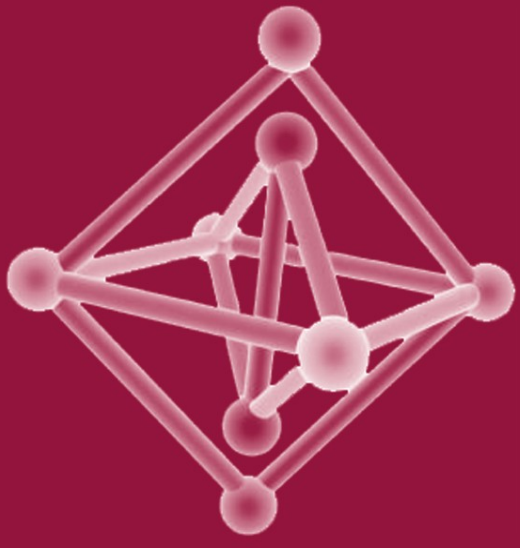
Seok-Hee Hong
Takao Nishizeki
Wu Quan (Eds.)

LNCS 4875

Graph Drawing

15th International Symposium, GD 2007
Sydney, Australia, September 2007
Revised Papers



Springer

Lecture Notes in Computer Science 4875

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Seok-Hee Hong Takao Nishizeki
Wu Quan (Eds.)

Graph Drawing

15th International Symposium, GD 2007
Sydney, Australia, September 24-26, 2007
Revised Papers

Springer

Volume Editors

Seok-Hee Hong
University of Sydney
School of Information Technologies, J12
NSW 2006, Australia
E-mail: shhong@it.usyd.edu.au

Takao Nishizeki
Tohoku University
Graduate School of Information Sciences
Sendai 980-8579, Japan
E-mail: nishi@ecei.tohoku.ac.jp

Wu Quan
University of Sydney
School of Information Technologies, J12
NSW 2006, Australia
E-mail: kevinqw@it.usyd.edu.au

Library of Congress Control Number: 2007942802

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 3-540-77536-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-77536-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12210626 06/3180 5 4 3 2 1 0

Preface

The 15th International Symposium on Graph Drawing (GD 2007) was held in Sydney, Australia, September 24–26, 2007.

In response to the call for papers, the Program Committee received 74 submissions. Each submission was reviewed by at least three Program Committee members. After an extremely rigorous review process and extensive discussions, the committee accepted 27 long papers and 9 short papers. All these 36 papers were presented at the conference. In addition, six posters were accepted and displayed throughout the conference.

Two distinguished speakers invited by the Program Committee Chairs delivered impressive talks during the conference. Brendan D. McKay from Australian National University gave the presentation on"Computing Symmetries of Combinatorial Objects," while Norishige Chiba from Iwate University talked about "Large-Scale Graphics: Digital Nature and Laser Projection."

The traditional graph drawing contest was successfully held under the direction of Christian Duncan. The report of the contest is included in the proceedings. A one-day workshop on Constraint-Based Layout of Diagrams and Documents was held in conjunction with the conference.

The year 2007 marks the 60th birthday of Takao Nishizeki. The symposium celebrated his vast contribution to graph drawing, graph algorithms, graph theory and security.

The conference received generous support from the hosting organization, the University of Sydney, and from our gold sponsors: Tom Swayer, ILOG, and HxI Initiative which includes NICTA, CSIRO, DSTO, as well as from the silver sponsor: yWorks.

We would like to thank all the Program Committee members and external referees for their excellent work, especially given the time constraints. We also thank all those who submitted papers for consideration, thereby contributing to the high quality of the conference.

Finally, we would like to express our deep gratitude to the Organizing Committe members, Sharon Chambers, Peter Eades, Wei-Ying Ho and Tony Huang, for their hard work in making the conference a great success.

Next year, the symposium will be held on Crete, Greece, September 22–24, organized by Ioannis Tollis.

