$ are classified as $a$-relations (*arc*), $p$-relations (*preserve*) and $r$-relations (*repel*). Lemma 6 is the heart of the proof of Theorem 5 as it justifies why the flat shifts (i.e. $r$-relations) can be combined to obtain a rigid surface.

**Lemma 6.** *The relation $\Gamma_i$ defined on $\mathcal{F}_i$ is acyclic, for $i = 1, 2, 3$.*

*Proof.* By symmetry it is enough to prove the case $i = 1$.

We identify the $a$- and $p$-relations with edges of the Schnyder wood $S$. The set of vertices lying on a common 1-flat is strongly connected in $S$ via bidirected green-blue edges. We define a surjective map from the set of red edges in $S$ to the set of $a$-relations by mapping an edge $(u, v)$ to the relation $F(u) < F(v)$. Similarly, there is a surjective map from the blue and green unidirected edges in $S$ to the $p$-relations (if $(v, u)$ is such an edge, then $F(u) < F(v)$ is in $\Gamma_1$).

In order to deal with the $r$-relations we construct a Schnyder wood $S'$ from $S$ using edge splits (see page 420). Let $e = (v, u) \in S$ be a unidirected blue edge and $F(u) < F(v)$ the corresponding $p$-relation. Let $F(u_k) > \ldots > F(u_1)$ be the set of flats that have an $r$-relation $F(v) < F(u_j)$ related to $e$, the order on this set coming from the $a$-relations. The edges $\{u, u_1\}$ and $\{u_{j-1}, u_j\}$ are bidirected in red and green in $S$. Construct $S'$ by splitting the edges $\{u, u_1\}$, $\{u_1, u_2\}$, ..., $\{u_{k-1}, u_k\}$ towards $v$. This is legal since the angle of $v$ in the face in question has label 2 (green), see Lemma 5.

Repeat this operation for other $r$-relations in $\Gamma_1$ which come from unidirected blue edges. A symmetric operation is used to introduce edges for all $r$-relations in $\Gamma_1$ which come from unidirected green edges in the Schnyder wood $S$.

In the Schnyder wood $S'$ we associate an edge with every relation in $\Gamma_1$. The $a$-relations and $p$-relations are mapped as above while with an $r$-relation

$F(u) < F(v)$ we associate the blue or green edge $(v, u)$ which was introduced into $S'$ by a split.

The idea is to show that a cycle $C$ in $\Gamma_1$ would induce a cycle $C'$ in $T_1 \cup T_2^{-1} \cup T_3^{-1}$ using the inverse of the mapping from edges to relations described above. Here the $T_i$, $i \in \{1, 2, 3\}$, are the respective trees of $S'$ and the existence of $C'$ yields a contradiction to Lemma 1. Note that consecutive relations $F(u) < F(v)$ and $F(u') < F(v')$ in $C$, i.e., $F(v) = F(u')$, may correspond to different vertices $v \neq u'$ from the flat $F(v)$. This does not yield gaps in the intended cycle $C'$ because vertices on the same flat are connected by a path of green-blue bidirected edges. The contradiction shows that $\Gamma_i$ is acyclic. □

Let $\mathfrak{S}$ be the orthogonal surface supporting $S$ which is generated by the face counting vectors (c.f. Theorem 3). Let $\Gamma_i^*$ be the transitive closure of $\Gamma_i$ which is an order on $\mathcal{F}_i$ by Lemma 6. Let $L_i$ be a linear extension of $\Gamma_i^*$. An $i$-flat $F_i$ of $\mathfrak{S}$ is mapped to its position on $L_i$, more formally to $\alpha_{F_i} = |\{F_i' \in \mathcal{F}_i : F_i' < F_i \text{ in } L_i\}|$. With $V$ we associate a set of points $\mathcal{V}_\alpha = \{(\alpha_{F_1(v)}, \alpha_{F_2(v)}, \alpha_{F_3(v)}) \mid v \in V\} \subset \mathbb{R}^3$. We will outline the rest of the proof of Theorem 5, the first step is to prove the following lemma.

**Lemma 7.** *If $R_i(u) = R_i(v)$, then $u_i' = v_i'$ and if $R_i(u) \subset R_i(v)$, then $u_i' < v_i'$.*

Lemma 7 is the key to proving the following four statements, which complete the proof of Theorem 5: $\mathcal{V}_\alpha$ is an antichain in $\mathbb{R}^3$, $\mathfrak{S}_{\mathcal{V}_\alpha}$ is non-degenerate, $\mathfrak{S}_{\mathcal{V}_\alpha}$ supports the Schnyder wood $S$, and $\mathfrak{S}_{\mathcal{V}_\alpha}$ is rigid. This completes the proof sketch for Theorem 5.

Next, we present a simple algorithm which, given a Schnyder wood $S$, computes a rigid orthogonal surface $\mathfrak{S}$ inducing $S$.

**Proposition 2.** *There is an $O(n)$ algorithm computing a rigid orthogonal surface for a given Schnyder wood $S$.*

*Proof.* We assume that $S$ is given in the form of adjacency lists ordered clockwise around each vertex. With each edge in the adjacency list of a vertex $v$, the information about the coloring and orientation of that edge is stored. By symmetry it is sufficient to show how to obtain the first coordinate for all vertices of $S$ in linear time. Produce a copy of the vertex set. On this copy build a digraph $D_r$: For every red edge there is an edge pointing in the same direction in $D_r$ and for all blue and green unidirected edges there is an edge pointing in the opposite direction. Check at each original vertex if its red outgoing edge is green in the reverse direction and if it has a unidirected blue incoming edge. If so, there is an edge from the start of the blue edge to the end of the red outgoing edge. This single repel-edge is sufficient as other repel relations associated to the same unidirected blue edge will be implied by transitivity. Treat repel relations associated to green unidirected edges analogously. Finally, contract all blue-green edges from $S$ in $D_r$. Then, compute a topological sorting of $D_r$ and assign each vertex the topsort-number of its flat as first coordinate. All this can be done in $O(n)$ time. Three runs of this procedure, one for each coordinate are required. The correctness of the algorithm is implied by Theorem 5. □

**Theorem 6.** *Let $P$ be a 3-polytope with $n$ vertices. Then, a Brightwell-Trotter realizer for $P$ can be computed in $O(n)$ time.*

*Proof.* As Fusy et al. [11] show, a Schnyder wood $S$ for the edge graph of $P$ can be computed in $O(n)$ time. Alternatively this can be done with the help of Kant's algorithm [12] as well. From $S$ construct a rigid orthogonal surface $\mathfrak{S}$, in time $O(n)$ using Proposition 2. Then, $\mathfrak{S}$ induces a Brightwell-Trotter realizer of $P$ by Proposition 1. $\square$

## 4 Coplanar Surfaces

An orthogonal surface is called *coplanar*, if there exists a constant $c \in \mathbb{R}$ such that every minimum $v$ on the surface fulfills $v_1 + v_2 + v_3 = c$. Schnyders classic approach of drawing graphs using the face-count vectors $\{(v_1, v_2, v_3) | v \in V\}$ yields a subclass of all coplanar surfaces, see Figure 1. We now generalize the classic approach of counting every bounded face with weight one by allowing more general face weights. We then use coordinate vectors recording the sum of weights in the regions of a vertex. We show that this construction, essentially, yields all coplanar surfaces supporting a given Schnyder wood, and thus all non-degenerate coplanar surfaces can be obtained from some Schnyder wood this way. Geodesic embeddings on coplanar surfaces have the pleasant property that the positions of the vertices in the plane yield a crossing-free and convex straight-line drawing of the underlying graph. Similar approaches for non-coplanar surfaces fail as the drawings need not be crossing-free.

**Theorem 7.** *Let $\mathfrak{S}$ be a coplanar orthogonal surface supporting a Schnyder wood $S$. Then there is a unique weight function $w : F(S) \to \mathbb{R}$ on the set of bounded faces of $S$ and a unique translation $t \in \mathbb{R}^3$ such that for all $v \in V(S)$ and $i \in \{1, 2, 3\}$*

$$v_i = t_i + \sum_{F \in R_i(v)} w(F).$$

**Remark.** A Schnyder wood $S$ and a weight function $w$ define an orthogonal surface $\mathfrak{S}_{S,w}$. This surface, however, need not support the initial Schnyder wood. From the proof of Theorem 3 it follows that a necessary and sufficient condition for an embedding $S \hookrightarrow \mathfrak{S}_{S,w}$ is that

$$R_i(u) \subseteq R_i(v) \quad \Longrightarrow \quad \sum_{F \in R_i(u)} w(F) \leq \sum_{F \in R_i(v)} w(F)$$

with strict inequality whenever $R_i(u) \subset R_i(v)$.

*Proof sketch for Theorem 7.* Let $\mathfrak{S}$ be a coplanar orthogonal surface and $S$ a Schnyder wood induced by $\mathfrak{S}$. Let where $\{i, j, k\} = \{1, 2, 3\}$. We define $t_i = (a_j)_i = (a_k)_i$, which is possible since $F_i(a_j) = F_i(a_k)$ for the suspension vertices $a_1, a_2, a_3$ of $S$. First, shift the surface by $(t_1, t_2, t_3)$, such that the suspension vertices now have coordinates $(c, 0, 0), (0, c, 0), (0, 0, c)$ and $v_1 + v_2 + v_3 = c$ for all $v$.