October 2007

Seok-Hee Hong
Takao Nishizeki
Wu Quan

Organization

Steering Committee

Franz J. Brandenburg	University of Passau
Giuseppe Di Battista	Università Roma Tre
Peter Eades	National ICT Australia Ltd., University of Sydney
Hubert de Fraysseix	Centre d'Analyse et de Mathematique Sociale
Seok-Hee Hong	University of Sydney
Michael Kaufmann	University of 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

Giuseppe Di Battista	Università Roma Tre
Therese Biedl	University of Waterloo
Franz J. Brandenburg	University of Passau
Ulrik Brandes	University of Konstanz
Peter Eades	NICTA, University of Sydney
Hubert de Fraysseix	Centre d'Analyse et de Mathematique Sociale
Emden R. Gansner	AT&T Labs
Seok-Hee Hong	University of Sydney (Co-chair)
Giuseppe Liotta	Università degli Studi di Perugia
Kim Marriott	Monash University
Petra Mutzel	University of Dortmund
Hiroshi Nagamochi	Kyoto University
Takao Nishizeki	Tohoku University (Co-chair)
János Pach	New York University
Md. Saidur Rahman	BUET
Roberto Tamassia	Brown University
Ioannis G. Tollis	University of Crete
Dorothea Wagner	Universität Karlsruhe
Sue Whitesides	McGill University
Stephen Wismath	University of Lethbridge
Hsu-Chun Yen	National Taiwan University

Organizing Committee

Sharon Chambers	University of Sydney
Peter Eades	NICTA, University of Sydney
Wei-ying Ho	University of Sydney
Seok-Hee Hong	University of Sydney (Co-chair)
Tony Huang	University of Sydney
Wu Quan	University of Sydney (Co-chair)

Contest Committee

Christian A. Duncan	Louisiana Tech University (Chair)
Stephen G. Kobourov	University of Arizona
Georg Sander	ILOG

External Referees

Patrizio Angelini
Tanveer Awal
Christian Bachmeier
Melanie Badent
Reinhard Bauer
Michael Baur
Carla Binucci
Krists Boitmanis
Wolfgang Brunner
Timothy Chan
Markus Chimani
Pier Francesco Cortese
Daniel Delling
Walter Didimo
Tim Dwyer
Fabrizio Frati
Eric Fusy
Emilio Di Giacomo
Luca Grilli
Carsten Gutwenger
Robert Görke
Martin Harrigan
Patrick Healy
Andreas Hofmeier
Martin Holzer
Md. Rezaul Karim
Bastian Katz
Ken-ichi Kawarabayashi
Karsten Klein
Yehuda Koren
Shankar Krishnan
Ago Kuusik
Anna Lubiw
Steffen Mecke
Sascha Meinert
Kazuyuki Miura
Martin Nöllenburg
Stephen North
Pietro Palladino
C. Papamanthou
Maurizio Patrignani
Maurizio Pizzonia
Ignaz Rutter
Md. Abul Hassan Samee
Thomas Schank
Michael Schulz
Khaled M. Shahriar
Geza Toth
Vassilis Tsiaras
Hoi-Ming Wong
David Wood
Michael Wybrow

Sponsoring Institutions


The University of Sydney
Australia

Changing the rules of business


Australian Government
Department of Defence
Defence Science and
Technology Organisation

Table of Contents

Invited Talks

Papers

Graph Drawing Contest

Computing Symmetries of Combinatorial Objects
(Abstract)

Brendan D. McKay

Department of Computer Science,
Australian National University
bdm@cs.anu.edu.au

We survey the practical aspects of computing the symmetries (automorphisms) of combinatorial objects. These include all manner of graphs with adornments, matrices, point sets, etc.. Since automorphisms are just isomorphisms from an object to itself, the problem is intimately related to that of finding isomorphisms between two objects.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, p. 1, 2007.

Large-Scale Graphics: Digital Nature and Laser Projection
(Abstract)

Norishige Chiba

Department of Computer Science, Faculty of Engineering,
Iwate University, Japan
nchiba@cis.iwate-u.ac.jp

In this talk, I will sketch out two challenging research topics by showing computer generated visual materials. One is raster-graphics technologies on how to represent large-scale natural sceneries, and the other is laser projection technologies enabling us to display large-scale vector graphics. The former topic includes the modeling and rendering techniques having the both abilities of LOD (Level-Of-Detail) and anti-aliasing indispensable for efficiently and effectively representing large-scale scenes including a huge amount of fine objects like botanical trees, and the efficient real-time animation techniques implemented by utilizing 1/ - noise for defeating the computational time required for strict physically-based simulation. The latter topic is the exploratory research on laser projection where there is almost no researcher yet. Laser graphics has strong relation to pen and ink illustration in the field of NPR (Non-Photorealistic-Rendering) and might be usable to represent Graph Drawing.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, p. 2, 2007.

Crossing Number of Graphs with Rotation Systems

Michael J. Pelsmajer[1], Marcus Schaefer[2], and Daniel Štefankovič[3]

[1] Illinois Institute of Technology, Chicago, IL 60616, USA
pelsmajer@iit.edu
[2] DePaul University, Chicago, IL 60604, USA
mschaefer@cs.depaul.edu
[3] University of Rochester, Rochester, NY 14627, USA
stefanko@cs.rochester.edu

Abstract. We show that computing the crossing number of a graph with a given rotation system is **NP**-complete. This result leads to a new and much simpler proof of Hliněný's result, that computing the crossing number of a cubic graph (without rotation system) is **NP**-complete. We also investigate the special case of multigraphs with rotation systems on a fixed number k of vertices. For $k = 1$ and $k = 2$ the crossing number can be computed in polynomial time and approximated to within a factor of 2 in linear time. For larger k we show how to approximate the crossing number to within a factor of $\binom{k+4}{4}/5$ in time $O(m^{k+2})$ on a graph with m edges.

Keywords: crossing number, computational complexity, computational geometry.

1 Introduction

Computing the crossing number is **NP**-complete, as shown by Garey and Johnson [5]. Hliněný recently showed, using a rather complicated construction, that even determining the crossing number of a cubic graph is **NP**-complete [6], a long-standing open problem [1].

We investigate a new approach to cubic graphs through graphs with rotation systems. We show that determining the crossing number of a graph with a given rotation system is **NP**-complete, and then prove that this problem is equivalent to determining the crossing number of a cubic graph. This also gives a new and easy proof that determining the minor-monotone crossing number (defined in [2]) is **NP**-complete.

Graphs with rotation systems are of interest in their own right; we have encountered them several times during recent research projects [12,14,13]. Indeed, at the core of our separation of the crossing number from the odd crossing number is a loopless multigraph on two vertices with rotation [13]. In Section 4 we will see that the crossing number can be computed efficiently for one-vertex graphs with rotation and at least approximated efficiently for loopless multigraphs on

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 3–12, 2007.

two vertices (the problem is in polynomial time for two-vertex multigraphs but requires linear programming [14]). We also show some interesting connections to string matching problems. Finally, we give an approximation algorithm to compute the crossing number of k-vertex multigraphs with rotation to within a factor of $O(k^4)$. We do not know whether this problem can be solved exactly in polynomial time.

2 NP- Hardness

Consider a graph drawn in the plane (or any orientable surface). The *rotation* of a vertex is the clockwise order of its incident edges. A *rotation system* is the list of rotations of every vertex. We are interested in drawings of a graph in the plane with a fixed rotation system.

We also consider "flipped" rotations (previously seen in [13]). Given a rotation of a vertex v, the *flipped rotation* reverses the cyclic order of the edges incident to v.

Theorem 1. *Computing the crossing number of a graph with a given rotation system is* **NP***-complete. The problem remains* **NP***-complete if we allow the rotation at each vertex to flip independently.*

Proof. We adapt Garey and Johnson's reduction from OPTIMAL LINEAR ARRANGEMENT to CROSSING NUMBER [5]. Given a graph $G = (V, E)$, a *linear arrangement* is an injective function $f : V \to 1, \ldots, |V|$, and the *value* of the arrangement is computed as

$$\sum_{uv \in E} |f(u) - f(v)|.$$

Given G and k, deciding whether G allows a linear arrangement of value at most k is **NP**-complete [5, GT42].

Let us fix a connected graph $G = (V, E)$, with $V = v_1, \ldots, v_n$, $m = |E|$, and k. We may assume that $n \leq m$. From G we construct an edge-weighted graph H with fixed rotation system, as shown in Figure 1. The use of weighted edges simplifies the construction; later we will replace each weighted edge by a small unweighted graph, obtaining a simple graph H' with a fixed rotation system. Note that for a fixed drawing of a weighted graph, a crossing of an edge of weight k with an edge of weight l contributes kl to the crossing number.

We start with a cycle $(u_1, \ldots, u_{4n})$, and a single vertex u_0 connected to each vertex on the cycle. We choose the edge-weights of this part of the graph so high that it has to be embedded without any intersections.

For every $1 \leq i \leq 2n$ we connect u_i to u_{4n+1-i} by a path P_i of length 2 and edges of weight w. Furthermore, we connect the midpoints of P_i and P_{2n+1-i} by a path Q_i of length 3 with edges of weight w', whose middle edge $a_i b_i$ has been replaced by two edges of weight $w'/2$ $(1 \leq i \leq n)$.