Let $f$ be the number of faces of $S$. With the region $R_i(v)$ of a vertex $v$ we associate a row vector $r_i(v)$ of length $f-1$ with a component for each bounded face of $F$. The vector $r_i(v)$ is defined by

$$r_i(v)_F = 1 \text{ if } F \in R_i(v) \text{ and } r_i(v)_F = 0 \text{ otherwise.}$$

The existence of a weight assignment to the faces realizing the normalized surface $\mathfrak{S}$ is equivalent to finding a vector $w \in \mathbb{R}^{f-1}$ such that

$$\forall v \in V, \forall i \in \{1, 2, 3\} : \quad r_i(v) \cdot w = v_i \qquad (*)$$

**Claim 1.** The rank of the linear system $(*)$ is at most $f-1$. *Proof omitted.* △

**Claim 2.** Let $e_F$ be the $(f-1)$-dimensional row vector with a single one at the position corresponding to the face $F$. Then, $e_F$ is in the span of the region-face incidence vectors $\{r_i(v) \mid i \in \{1, 2, 3\}, v \in V\}$.

*Proof sketch.* For the proof we distinguish several cases: the boundary of $F$ is a directed cycle, two special edges of the same color are unidirected in opposite directions, two special edges of different colors are unidirected in opposite directions. There are several subcases to be distinguished. We present details for only one case here, the other proofs are similar.

If the boundary of $F$ is not a directed cycle, we may assume that the three special edges $e_1$, $e_2$, $e_3$ have endvertices $v_1$, $w_1$, $v_2$, $w_2$, $v_3$, $w_3$ clockwise in this order on the boundary of $F$ (possibly $w_{i-1} = v_i$). Say $e_1 = (v_1, w_1)$, $e_2 = (w_2, v_2)$, are two unidirected of the same color in opposite directions.

We treat the case that $w_1 = v_2$ and $e_3$ is directed as $(w_3, v_3)$, (this includes the case where $e_3$ is bidirected) explicitly. The left of Figure 5 shows the situation with $i = 1$.

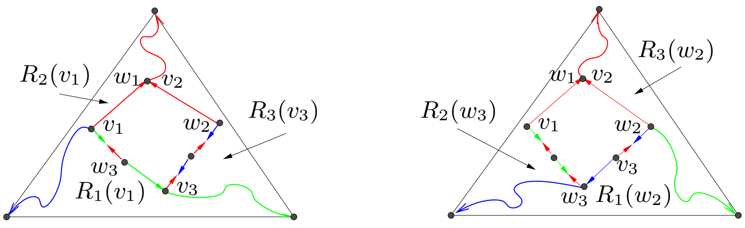


**Fig. 5.** Faces without directed cycle and $w_1 = v_2$

As illustrated in the figure $R_1(v_1)$, $R_2(v_1)$ and $R_3(v_3)$ partition $B \setminus F$, hence

$$\mathbb{1} - (r_i(v_1) + r_{i+1}(v_1) + r_{i-1}(v_3)) = e_F.$$

The case that $e_3$ is directed as $(v_3, w_3)$ is shown in the right part of Figure 5.△

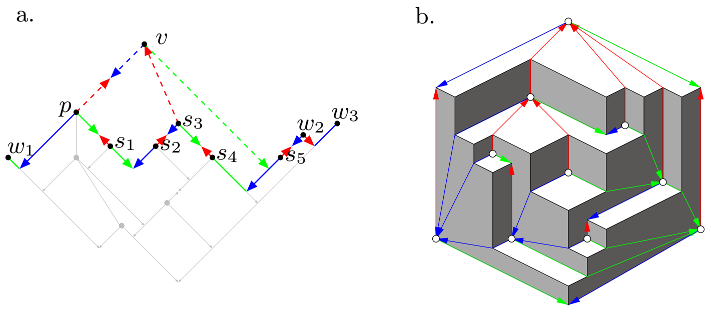


**Fig. 6.** Part a shows the projection of the explored part of an orthogonal surface on the sweep plane. The dashed arrows represent the new edges added with $v$, the other colored arrows are the sweep front. The dotted line segments and vertices are the part of the surface that has already been explored. Part b illustrates Proposition 3, it shows a rigid but not coplanar surface.

Claims 1 and 2 together imply that the linear system $(*)$ has rank $f-1$ and hence a unique solution. □

Next we show how to obtain an efficient algorithm that computes the representation of Theorem 7 for a given orthogonal surface $\mathfrak{S}$.

**Theorem 8.** *Let a non-degenerate, axial, coplanar orthogonal surface $\mathfrak{S}$ be given, which is generated by $n$ minima. A Schnyder wood $S$ for $\mathfrak{S}$ can be computed in $O(n \log n)$ time. Given $S$, the translation vector and the face weights can be computed in $O(n)$ time.*

*Proof.* We first sketch how to extract the Schnyder wood $S$ from $\mathfrak{S}$. The algorithm scans $\mathfrak{S}$ from bottom to top with a sweep plane $P$ orthogonal to the $x_1$-axis. Having seen a subset $W \subset V$ of the generators of $\mathfrak{S}$ the algorithm knows all colored and directed edges of $S$ which are induced by $W$. Figure 6.a shows a snapshot of the intersection of $P$ with $\mathfrak{S}$. The order in which the sweep considers the generators is the lexicographic order on $(x_1, x_2)$. When a minimum $v$ is added, its blue outgoing edge and incoming red edges are added as well. The green outgoing edge is added only if it is not a green-blue bidirected edge, in this case it is added when the next minimum is treated. This procedure builds the Schnyder wood step by step, and needs $O(n \log n)$ time when the sweep front is implemented as a dynamic search tree.

The second part of the algorithm is the computation of the face weights. After normalizing all coordinate vectors the faces are now considered one by one. When considering a face $F$, we first determine the type of $F$. Based on the proof of Theorem 7 we distinguish twenty such types.

The weight of $F$ can be computed from certain region weights of boundary vertices of $F$. The required region weights are the coordinates of these vertices. If for example we are in the case shown in Figure 5 then the weight of $F$ is $c-(w_2)_1-(w_3)_2-(w_2)_3$ where $c$ is the constant obtained through normalization.

When scanning a face $F$ we touch each edge only once and every edge lies in at most two inner faces. This implies a runtime of $O(n)$. □

Face counting produces coplanar surfaces supporting a given Schnyder wood $S$. In Section 3 we have seen how to construct a rigid surface supporting $S$. Coplanarity and rigidity are useful properties for an orthogonal surface. It is natural to ask whether every Schnyder wood has a supporting surface with both properties. The answer to this question is negative, the proof of Proposition 3 can be found in [10].

**Proposition 3.** *The Schnyder wood shown in Figure 6.b cannot be embedded on a rigid and simultaneously coplanar surface.*

## 5 Conclusions

We conclude our investigations of the connections between Schnyder woods and orthogonal surfaces with an open problem of a flavor similar to Theorem 7. Let $S$ be a Schnyder wood induced by an orthogonal surface $\mathfrak{S}$. From Proposition 1 it follows that the bounded faces of $S$ are in bijection with the maxima of $\mathfrak{S}$. We refer to the set of minima and maxima as $\mathcal{V} \cup \mathcal{F}$. For $p \in \mathfrak{S}$ we define its *height* as $h(p) = p_1 + p_2 + p_3$.

**Problem.** Given $\mathfrak{S}$, do the Schnyder Wood $S$ and the vector $h = (h(v))_{v \in \mathcal{V} \cup \mathcal{F}}$ of heights uniquely determine $\mathfrak{S}$?

We can prove this in the case where the underlying graph is a stacked triangulation. With computer's help we have verified that the answer is affirmative for small triangulations with up to twelve vertices.

**Acknowledgements.** Stefan Felsner thanks Nicolas Bonichon for inspiring discussions. Florian Zickfeld was partially supported by the DFG Leibniz grant of Günter M. Ziegler and the Studienstiftung des deutschen Volkes.

## References

1. I. Bárány and G. Rote, *Strictly convex drawings of planar graphs*, 2005. arXiv:cs.CG/0507030.
2. N. Bonichon, S. Felsner, and M. Mosbah, *Convex drawings of 3-connected planar graphs*, in Graph Drawing (Proc. GD '04), vol. 3383 of Lecture Notes in Comput. Sci., 2004, pp. 60–70.
3. G. Brightwell and W. T. Trotter, *The order dimension of convex polytopes*, SIAM J. Discrete Math., 6 (1993), pp. 230–245.
4. G. Brightwell and W. T. Trotter, *The order dimension of planar maps*, SIAM J. Discrete Math., 10 (1997), pp. 515–528.
5. G. Di Battista, R. Tamassia, and L. Vismara, *Output-sensitive reporting of disjoint paths*, Algorithmica, 23 (1999), pp. 302–340.
6. S. Felsner. http://www.math.tu-berlin.de/~felsner/Schnyder.bib.

7. S. Felsner, *Convex drawings of planar graphs and the order dimension of 3-polytopes*, Order, 18 (2001), pp. 19–37.
8. S. Felsner, *Geodesic embeddings and planar graphs*, Order, 20 (2003), pp. 135–150.
9. S. Felsner, *Geometric Graphs and Arrangements*, Vieweg Verlag, 2004.
10. S. Felsner and F. Zickfeld, *Schnyder woods and orthogonal surfaces*, 2006. http://www.math.tu-berlin.de/~felsner/swaos.pdf.
11. E. Fusy, D. Poulalhon, and G.Schaeffer, *Dissection and trees, with applications to optimal mesh encoding and random sampling*, in Proc. 16. ACM-SIAM Sympos. Discrete Algorithms, 2005, pp. 690–699.
12. G. Kant, *Drawing planar graphs using the lmc-ordering*, in Proc. 33rd IEEE Sympos. on Found. of Comp. Sci., 1992, pp. 101–110.
13. C. Lin, H. Lu, and I.-F. Sun, *Improved compact visibility representation of planar graphs via Schnyder's realizer*, SIAM J. Discrete Math., 18 (2004), pp. 19–29.
14. E. Miller, *Planar graphs as minimal resolutions of trivariate monomial ideals*, Documenta Math., 7 (2002), pp. 43–90.
15. W. Schnyder, *Planar graphs and poset dimension*, Order, 5 (1989), pp. 323–343.
16. W. Schnyder, *Embedding planar graphs on the grid*, in Proc. 1st ACM-SIAM Sympos. Discrete Algorithms, 1990, pp. 138–148.
17. W. T. Trotter, *Combinatorics and Partially Ordered Sets: Dimension Theory*, The Johns Hopkins University Press, 1992.