Finally, we encode G as follows: for each edge $v_i v_j \in E$ we add an edge from a_i to b_j (with $i < j$, an arbitrary choice). The rotation of H is as shown in

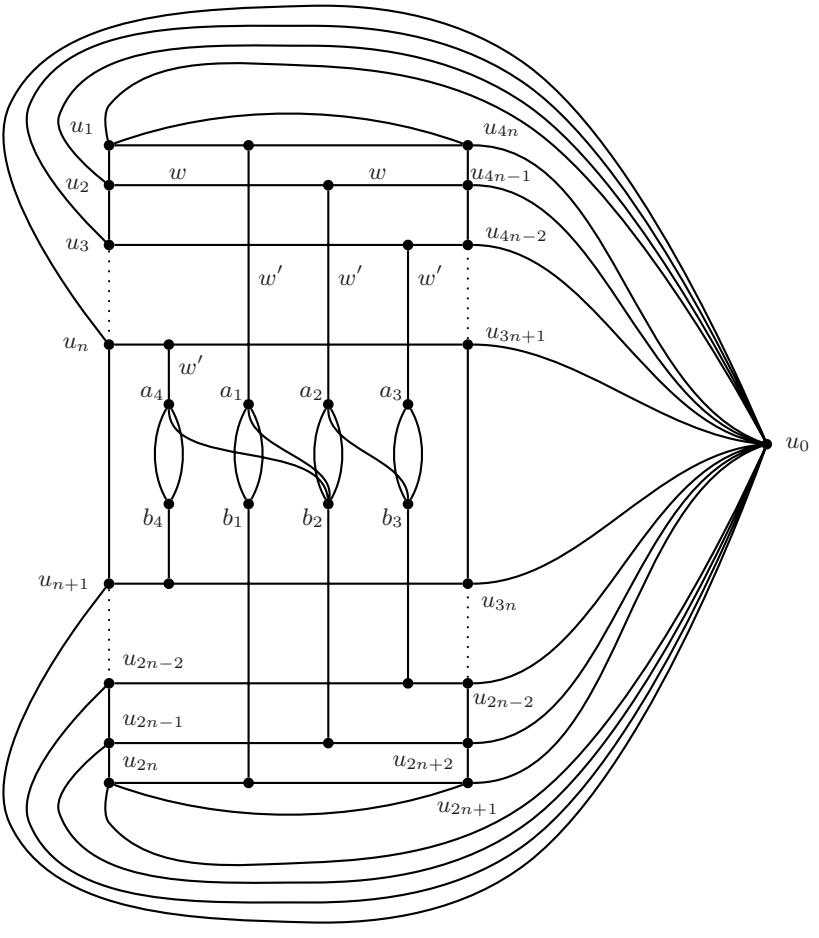


Fig. 1. The graph H

Figure 1. At a_i, each edge from E is inserted into the rotation at a_i between the two a_i, b_i-edges of weight $w'/2$; we do likewise at every b_i. The edges of E at a_i can be ordered arbitrarily (same at b_i).

This concludes the description of H. We let $k' = n(n-1)ww' + kw' + m^2$, where $w = 5m^4$ and $w' = 2m^2$. We claim that G allows a linear arrangement of value at most k if and only if H (with the rotation system shown in the drawing) has crossing number at most k'.

If G has a linear arrangement of value at most k, we can draw H using the order of the v_i in that linear arrangement to obtain a drawing of crossing number at most k' (the m^2 term compensates for the potential pairwise crossings of the edges in H that represent edges in E).

For the reverse implication, consider a drawing of H with crossing number at most $k' = n(n-1)ww' + kw' + m^2$. Then $k' < n^2ww' + m^2w' + m^2$, and

by choice of w and w' this is at most $10m^8 + 2m^4 + m^2 < w^2$. Hence, in our drawing, no two edges of weight w intersect each other, and, therefore, the paths P_i $(1 \le i \le 2n)$ are drawn as shown in Figure 1.

Next, consider the modified paths Q_i. Q_i must intersect each of the paths P_{i+1} through P_{2n-i}, contributing $(2n-2i)ww'$ to the crossing number. Summing these values for $i = 1, \ldots, n$, we observe a contribution of at least $n(n-1)ww'$ by intersections between the Q_i and the P_i to the crossing number. This leaves $k' - n(n-1)ww' = kw' + m^2 < m^2w' + m^2 = (w'/2)w' + w'/2 < w'w' < w'w$ crossings, implying that there cannot be any further intersections between a Q_i and a P_i (since it would contribute $w'w$ to the crossing number, more than is left). By the same reasoning, we also do not have intersections between any two Q_i.

Finally, we want to argue that all the a_i and b_i lie between P_n and P_{n+1}. Since Q_n lies entirely between P_n and P_{n+1} (as we argued earlier), so do a_n and b_n. Consider any a_i or b_i. As G is connected by assumption, there is a path from a_n to a_i using edges encoding G and edges of weight $w'/2$. If this path intersects P_n or P_{n+1}, it contributes w or more to the crossing number. However, since $k' - n(n-1)ww' = kw' + m^2 < m^2w' + m^2 = 2m^4 + m^2 < 5m^4 = w$, this is not possible. Therefore, a_i and b_i are also located between P_n and P_{n+1}.

In summary, the drawing of H looks as shown in Figure 1. This drawing clearly indicates a linear arrangement f of G. An edge $e = uv$ contributes at least $|f(u) - f(v)|w'$ to the crossing number of H, so $\sum_{uv \in E} |f(u) - f(v)| \le kw' + m^2$. Since $m^2 < w$, the value of the linear arrangement is at most k.

The last step is to replace each edge e of weight x by x parallel edges, and then subdivide each of those edges: the effect is that e is replaced by a copy of $K_{2,x}$ with the endpoints of e identified with the partite set of size 2. The new edges are inserted in the rotation at where e was, and the new edges are ordered as indicated in Figure 2. Thus we obtain an unweighted graph H' from H. Since we can draw any of the parallel edges alongside whichever one is involved in the smallest number of crossings, we may assume that an optimal drawing of H' has all parallel edges routed in parallel; also, subdivisions do not affect the crossing number. Therefore, $\mathrm{cr}(H') = \mathrm{cr}(H)$, and H' is an unweighted graph with fixed rotation system for which is it is **NP**-hard to determine the crossing number.

Note that the argument showing that the drawing of H looks as shown in Figure 1 did not make any assumptions about the rotation at a vertex. Therefore, even if we allow flipped rotations, we can still conclude that the drawing of H yields a linear arrangement of value at most k. Consequently, computing the crossing number of graphs with rotation systems remains **NP**-complete if we allow rotations to flip.

Remark 1. The construction in the proof of Theorem 1 can be modified to work for other crossing number variants, such as odd-crossing number, pair-crossing number, and rectilinear crossing number (for which all edges of the graph have to be realized as line segments).

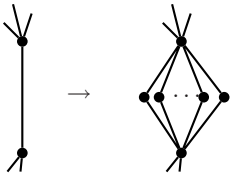


Fig. 2. Replacing an edge by parallel paths

3 Cubic Graphs

We can use Theorem 1 to prove that computing the crossing number of a cubic graph is **NP**-complete. This was a long-standing open question that was solved only recently by Petr Hliněný, using a rather complicated construction. The idea of the proof is to replace each vertex of a graph with rotation system with a hexagonal grid, simultaneously making the graph cubic and mimicking the rotation system. (Hexagonal grids are used in Hliněný's original proof as well.)

Theorem 2 (Hliněný [6]). *Computing the crossing number of a* 3*-connected, cubic graph is* **NP***-complete.*

Remark 2. The argument of Theorem 2 also works for straight-line drawings. Combining this observation with Remark 1 shows that it is **NP**-hard to compute the rectilinear crossing number of a cubic graph.

As Hliněný observes, Theorem 2 also implies that computing the minor-monotone crossing number is **NP**-complete [6]. Another result, which follows immediately (as observed in [3]) is that it is **NP**-hard to find a drawing of a directed graph in which all incoming (and therefore all outgoing) edges at a vertex are consecutive and which minimizes the crossing number.

Our Theorem 1 is in turn derivable from Hliněný's result, as we will show in the full version of the paper.

4 Parameterization

One way to parameterize the crossing number problem is by the number of vertices of the graph. The question becomes interesting if we allow multiple edges and loops. Without rotation, the problem is equivalent to computing the crossing number of a weighted graph without multiple edges and loops, with the cost of an intersection being the product of the weights of the edges involved: Given a graph $G = (V, E)$ with multiple edges and loops, note that in a crossing-number optimal drawing any two edges with the same endpoints can be routed in parallel. If we let G' be the complete graph on V with edge weights $w(uv)$ equal to the number of edges in E between u and v, then the weighted crossing number of G' equals $\mathrm{cr}(G)$. Moreover, that weighted crossing number of G' can be easily computed by exhaustively trying all possible drawings in time $O(2^{|V|^2}(\log|E| + |V|^2))$.