# Partitions of Graphs into Trees

Therese Biedl and Franz J. Brandenburg

[1] School of Computer Science, University of Waterloo, N2L3G1, Canada
biedl@uwaterloo.ca

[2] Lehrstuhl für Informatik, Universität Passau, 94030 Passau, Germany
brandenb@informatik.uni-passau.de

**Abstract.** In this paper, we study the *k-tree partition problem* which is a partition of the set of edges of a graph into $k$ edge-disjoint trees. This problem occurs at several places with applications e.g. in network reliability and graph theory. In graph drawing there is the still unbeaten $(n-2) \times (n-2)$ area planar straight line drawing of maximal planar graphs using Schnyder's realizers [15], which are a 3-tree partition of the inner edges. Maximal planar bipartite graphs have a 2-tree partition, as shown by Ringel [14]. Here we give a different proof of this result with a linear time algorithm. The algorithm makes use of a new ordering which is of interest of its own. Then we establish the NP-hardness of the $k$-tree partition problem for general graphs and $k \geq 2$. This parallels NP-hard partition problems for the vertices [3], but it contrasts the efficient computation of partitions into forests (also known as arboricity) by matroid techniques [7].

## 1 Introduction

A *k-tree partition* of a graph $G = (V, E)$ is the partition of the set of edges $E$ into $k$ disjoint subsets which each induce a tree. Alternatively, the edges of $G$ are colored by $k$ colors and each color induces a tree. The trees are not necessarily spanning trees. The *k-tree partition problem* for a graph $G$ and an integer $k$ is whether or not $G$ has a $k$-tree partition.

A relaxed version without connectivity is the *arboricity* $a(G)$ of a graph $G$, which is a partition of the edges of $G$ into at most $a(G)$ forests. A well-known theorem by Nash-Williams states that a graph has arboricity $c$ if and only if every non-trivial subgraph $H$ has at most $c(|V(H)|-1)$ edges [11,12]. In particular, this implies that every planar graph has arboricity at most 3, and every planar bipartite graph has arboricity 2. In fact, the two forests of the arboricity-decomposition must be “almost” trees: either one is a spanning tree and the other has $n-3$ edges, or both have $n-2$ edges. Ringel [14] proved that in fact, any maximal planar bipartite graph can be split into two trees, both with $n-2$ edges.

In this paper, we study algorithmic aspects of splitting a maximal planar bipartite graph into two trees. The proof by Ringel [14] is algorithmic in nature, but not particularly fast; it can be implemented in quadratic time. We give a different proof to show that every maximal planar bipartite graph can be split into two trees; the resulting algorithm is quite simple and can easily be implemented in linear time. As a side-effect, we develop a special vertex ordering for maximal planar bipartite graphs, which may be of interest of its own.

For general graphs with $n$ vertices and $m$ edges the arboricity can efficiently be computed by matroid techniques [7]. Here the relaxation to forests is crucial. We show the NP-hardness of the $k$-tree partition problem for every $k \geq 2$. For $k = 2$ this is proved by a reduction from the Not-All-Equal-3SAT problem, and for $k \geq 3$ there is a reduction from the $k$-coloring problem. These NP-hardness results complement common and extended versions of partition and coloring problems [2,3,8], which however are defined for the vertices.

Partitioning graphs into trees is also used in graph drawing. For example, the Schnyder realizers [15] are a partition of the inner edges of a maximal planar graph into three trees. They still yield the best known area bounds for straight-line grid drawings of maximal planar graphs.

In this paper, first we study maximal planar bipartite graphs and how they split into two trees, and then we show the NP-hardness results for the general case.

## 2 Maximal Planar Bipartite Graphs

Let $G = (V, E)$ be a maximal planar bipartite (mpb) graph. Thus, $G$ has a vertex partition $V = W \cup B$ into *white vertices* $W$ and *black vertices* $B$ such that each edge connects a white vertex with a black vertex. Furthermore, $G$ can be drawn in the plane without crossings such that every face has exactly four incident edges. It is well-known that $G$ has $2n - 4$ edges (where $n = |V|$) and is bi-connected (i. e., cannot be disconnected by removing one vertex.)

### 2.1 A Vertex Ordering for mpb Graphs

In this section, we present a vertex ordering for maximal planar bipartite graphs, which we will then use in the next section to obtain a split of an mpb graph into two trees.

**Theorem 1.** *Let $G$ be a maximal planar bipartite graph with a fixed planar embedding and a fixed outer-face. Then there exists a vertex ordering $v_1, \ldots, v_n$ of $G$ such that*

- *$v_1$ and $v_n$ are the two black vertices on the outer-face.*
- *For all $i > 1$, vertex $v_i$ is on the outer-face of the graph induced by $v_1, \ldots, v_i$.*
- *Every white vertex $v_i$ has exactly one predecessor, i. e., neighbor with a smaller number.*
- *Every black vertex $v_i$, $i > 1$ has at least two predecessors.*

We will call such a vertex ordering an *mpb-ordering*. See Figure 1. To prove Theorem 1, we need an auxiliary graph. Let $E_B$ be the *black diagonals*, i. e., for every face $f$ in $G$ (which has exactly two black vertices since $G$ is maximal

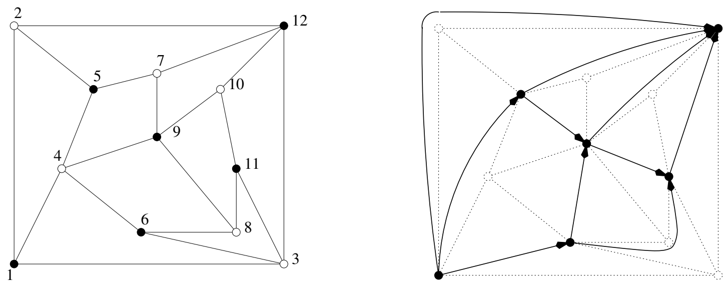


**Fig. 1.** An mpb-ordering of $G$, and the graph $G_B$ (solid) with a bipolar orientation

planar bipartite), add an edge between the two black vertices on $f$ to $E_B$. Let $G_B$ be $(B, E_B)$, i.e., take the black vertices and black diagonals only. See also Figure 1.

One can show that $G_B$ is bi-connected [4]. Therefore, we can compute an *st-numbering* of $G_B$, i.e., an ordering $b_1, \ldots, b_l$ of the black vertices such that for any $1 < j < l$, vertex $b_j$ has at least one predecessor and at least one successor, i.e., neighbor with a larger index [10]. Moreover, we can choose which vertices should be $b_1$ and $b_l$, and we can compute this order in linear time [6]. We choose here $b_1$ and $b_l$ to be the two black vertices on the outer-face of $G$.

From this *st*-numbering, we can obtain a *bipolar orientation*, i.e., an acyclic orientation of the edges of $G_B$ such that there is only one *source* (vertex without incoming edge) and only one *sink* (vertex without outgoing edge), simply by directing every edge from the lower-indexed to the higher-indexed vertex.

Let $G^+ = (V, E \cup E_B)$ be the graph resulting from $G$ by adding the black diagonals. We now extend the bipolar orientation of $G_B$ into one of $G^+$ as follows. For every white vertex $w$, let $b_i$ be the neighbor of $w$ (in $G$) that has the *smallest* index among the neighbors of $w$. Orient the edge $(b_i, w)$ from $b_i$ to $w$, and all other edges incident to $w$ away from $w$. Clearly this orientation is bipolar: every white vertex must have degree at least 2 (by maximality), and has exactly one incoming edge by definition, and hence at least one outgoing edge. Furthermore, the orientation is acyclic since any directed path encounters increasingly larger indices in its black vertices.

From this bipolar orientation, we can recover a vertex ordering of all vertices of $G$, simply by computing a topological order in the acyclic graph. Let $v_1, \ldots, v_n$ be the resulting order; one can easily verify that it satisfies all conditions of Theorem 1.

Note that all steps of computing the mpb-ordering can easily be implemented in linear time.

The mbp-ordering is not a canonical ordering [5,9]: The black vertices act similar to the vertices of a canonical ordering, but white vertices have only one predecessor and violate the 2-connectivity property of a canonical ordering. E.g. the mbp-ordering of the graph in Figure 1 is not a canonical ordering.

## 2.2 Splitting into Two Trees

Now assume that we are given an mpb-ordering $v_1, \ldots, v_n$. We now show how to obtain a decomposition into two trees from it. We have two simple rules (see also Figure 2):

- For any white vertex, label the (unique) incoming edge with 1.
- For any black vertex $v_i \neq v_1$, label the leftmost incoming edge with 1 and the rightmost incoming edge with 2. Here, "leftmost" and "rightmost" are taken with respect to the planar embedding; recall that $v_i$ is in the outer-face of the graph induced by $v_1, \ldots, v_{i-1}$, hence we can sort its incoming edges by the order in which these neighbors appear on the outer-face.
  All other (if any) incoming edges of $v_i$ are called "middle incoming" and labeled with 2 as well. However, we will reverse the orientation of these edges, to make it easier to argue why the resulting structures must be trees.