The problem becomes nontrivial if the graph G is given with a rotation system of its edges. In the following sections we discuss the cases of one and two vertices connecting them with well-known problems such as determining the number of inversions in a permutation and finding the edit distance of two cyclic words. We also include a weak approximation result for the general case. We start by investigating the case of two vertices.

4.1 Two Vertices

In this section we consider graphs on two vertices, allowing multiple edges, but no loops. The crossing number of a two-vertex graph can be expressed as the solution of an integer linear program whose relaxation can be used to compute the optimal integer solution in polynomial time as we showed earlier [13].

Here we want to give a fast and simple 2-approximation algorithm for the two-vertex case. To do so, we look at the crossing number problem as an *edit-distance* problem on words. The edit distance between two words is the smallest number of operations transforming one word into the other. There are numerous variants of this problem depending on which operations are allowed and what the associated costs are [15,9]. There are also several papers studying objects other than words, such as trees and cyclic words (also known as necklaces) [10,11,7], but it seems the particular variant we find needful here—allowing only swaps (at unit cost) on cyclic words—has not so far been considered at all. A *swap* is the transposition of two adjacent letters in a word. A *cyclic word* is the equivalence class of a word under cyclic shifts (we will use the letter ρ to denote the cyclic shift of a word by one position to the right). The last and first letter of a cyclic word are considered adjacent. Let $d_s(u, v)$ be the smallest number of swaps transforming u into v, where u and v are normal words. Similarly, let $d_s^\rho(u, v)$ be the smallest number of swaps transforming u into v allowing cyclic shifts at no cost. Then $d_s^\rho(u, v)$ is the swapping distance of the two *cyclic* words represented by the words u and v. E.g. $d_s^\rho(abcd, cdba) = 1$, while $d_s(abcd, cdba) = 5$.

Computing d_s is easy (see [15]). Our goal is the computation of $d_s^\rho(u, v)$.

Swapping distance of Cyclic Words
Instance: Two words u, v, integer k.
Question: Is $d_s^\rho(u, v) \leq k$?

We do not know how hard this problem is in general; however, with the restriction that the words contain each letter exactly once, we can solve the problem. Indeed, in that case it is equivalent to computing the crossing number of a graph G with rotation system on two vertices (details will appear in the journal version).

We rephrase the restricted swapping-distance problem as follows: we can assume that $u = 123\cdots m$ and $v = \sigma(1)\sigma(2)\cdots\sigma(m)$ for some permutation σ of Z_m (the cyclic group of m elements). Letting G be the 2-vertex multigraph defined by the clockwise rotations u and v^R, we define $\mathrm{cr}(\sigma) := \mathrm{cr}(G)$. We call two permutations σ, τ *circular-equivalent* if there exists a k such that $\sigma(i) = \tau(i + k)$ for all $i \in Z_n$. Each equivalence class is a *circular permutation* (this corresponds

exactly to the cyclic words). We will use σ to represent a permutation as well as the corresponding circular permutation. If σ and τ are circular equivalent, then $\mathrm{cr}(\sigma) = \mathrm{cr}(\tau)$.

We next define a function $\widetilde{\mathrm{cr}}$ on circular permutations σ which will be seen to be related to the crossing number of the corresponding 2-vertex multigraph G. Consider a fixed permutation τ. We wish to consider "forward" and "backward distance" from i to $\tau(i)$ in Z_m, as if the the elements in the list τ were placed clockwise along a circle with the same distance between each consecutive pair (including $\tau(m)$ and $\tau(1)$). We define $d^+(i)$ to be $\tau(i) - i \mod m$; note that $0 \leq d^+(i) < m$. Also let $d^-(i) = i - \tau(i) \mod m$ and let $d(i) = \min(d^+(i), d^-(i))$. Note that if the aforementioned circle has circumference m, then $d^+(i)$ measures the clockwise distance along the circle from i to $\tau(i)$, and $d^-(i)$ measures the counterclockwise distance from i to $\tau(i)$. Finally, we define $d(\tau)$ to be the sum of $d(i)$ over $1 \leq i \leq m$.