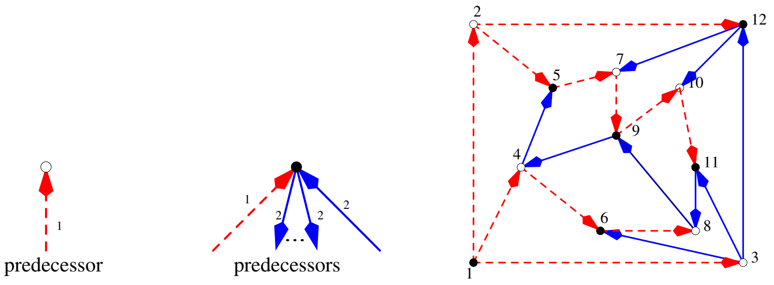


**Fig. 2.** Splitting the graph into two trees

From now on, let the 1-*edges* be the edges labeled 1, and the 2-*edges* be the edges labeled 2. We also use *1-path*, *1-cycle* and so on to mean a path/cycle of 1-edges. It is very easy to see that the 1-edges form a tree.

**Lemma 1.** *The* 1-*edges form a spanning tree.*

*Proof.* Since every vertex except $v_1$ has exactly one incoming 1-edge, there are $n-1$ 1-edges. $v_1$ has outgoing 1-edges, so the 1-edges span all vertices of the graph. No 1-edge had its orientation reversed, so the 1-edges form a directed acyclic graph. It is well-known that such a graph is a spanning tree.

Now we come to the significantly harder part of proving that the 2-edges form a tree. We first need observations about the order of edges around each vertex; see also Figure 3.

*Claim.* $v_1$ has only outgoing 1-edges. For any other black vertex, the incident edges are clockwise in the planar embedding as follows:

- One incoming 1-edge.
- Some number (possibly none) of outgoing 1-edges.

– One incoming 2-edge.
– Some number (possibly none) of outgoing 2-edges.

*Proof.* This follows directly from the way labels and directions were assigned to vertices, plus the fact that every successor of a black vertex is a white vertex and hence contributes an outgoing 1-edge.

*Claim.* Let $w_1, w_2$ be the white vertices on the outer-face. For every white vertex, the incident edges are ordered clockwise in the planar embedding as follows:

– One incoming 1-edge.
– Some number (possibly none) of outgoing 2-edges.
– One incoming 2-edge (except for $w_1$ and $w_2$).
– Some number (possibly none) of outgoing 1-edges.

*Proof.* Clearly each white vertex $w$ has an incoming 1-edge. Now consider any successor $b$ of $w$, which is a black vertex. The label of the edge $(w, b)$ depends on whether $w$ is a left, middle or right predecessor of $b$. If $w$ is a right (left) predecessor, then $(w, b)$ is labeled 2 (1), and its orientation is maintained. If $w$ is a middle predecessor of $b$, then $(w, b)$ is labeled 2 and turned around. Clearly the clockwise order of edges around $w$ corresponds to whether $w$ is a right, middle, left predecessor, so all that remains to argue is that if $w \neq w_1, w_2$, then it indeed must be the middle predecessor exactly once.

Consider the moment when we add the vertex $b$ that makes $w$ disappear from the outer-face. Then $b$ must be black (white vertices have indegree 1), and adjacent to $w$ by maximality, so $w$ is a middle predecessor of $b$. It cannot be middle predecessor of anyone else, since it can disappear from the outer-face only once.

**Lemma 2.** *The* 2*-edges form a tree.*

*Proof.* Let $v_1, w_1, v_n, w_2$ be the outer-face in clockwise order. By the claims, every vertex except $v_1, w_1, w_2$ has exactly one incoming 2-edge, so there are $n - 3$ 2-edges. $w_2$ has an outgoing 2-edge (to vertex $v_n$), so the graph spanned by 2-edges has $n - 3$ edges and $n - 2$ vertices. To show that this graph is a tree it therefore suffices to show that it has no cycles.

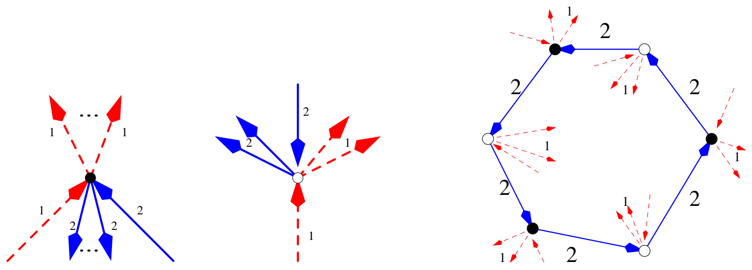


**Fig. 3.** Labeled edges around each vertex, and why no directed 2-cycle can exist

Assume we had a cycle of 2-edges $C$. Since every vertex has at most one incoming 2-edge, such a cycle must necessarily be directed. Assume that the cycle is directed counter-clockwise in the fixed planar embedding; the case of a clockwise directed cycle is very similar. By the first Claim, no black vertex on $C$ can have 1-edges between the incoming 2-edge and the outgoing 2-edge of $C$. Hence, a black vertex on $C$ has no incident 1-edges on the inside of $C$. Similarly by the second Claim a white vertex has no incident 1-edges on the outside of the cycle. See also Figure 3.

We know that the 1-edges form a rooted tree, and its root $v_1$ has no incident 2-edges and hence is not part of $C$. Now where is $v_1$ located? Assume that it is outside cycle $C$. Let $w$ be a white vertex on $C$, and let $b$ be its predecessor on the (unique) 1-path from $v_1$ to $w$. Vertex $b$ is inside $C$ by the above, hence there exists a directed 1-path from the outside of $C$ to the inside of $C$. However, the order of edges around each vertex of $C$ makes this impossible: any directed 1-path can reach $C$ from the outside only at a black vertex, and is immediately directed back to the outside from there. Hence $v_1$ must be inside $C$. But now we can repeat the argument with a black vertex $b$ on $C$; no directed 1-path can go from inside $C$ to the neighbor of the incoming 1-edge of $b$ (which is outside $C$). So we obtain a contradiction and no directed 2-cycle can exist.

**Theorem 2.** *Every maximal planar bipartite graph has a* 2*-tree partition. Furthermore, such a partition can be found in linear time.*

Not only gave we a split into two trees, we also obtained that the edges around each vertex are ordered in a special way when considering the incoming/outgoing edges of each tree. Note that this is similar to the edge-orderings obtained when splitting a triangulated planar graph into three trees [15]. An $\lfloor n/2 \rfloor \times \lceil n/2 - 1 \rceil$ grid drawing of planar bipartite graphs has been obtained in [1] using different techniques.

## 3 Tree Partitions of General Graphs

In this section we address the complexity of the tree partition problem. Recall that a $k$-tree partition of a graph is equivalent with a $k$-edge coloring such that each color induces a tree. Our problem is complementary to common and generalized partition and coloring problems of graphs which however address the vertices. Such problems have been studied in many versions, see, e.g., [2,3,8].

Clearly, a graph has a 1-tree partition if and only if it is a tree. This can be checked easily. All other cases are NP-hard. It turns out that connectivity is the crucial factor for the NP-hardness, since the partition into forests can be solved in polynomial time.

**Theorem 3.** *It is NP-hard to test whether a graph $G$ has a* 2*-tree partition.*

*Proof.* We reduce from the Not-All-Equal-3SAT problem [8]. The construction extends the reduction to the 3-coloring problem in [13].

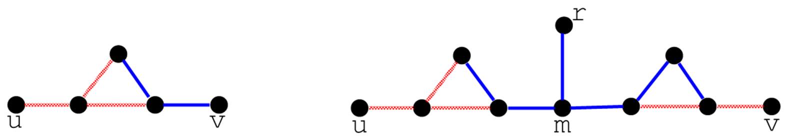


**Fig. 4.** Switch and Double-Switch

Let $\alpha = c_1, \ldots, c_m$ be an expression with clauses $c_1, \ldots, c_m$ and variables $x_1, \ldots, x_n$ and such that there is a clause $(x, x, \bar{x})$ for every variable $x$. The assignment must be such that there is a true and a false literal in each clause. Let $m > 1$. Construct a graph $G(\alpha)$ which has a 2-tree partition into a blue and a red tree if and only if $\alpha$ has a Not-All-Equal assignment. $G(\alpha)$ has a distinguished root $r$. The reduction is based on two gadgets, a *switch* and a *double-switch* between two vertices $u$ and $v$, as shown in Figure 4. In a double-switch there is a direct edge to the root. In both gadgets, only $u$ and $v$ (and $r$ for the double-switches) may have edges incident to other vertices.

We immediately observe:

*Claim.* In a 2-tree partition,

– the edges of a triangle cannot belong to a single tree and must be colored with different colors,
– the edges at the endpoints $u$ and $v$ of a switch are colored by different colors, and
– the edges at the endpoints $u$ and $v$ of a double-switch are colored by the same color, which is different from the color of the edges at $m$.

We can now give the reduction.

For every variable $x$ construct a switch $S(x)$ with the vertices $x$ and $\bar{x}$, and directly connect these vertices to the root with an edge. For every clause $c_i$ construct a triangle with vertices $c_i(1), c_i(2), c_i(3)$ which are identified with the literals. Connect each variable $x$ in $S(x)$ with its occurrences in the clauses by paths of length two. Each such path $p_j$ has a middle vertex $m_j$ and two edges $e_j = (x, m_j)$ and $f_j = (m_j, x_j)$, where $x_j$ is the $j$-th occurrence of $x$ in some clause. Finally, connect any two such vertices $m_i$ and $m_j$ by a double-switch.

Hence each variable $x$ has a gadget $H(x)$ consisting of the switch $S(x)$ and the triangle for the clause $(x, x, \bar{x})$ with the vertices $x'$, $x''$ and $\bar{x}'$. There are paths between $x$ and $x'$, $x$ and $x''$ and $\bar{x}$ and $\bar{x}'$, and there is a double-switch between the first two paths, see Figure 5.

*Claim.* $\alpha$ has a Not-All-Equal truth assignment if and only if $G(\alpha)$ has a 2-tree partition.

*Proof.* First, suppose there is a Not-All-Equal truth assignment. Then color the edges from the root to each vertex representing a true literal blue, and the edges towards the false literals red, and for the edges between the root and the double-switches take the opposite color of the edges at the endpoints. Complete the

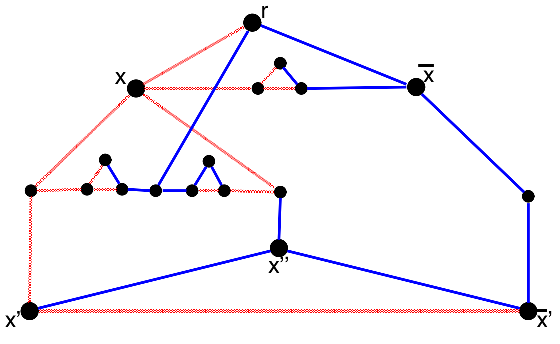


**Fig. 5.** The gadget of a variable x

coloring of the remaining edges of the switches and the double-switches. They separate edges with the same color at the endpoints. Finally, color the edges in the triangles of the clauses such that there are no monochrome cycles. The latter is doable, since there is a Not-All-Equal assignment with a true and a false literal for each clause. Thus there is a blue and a red path from the root to each triangle. Now all edges are properly colored, and each color induces a tree.

Conversely, if there is a 2-tree partition, then there is a Not-All-Equal assignment of the variables by the color of the edges on the monochrome paths in the gadgets $H(x)$, and there is a recoloring of the edges such that blue edges correspond to true and red edges to false.

*Claim.* For every gadget $H(x)$ the path from $\bar{x}$ to $\bar{x}'$ is monochrome, say blue, and there is blue path between $\bar{x}$ and the root. There is a monochrome red path between $x$ and $x'$ or $x$ and $x''$ and a red path between $x$ and the root.

*Proof.* By the first claim both colors are present in the triangles for the clause. Suppose the path between $\bar{x}$ and $\bar{x}'$ is not monochrome. Then it is a dead end and does not connect the edges in the triangle of the clause to the root. Now both colors in the triangle for the clause must come through $x$. Then there must be a blue path and a red path from $x$ to $x'$ and to $x''$. This is impossible by the double-switch, whose ends have the same color. Thus the path from $\bar{x}$ to $\bar{x}'$ is monochrome, and there must be a monochrome path of the other color to $x'$ or $x''$ through $x$. All red (blue) edges must be connected by monochrome paths, and this connection can only be established through to root.

For every variable we color the vertex $x$ in $S(x)$ by the color of the monochrome path to $x'$ or $x''$ and accordingly for $\bar{x}$, and we assign $x$ the value true if $x$ is blue, and false if $x$ is red. By the previous claim this is consistent for a pair $x, \bar{x}$.

Finally, to achieve a coloring which agrees with an assignment we may recolor some edges.

First, for every $x$ all paths between $x$ in $S(x)$ and $x$ in the triangles of the clauses can be colored with the color of $x$.

Therefore, observe that by the double-switches two such paths cannot be monochrome and with different colors. Suppose that $x$ is blue and let $p = (e, f)$ be a path from $x$ with different colors for $e$ and $f$. Then $e$ cannot be red by the connectivity of the tree. If $f$ is red, then it is a dead end. The two vertices of the triangle which are not incident with $f$ are attached to a blue and a red edge, and these edges have monochromatic paths to the root. Now the edge $f$ can be recolored blue. Subsequently, the edge in the triangle between the two vertices with an attached blue edge must be colored red, and one of the other edges in the triangle must be red.

Finally, suppose $x$ is blue and the edge from $x$ to the root is red. Then recolor this edge blue. This may induce blue cycles from the root to $x$, via a blue path to a triangle and back to the root. We break each such cycle in the triangle by recoloring the edge between the two attached blue edges and let another edge in the triangle be red. This is consistent with the 2-tree partition.

Now all edges incident with a vertex $x$ of a literal are single-colored, and every clause has a red and a blue edge. Hence, there is a consistent Not-All-Equal truth assignment.

For $k \geq 3$ we reduce from the $k$-coloring problem. Let $G$ be an instance of $k$-coloring. Add a new root $r$ to $G$ and connect all vertices of $G$ with the root. Replace each edge $(u, v)$ of $G$ by a switch as shown in Figure 4. Then the coloring of $G$ one-to-one corresponds to the tree partition of the constructed graph. In any $k$-tree partition of the resulting graph if the edge from the root to a vertex $v$ is in the $i$-th tree, then $v$ is assigned the $i$-th color. By the switches, adjacent vertices are colored differently. Conversely, color the edges from a vertex $v$ to the root according to the given color of $v$, color the remaining edges in the switches appropriately.

We summarize:

**Theorem 4.** *For every $k \geq 2$ is NP-hard whether a graph $G$ has a $k$-tree partition.*

## Acknowledgements

We thank an anonymous referee for the valuable suggestions.

## References

1. T. Biedl and F. J. Brandenburg. Drawing planar bipartite graphs with small area. In *Proc. 17th Canadian Conference on Computational Geometry, CCCG'05, University of Windsor, Ontario, Canada, August 10-1, 2005*, volume 17, 2005.
2. A. Brandstädt, V. B. Le, and T. Symczak. The complexity of some problems related to graph 3-colorability. *Disc. Appl. Math.*, 89(1):59–73, 1998.
3. H. Broersma, F. V. Fomin, J. Kratochvil, and G. J. Woeginger. Planar graph coloring avoiding monochromatic subgraphs: Trees and paths make it difficult. *Algorithmica*, 44:343–361, 2006.

4. H. de Fraysseix, P. O. de Mendez, and J. Pach. Representation of planar graphs by segments. *Intuitive Geometry*, 63:109–117, 1991.
5. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
6. S. Even and R. E. Tarjan. Computing an *st*-numbering. *Theoretical Computer Science*, 2:436–441, 1976.
7. H. N. Gabow and H. H. Westermann. Forests, frames, and games: Algorithms for matroid sums and applications. *Algorithmica*, 7:465–497, 1992.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco, 1979.
9. G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996.
10. A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs, International Symposium Rome 1966*, pages 215–232. Gordon and Breach, 1967.
11. C. S. J. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *J. London Math. Soc.*, 36:445–450, 1961.
12. C. S. J. Nash-Williams. Decomposition of finite graphs into forests. *J. London Math. Soc.*, 39:12, 1964.
13. C. Papadimitriou. *Computational Complexity.* Addison Wesley, Reading MA, 1994.
14. G. Ringel. Two trees in maximal planar bipartite graphs. *J. Graph Theory*, 17: 755–758, 1993.
15. W. Schnyder. Embedding planar graphs on the grid. In *1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 138–148, 1990.

# The Website for Graph Visualization Software References (GVSR)

Bruno Pinaud, Pascale Kuntz, and Fabien Picarougne

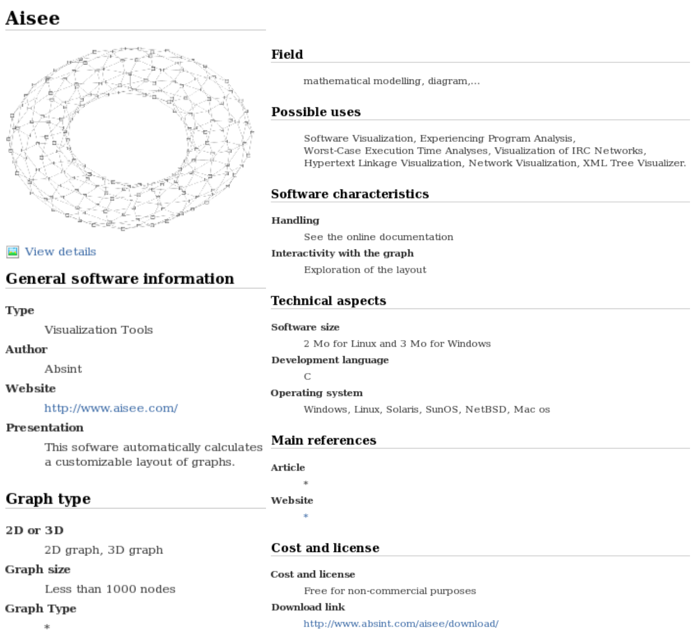
Laboratoire d'informatique de Nantes Atlantique - France
bruno.pinaud@univ-nantes.fr

**Abstract.** Graph drawing software are now commonly used. However, the choice of a well-adapted program may be hard for an inexperienced user. This poster presents a website (http://www.polytech.univ-nantes.fr/GVSR/) built to help users choose a program adapted to their problems. So far, this site uniformely presents fifty programs and aims at helping users both in their choices and in comparing the programs.

The current profusion of available graph drawing programs lets the non-specialist user often confused. Some programs have been developed in close partnership with the scientific community (*e.g.* Pajek for social networks). But generally speaking, the choice of a program well-adapted to both the data and the methodology remains difficult for a user expert in his field but not in graph drawing. Recent books can be used as guides [1,2], and several websites present lists of general or specialized programs [3,4,5] or synthetic views of different types of layouts [6]. However, either the information is too complex for a non-specialist or it is presented incompletely or in a heterogeneous form uneasy to explore. Consequently much effort is required to compare the various programs.

Those restrictions led us to develop a website called "*Graph Visualization Software References*" (http://www.polytech.univ-nantes.fr/GVSR/). Built as a directory, it presents the programs with a uniform text-based description along with a screenshot. This site keeps evolving and so far contains about fifty various software descriptions classified into five types: libraries, visualization tools, knowledge representation, 3D only tools, and specific tools. Our objectives are to facilitate the users' choices and to compare programs with common criteria.

Each program has its own description card (Fig. 1) made of a screenshot, general information (*e.g.* author(s)' name, website, ...), specific information on the visualization (*e.g.* possible uses, graph type(s), ...), technical information (*e.g.* operating system(s), license(s), ...) and references (*e.g.* publication(s), website(s)). Each card is described by an XML file. The files are managed with the native XML database "*Exist*". Some of the existing functionalities are an automatic indexing of the data and an organization of the XML files in collections like in a computer hard-disk. The communication with the user's web browser is done with the *Apache Tomcat* servlet container *via* "*JavaServer Pages* (JSP)". This technology allows to easily create websites with a dynamic content independent from the server and client architectures. In addition, the site allows users to propose new programs by simply completing an enclosed form. Finally, the site has an XQUERY-based search engine.



**Aisee**

View details

**General software information**

**Type**
Visualization Tools
**Author**
Absint
**Website**
http://www.aisee.com/
**Presentation**
This sofware automatically calculates a customizable layout of graphs.

**Graph type**

**2D or 3D**
2D graph, 3D graph
**Graph size**
Less than 1000 nodes
**Graph Type**
*

**Field**

mathematical modelling, diagram,...

**Possible uses**

Software Visualization, Experiencing Program Analysis, Worst-Case Execution Time Analyses, Visualization of IRC Networks, Hypertext Linkage Visualization, Network Visualization, XML Tree Visualizer.

**Software characteristics**

**Handling**
See the online documentation
**Interactivity with the graph**
Exploration of the layout

**Technical aspects**

**Software size**
2 Mo for Linux and 3 Mo for Windows
**Development language**
C
**Operating system**
Windows, Linux, Solaris, SunOS, NetBSD, Mac os

**Main references**

**Article**
*
**Website**
*

**Cost and license**

**Cost and license**
Free for non-commercial purposes
**Download link**
http://www.absint.com/aisee/download/

**Fig. 1.** Example of a software description card

The content of the site keeps evolving by the addition of new programs and new functionalities. We are currently working on a benchmark for comparing different programs on the same graph base. However, as the graph description format is different for each program, we are working on a format converter *via* the Graph Exchange Language (GXL) [7].

## References

1. Kaufmann, M., Wagner, D., eds.: Drawing Graphs: Methods and Models. Volume 2025 of LNCS. Springer (2001)
2. Mutzel, P., Jünger, M.: Graph Drawing Software. Springer Verlag (2003)
3. `http://rw4.cs.uni-sb.de/users/sander/html/gstools.html`.
4. `http://directory.google.com/Top/Science/Math/Combinatorics/Software/Graph_Drawing`.
5. `http://www.dia.uniroma3.it/research/ACG.html`.
6. `http://www.visualcomplexity.com/vc/index.cfm`.
7. Winter, A.: Exchanging graphs with GXL. In Mutzel, P., Jünger, M., Leipert, S., eds.: Graph Drawing: 9$^{th}$ Int. Symp. (GD'01). Volume 2265 of LCNS. Springer (2002) 485–500

# Smoother Transitions Between Breadth-First-Spanning-Tree-Based Drawings

Christopher Homan[1], Andrew Pavlo[2], and Jonathan Schull[1]

[1] Rochester Institute of Technology
[2] University of Wisconsin-Madison

We demonstrate a collection of techniques that seek to make the transition between drawings based on two topologically distinct spanning trees of the same graph as clear as possible.

As Herman, Melançon, and Marshall note [HMM00], one way to draw a large graph is to extract a spanning tree from it, use a tree layout algorithm [CK95, Ead92, RT81, II90, TM02, GADM04, LY05] to draw the spanning tree, and then add back the graph edges not included in the spanning tree. The problem with this approach is that the drawings tend to favor the edges that are part of the spanning tree, even though they may be no more important in the underlying structure than non-spanning tree edges. One way of dealing with this problem is to facilitate exploration of multiple spanning trees.

Yee et al. [YFDH01] describe a system that produces layouts based on Eades' radial layout algorithm [Ead92] and lets users interactively select a new node as root. When this happens, the system first calculates a breadth-first spanning tree rooted at the selected node, and then smoothly transitions to a topologically distinct spanning tree. Although Yee et al.'s static layouts are free of edge crossings, transitions between trees can be hard to follow because there edge crossings do occur.

A number of tree-based graph visualizations, such as RINGS [TM02], RDT [JP98], and others [LRP95, Mun97, Wil99] also allow users to reconfigure views of a given tree, and some even allow users to change the root node. They do not, however, let the user select and smoothly transition to a different spanning tree built from a different collection of edges. To our knowledge only Yee et al.'s system [YFDH01] and one mentioned by Melançon and Herman [MH98] support smooth transitions between different spanning trees of the same graph.

We have built a system to use as a test bed for improving the sort of transitions between topologically distinct graphs that Yee et al. and Melançon and Herman use. Here we present some preliminary results. Like Yee et al. we only use breadth-first search trees. These often share common subtrees, especially when the roots of the trees are closely related. A major thrust of our research concerns layouts that make it easier for users to perceive the migration of these common subtrees as they disconnect from their old parents, and then reconnect at their new parents' locations.

As illustrated in our poster, our static layouts, use a variant of what Lin and Yen call "a balloon drawing subtree with non-uniform size" [LY05] (flattened-out cone drawings [CK95, JP98]) rather than the radial layout of Eades [Ead92] and Yee et al. Our variant balloon drawing method places children not around the entire balloon, but rather on the centrifugal semi-circle of the balloon that lies outside the parent's balloon. This allows us to guarantee that the distance from a nonroot node to the root monotonically increases with the depth of the node, even though balloon layouts forego the stronger invariant that all nodes of a given depth are equidistant from the parent. It also allows us to replace node and link diagrams with solid-looking structures with highly idiosyncratic silhouettes, and with sub-structures that can be gracefully detached and moved, to striking visual effect, from the parent in an old spanning tree to the new parent in the new spanning tree.

Thus, rather than adopt the classical balloon tree convention of drawing each node as a point lying on the perimeter of its balloon, we draw it as a hemisphere covering the node's balloon. Building on Biederman's [Bie87] and Irani and Ware's [IW03] research on the human visual system's pre-attentive capacity to construct shape representations from shading and silhouette, we also shade the hemispheres to emphasize the idiosyncratic form of each structure. (Empirical work will be required to determine whether a visualization this strange is useful, but the work of Irani and Ware suggests that such idiosyncratically shaped three-dimensional forms enhance memory and perception, especially by novices, of underlying graph relationships.)

Our animation algorithm is, to our knowledge, novel. Given a graph drawn according to a breadth-first spanning tree (hereafter known as the "old drawing") and a node chosen to be the new root, the algorithm:

1. Calculates a breadth-first spanning tree rooted at the chosen node.
2. Calculates a new drawing based on the new spanning tree. Stores the angle and distance from each nonroot node to its new parent in the new layout (in effect, each such node's parent becomes the origin of the coordinate system that holds the node's position) we call this distance and angle the *new relative coordinates* of the node.
3. Calculates for each nonroot node the angle and distance in the *old drawing* from its *new* parent. We call these the *old relative coordinates.*

Then the algorithm generates each frame of the animation by interpolating between the old and new relative coordinates and calculating the absolute position of each node by recursively calculating the absolute position of its parent. More details are in [PHS06].

## References

[Bie87] Irving Bierderman. Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94(2):115–147, 1987.

[CK95] J. Carrire and R. Kazman. Research report: Interacting with huge hierarchies: Beyond cone trees. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 74–81, 1995.

[Ead92] Peter Eades. Drawing free trees. *Bulletin of the Institute of Combinatorics and its Applications*, 5:10–36, 1992.

[GADM04] S. Grivet, D. Auber, J.P. Domenger, and G. Melançon. Bubble tree drawing algorithm. In *International Conference on Computer Vision and Graphics*, pages 633–641, 2004.

[HMM00] Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[II90] J. Q. Walker II. A node-positioning algorithm for general trees. *Software: Practice and Experience*, 20(7):685–705, 1990.

[IW03] Pourang Irani and Colin Ware. Diagramming information structures using 3d perceptual primitives. *ACM Transactions on Computer-Human Interaction*, 10(1):1–19, 2003.

[JP98] Chang-Sung Jeong and Alex Pang. Reconfigurable disc trees for visualizing large hierarchical information space. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 19–25, 1998.

[LRP95] John Lamping, Ramana Rao, and Peter Pirolli. The hyperbolic browser: A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[LY05] Chun-Cheng Lin and Hsu-Chun Yen. On balloon drawings of rooted trees. In *GD '05: Revised Papers from the 13th International Symposium on Graph Drawing*, pages 285–296, London, UK, 2005. Springer-Verlag.

[MH98] Guy Melançon and Ivan Herman. Circular drawings of rooted trees. Technical Report INS-R9817, CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, 1998.

[Mun97] Tamara Munzner. H3: Laying out large directed graphs in 3d hyperbolic space. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 2–10, 1997.

[PHS06] Andrew Pavlo, Christopher M. Homan, and Jonathan Schull. A parent-centered radial layout algorithm for interactive graph visualization and animation. Technical Report `http://arxiv.org/abs/cs/0606007`, arXiv.org, 2006.

[RT81] Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, 7(2):223–228, 1981.

[TM02] Soon Tee Teoh and Kwan-Liu Ma. Rings: A technique for visualizing large hierarchies. In *GD '02: Revised Papers from the 10th International Symposium on Graph Drawing*, pages 268–275, London, UK, 2002. Springer-Verlag.

[Wil99] Graham J. Wills. NicheWorks — interactive visualization of very large graphs. *Journal of Computational and Graphical Statistics*, 8(2):190–212, 1999.

[YFDH01] Ka-Ping Yee, Danyel Fisher, Rachna Dhamija, and Marti A. Hearst. Animated exploration of dynamic graphs with radial layout. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 43–50, 2001.

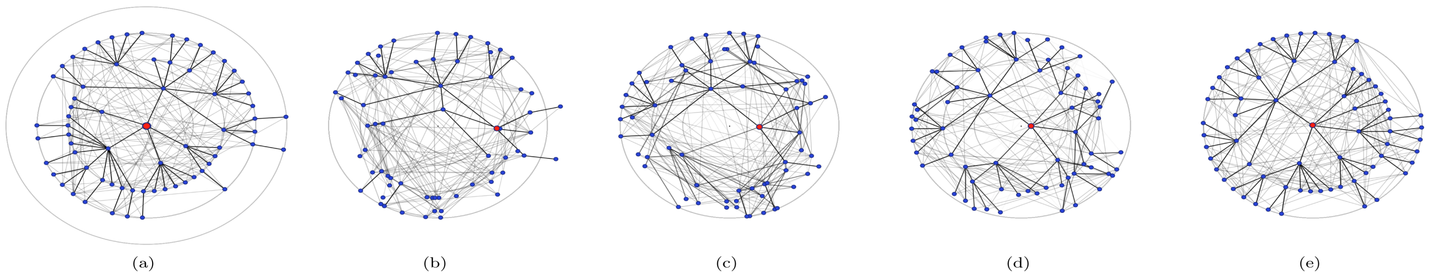


**Fig. 1.** An example transition between two spanning trees drawn using Eades' radial layout algorithm. Nodes are placed on rings concentric to the root node that correspond to their depth in the tree. Thick edges denote spanning tree edges; thin edges denote non-spanning tree edges. In Figure 1(b) a new root node is selected, the thick lines are redrawn to denote edges of the new spanning tree, and the graph begins to transition to the final layout for the new spanning tree, shown in Figure 1(e).

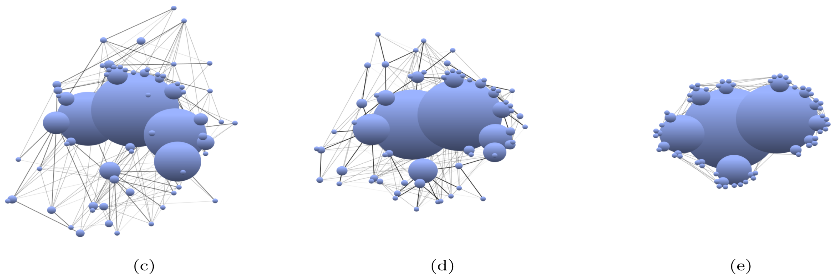


**Fig. 2.** In this alternative radial layout of the same tree as Figure 1, nodes are drawn as "balloons" where each child node is positioned on the centrifugal semi-circle of the balloon that lies outside its parent. No thick edges need to be drawn in Figure 2(a), because edges as well as nodes are denoted by the overlapping balloons. In Figure 2(b), a new node is selected, thick edges are used to denote the edges of the new spanning tree edges, and the graph begins to transition to the final layout for the new spanning tree in Figure 2(e), in which explictly drawn spanning tree edges are again unnecessary.

# Fast Node Overlap Removal—Correction

Tim Dwyer[1], Kim Marriott[1], and Peter J. Stuckey[2]

[1] Clayton School of IT, Monash University, Australia
{tdwyer,marriott}@mail.csse.monash.edu.au
[2] NICTA Victoria Laboratory
Dept. of Comp. Science & Soft. Eng., University of Melbourne, Australia
pjs@cs.mu.oz.au

**Keywords:** graph layout, constrained optimization, separation constraints.

## 1 Introduction

Our recent paper [1] details an algorithm for removing overlap between rectangles, while attempting to displace the rectangles by as little as possible. The algorithm is primarily motivated by the node-overlap removal problem in graph drawing. The algorithm treats $x$- and $y$-dimensions separately, each as an instance of the *variable placement with separation constraints (VPSC)* problem:

> Given $n$ variables $v_i \in V$, a weight $w_i \geq 0$ and a desired value $d_i$ for each variable and a set of separation constraints $C$ over these variables find an assignment to the variables which minimizes $\sum_{i=1}^{n} w_i \times (v_i - d_i)^2$ subject to $C$.

Each separation constraint $c \in C$ has form $u+g \leq v$ where $g \geq 0$ is the minimum separation between variables $u$ and $v$.

In [1] we gave a procedure, *satisfy_VPSC*, that was intended to find a feasible but possibly non-optimal solution to the VPSC. Unfortunately, the algorithm we gave for *satisfy_VPSC* contained an error which means that in rarely occurring cases it may return an infeasible solution.

The basic approach of the algorithm *satisfy_VPSC* was to merge variables into larger and larger "blocks" of variables connected by active constraints. The algorithm processed the variables from smallest to greatest based on some total order given by the relation $\preceq_C$ where $u \preceq_C v$ iff there is a constraint $c \in C$ of form $u + g \leq v$.

Naive implementation of this algorithm has worst-case complexity of $O(|V| \cdot |C|)$. In order to improve efficiency, the algorithm given in [1] used a priority queue for each block $b$ to store the block's "in" constraints, i.e. those constraints of form $u + g \leq v$ where $v$ is in block $b$, ordered by their violation. When two blocks were merged so were their priority queues. Implementing the priority queues as *pairing heaps* [2] improved the amortized worst case complexity of the algorithm to $O((|V| + |C|) \log |C|)$.

Unfortunately, we have subsequently realized that in rare cases merging of priority queues meant that the algorithm given in [1] could return a infeasible solution. The problem is that when a block $b$ is moved the violation of its "in" constraints changes value. It was claimed in [1], that the relative ordering of the constraints in the priority queue will not be changed as the result of a block's movement since all variables in the block will be moved by the same amount and so the violation of the constraints will be changed by the same amount for all non-internal constraints. This is true for the constraints in the priority queue of the current active block $b$ but may not be true for constraints in a priority queue of a non-active block $b'$ whose constraints refer to variables that are in $b$. This may mean that the priority queue of $b'$ becomes invalid and, if at some future time $b'$ becomes merged with the active block, can lead to an error.

It is relatively straightforward to fix the problem. We keep a time stamp for each block $b$, which is the time it last moved, and a time stamp for each constraint $c$ in a priority queue implemented as a pairing heap, which is the time it was placed in the priority queue. When a constraint $c$ is encountered in the priority queue of the currently active block $b$ (as the result of coming to the "top" of the heap during a remove operation) we check that the other block $b'$ that it refers to has not been moved since $c$ was placed in the priority queue. If this is not true, the relative placement of the constraint in the queue could be wrong, so a down heap operation is performed on the constraint. A down heap is required since $c$'s violation relative to the other constraints in the priority queue can only have decreased as block $b'$ must have moved to the left. Since $c$'s violation has decreased it is quite safe to lazily correct the problem.

The worst case complexity of the the corrected *satisfy_VPSC* is now $O((|V| \cdot |C|) \log |C|)$ since we might perform a down-heap operation repeatedly on the same constraint. However, if a depth-first traversal is used to construct the total ordering from the partial order, it is quite rare for this scenario to arise and in this case we believe that the expected amortized time complexity is $O((|V| + |C|) \log |C|)$. The revised algorithm is described in more detail in [3].

## References

1. Dwyer, T., Marriott, K., Stuckey, P.: Fast node overlap removal. In: Proceedings of the 13th International Symposium on Graph Drawing (GD'05). Volume 3843 of LNCS. (2006) 153–164
2. Weiss, M.A.: Data Structures and Algorithm Analysis in Java. Addison Wesley Longman (1999)
3. Dwyer, T., Marriott, K., Stuckey, P.: Fast node overlap removal correction. Available from: www.csse.monash.edu.au/∼tdwyer/FastNodeOverlapRemovalCorrection.pdf (2006)

# Graph-Drawing Contest Report

Christian A. Duncan[1], Gunnar Klau[2], Stephen G. Kobourov[3], and Georg Sander[4]

[1] Louisiana Tech University, Ruston, LA 71270, USA
duncan@latech.edu
[2] Freie Universität Berlin, 14195 Berlin, Germany
gunnar@math.fu-berlin.de
[3] University of Arizona, Tucson, AZ 85721, USA
kobourov@cs.arizona.edu
[4] ILOG, 94253 Gentilly Cedex, France
sander@ilog.fr

**Abstract.** This report describes the Thirteenth Annual Graph Drawing Contest, held in conjunction with the 2006 Graph Drawing Symposium in Karlsruhe, Germany. The purpose of the contest is to monitor and challenge the current state of the graph-drawing technology.

## 1 Introduction

This year's graph drawing contest had five distinct categories: the graph drawing challenge, the free-style contest, and three special graph categories. The graph drawing challenge, which took place during the conference, focused on area minimization of straight-line planar drawings. The free-style competition provided participants with the opportunity to present their best graph visualizations, with a focus on both aesthetic beauty as well as relevance to the graph drawing community. The first special category was a mystery theory graph of 101 nodes and 190 edges, with the judging of a graph based on both its visual and informational merit. The second special category focused on visualizing the History of the FIFA World Cup$^{\mathrm{TM}}$. This category had the requirement that the submission be an animation showing the evolution of the match results of the world cup over its long history. The third special category was more practically oriented. The task was to provide an informative visualization of the Java compile-time dependency graph with nodes representing Java classes and directed edges representing compile-time dependencies between two classes. Unfortunately, this task proved too challenging this year, and there were no submissions for this category. However, there were 20 submissions among the other categories. The remaining sections go into more details about each category and the winning submissions. Since many of the winning submissions were animations, interested viewers should visit the Contest's website[1] to download and view the winning animations along with their descriptions.

[1] http://gd2006.org/contest

## 2 Graph Drawing Challenge

This year's challenge dealt with minimizing the area of straight-line drawings of planar graphs. At the start of the one-hour on-site competition, the contestants were given six planar graphs ranging in size from 16 nodes to 400 nodes. The first four graphs were small enough to be manually created, and the last two were automatically generated. Vertices were assigned random grid positions in all of the graphs.

We allowed teams to participate in one of two categories, automated and manual. Manual teams came and solved the problems using ILOG's Simple Graph Editing Tool provided by the committee. The automated teams were allowed and highly encouraged to use additional software tools to help solve the problems. Interestingly, for two of the six graphs people manually obtained better solutions than the automated software.

Six teams entered the manual category and the winner was the team of Fabrizio Frati and Markus Geyer from Universitá Roma Tre and Universität Tübingen. Figure 1(a) shows their winning submission for Graph 2. Four teams entered the automated category and the winner was the team from Universität Tübingen led by Andreas Gerasch. Figure 1(b) shows their winning submission for Graph 4.

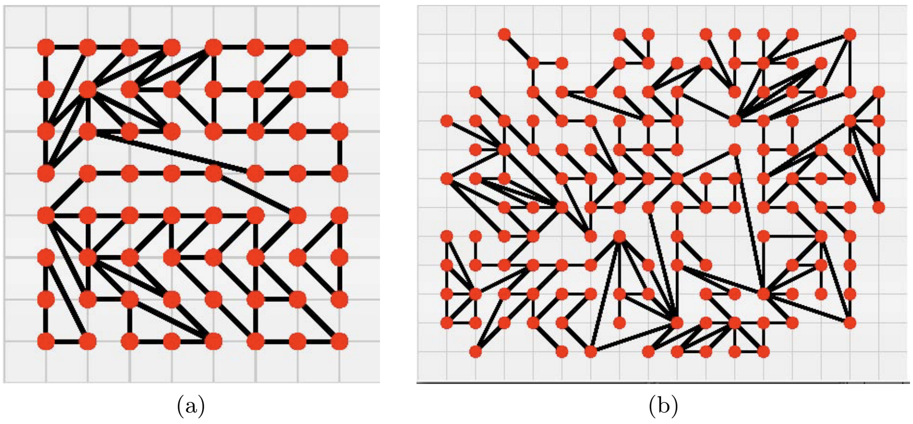


**Fig. 1.** The winning submissions for (a) Graph 2 by Frati and Geyer and (b) Graph 4 by the Tübingen team

## 3 Free-Style Contest

This year only two teams submitted entries to the free-style category. As there was no clear winner between the two, two honorable mentions were awarded.

Figure 2 shows the submission by the team of Emden Gansner and Yehuda Koren from AT&T Research Labs, which is an example of the improved circular layout algorithm presented at this year's symposium [1]. The graph represents

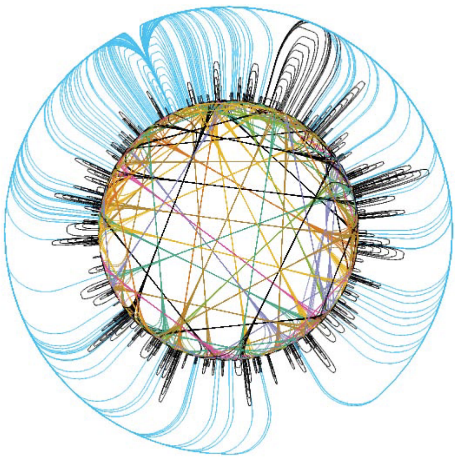


**Fig. 2.** An example drawing of the improved circular layout algorithm of Gansner and Koren

behind-the-scenes data sharing related to various popular websites. The graph has 2226 nodes and 4710 edges. The node with the highest degree (169) is ad.doubleclick.net, an advertising server. The external edges connected to this node are merged together, making the drawing cleaner. Otherwise, these external edges would dominate the drawing.

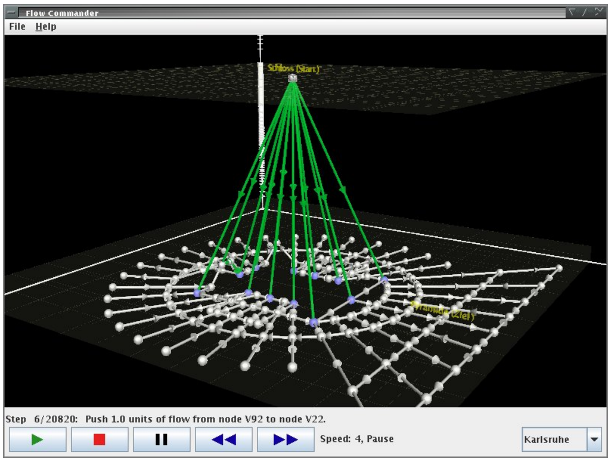
Figure 3 shows the submission by the team of Florian Böhl, Robert Görke, and Steffen Mecke from Universität Karlsruhe regarding their Flow Commander program.[2] The program provides an algorithm animation of the *Push-Relabel* network flow algorithm of Goldberg and Tarjan but is designed to support other graph algorithms. The application uses the Java3d library for a system independent means of interactively visualizing the animation in 3D-space.

## 4 Theory Graph Contest

There were five submissions in the theory-graph category with each contestant giving a different interpretation. The graph used was the graph $G_{10}$ described in [2] formed by the union of two trees $T_1 = (V, E_1)$ and $T_2 = (V, E_2)$ where each tree has a common root $v_0$ and 10 common children, $v_1, \ldots, v_{10}$. The root $r$ has 90 grand-children labeled $v_{ij}$ such that, for $1 \leq i, j \leq 10$ and $i \neq j$, we have $(v_i, v_{ij}) \in E_1$ and $(v_j, v_{ij}) \in E_2$. Several contestants observed that this graph is also identical to a modification of $K_{11}$ where every edge, except those adjacent to vertex $v_0$, is duplicated and then subdivided.

The committee awarded two first prizes to two equally interesting visualizations of the graph. Both winning submissions included animated visualizations of the graph. Figure 4 shows a snapshot of the winning submission by Michael

[2] See: http://i11www.iti.uni-karlsruhe.de/adw/jawsGD/GTVisualizer3D.jnlp



Step 6/20820: Push 1.0 units of flow from node V92 to node V22.

**Fig. 3.** The Flow Commander application for visualizing Network Flow algorithms

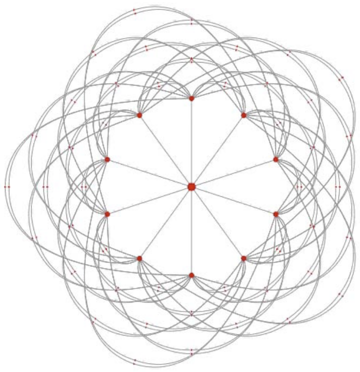
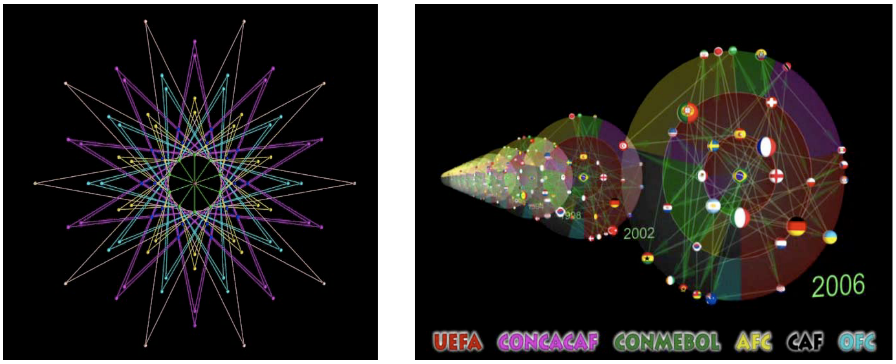


**Fig. 4.** A snapshot of McGuffin's winning animation of the theory graph

McGuffin of the Ontario Cancer Institute and the University of Toronto. Figure 5(a) shows a snapshot of the winning submission by Adel Admed, Seok-Hee Hong, Quan Nguyen, and Donald Taylor of the University of Sydney and National ICT Australia.

## 5 History of the World Cup Contest

There were three submissions for the History of the World Cup category. The data given was a collection of every match played in the final rounds of the FIFA World Cup including the country names and final scores. The broad goal was to provide a creative and informative *animated* visualization of the data as an evolving graph. There were many interesting submissions in this category,



(a) (b)

**Fig. 5.** A snapshot of the Sydney team's winning animations of (a) the theory graph and (b) the World Cup graph

and Figure 5(b) shows a snapshot of the winning animation of Adel Admed, Xiaoyan Fu, Seok-Hee Hong, Quan Nguyen, and Kai Xu from the University of Sydney and National ICT Australia. Their focus was to highlight strong countries and the performance of teams over time. In their approach, the team analyzed the graphs from three perspectives: clustering analysis, centrality analysis, and centrality-based thickness. They also visualized the graphs in three ways: radial view layout, centralily-based 2.5D drawings, and an evolutionary drawing based on team performance.

## Acknowledgments

We wish to thank the generous sponsors of the symposium and the contestants.

## References

1. Gansner, E.R., Koren, Y.: Improved circular layouts. In: Graph Drawing 2006. Lecture Notes in Computer Science, Springer (to appear)
2. Geyer, M., Kaufmann, M., Vrto, I.: Two trees which are self-intersecting when drawn simultaneously. In Healy, P., Nikolov, N.S., eds.: Graph Drawing. Volume 3843 of Lecture Notes in Computer Science., Springer (2005) 201–210

# Author Index