For a circular permutation σ, let $\widetilde{\mathrm{cr}}(\sigma)$ be the minimum of $d(\tau)$ over all $\tau \equiv \sigma$. Equivalently, $\widetilde{\mathrm{cr}}(\sigma) = \min_{1 \leq i \leq m} d(\sigma \circ \rho^i)$, where $\rho^i(j) = i + j$ for all $1 \leq i \leq m$.

We claim that $\widetilde{\mathrm{cr}}$ approximates the cyclic swapping distance of two words to within a factor of 2. We leave the proof to the journal version.

Theorem 3. *For a 2-vertex loopless multigraph G represented by a circular permutation σ,*

$$\mathrm{cr}(G) \leq \widetilde{\mathrm{cr}}(\sigma) \leq 2\,\mathrm{cr}(G).$$

Remark 3. The bounds of Theorem 3 are asymptotically optimal: for $\sigma_n := (1\ 2)(3\ 4)\cdots(2n-1\ 2n)$ we have $\widetilde{\mathrm{cr}}(\sigma) = 2n$ and $\mathrm{cr}(G) = n$; for the lower bound consider $\tau_n := (1\ n)$ (as a permutation of numbers $1, \ldots, 2n$), then $\widetilde{\mathrm{cr}}(\tau_n) = 2n - 2$ and $\mathrm{cr}(G) = 2n - 3$.

Remark 4. We have seen that the crossing number of a two-vertex graph equals the swapping distance of two cyclic words. If instead of cyclic words we consider normal words, the swapping distance still equals the crossing number of a two-vertex graph where both vertices lie on the boundary of a disk (and all the edges are within the disk). In that context, the analogue of Theorem 3 is known as Spearman's Footrule and was first proved by Diaconis and Graham [4].

Theorem 3 gives us a fast and easy way to approximate $\mathrm{cr}(G)$ for a 2-vertex multigraph with rotation system. Computing $\widetilde{\mathrm{cr}}(\sigma)$ from the definition can be done in quadratic time; however, this can easily be improved by first sorting the $d(i)$ (which can be done in linear time) and then trying all rotational shifts ρ^j of σ. We keep the optimal shifts sorted by value and distinguish between two different types of optimal shift: forward and backward. Updating the optimal shift and its direction might not be constant time for adding a single shift, but an amortized analysis shows that the whole algorithm can be made to run in linear time.

Corollary 1. *The crossing number of a 2-vertex loopless multigraph with rotation system can be approximated to within a factor of 2 in linear time.*

4.2 One Vertex

Given a graph with a rotation system on a single vertex (with loops), it is quite straightforward to compute its crossing number in quadratic time.

In contrast, a linear time algorithm for the one-vertex case would come as a surprise, since the problem contains as a special case a well-studied problem: computing the number of inversions of a permutation. Given a permutation π over $\{1,\dots,n\}$, an *inversion* of π is a pair (i,j) such that $i<j$ and $\pi(i)>\pi(j)$. It is well-known that the number of inversions of a permutation π equals $d_s(123\dots n, \pi(1)\pi(2)\dots\pi(n))$ (see, for example [8, Section 5.1.1]). The best-known algorithms for either problem run in $\Theta(n\log n)$.[1]

The inversion problem is easily encoded as a crossing number problem on a single vertex: simply let the rotation at the vertex be $12\dots n\pi(n)\pi(n-1)\dots\pi(2)\pi(1)$.

However, the one-vertex case can also be considered a special case of the two-vertex case (split the vertex into two and connect the two vertices with a large number of neighboring parallel edges). Hence we can approximate the crossing number of a one-vertex graph and therefore the number of inversions of a permutation in linear time to within a factor of 2 using our approximation algorithm. As we mentioned in Remark 4, this result is known as Spearman's Footrule.

We can compute the crossing number of a one-vertex graph exactly in time $\Theta(n\log n)$, which extends the algorithm for computing the number of inversions of a permutation. The proof will appear in the journal version.

Theorem 4. *The crossing number of a one-vertex graph with rotation system can be computed in time $O(n\log n)$.*

4.3 Several Vertices

There is little we can say at this point about how hard it is to compute the crossing number of a graph with a rotation system on a fixed number k of vertices when $k\geq 3$. Using results from a previous paper [12], however, we can give at least an approximation result. In this section we allow both loops and multiple edges.

Theorem 5. *We can approximate the crossing number of a multigraph $G=(V,E)$ with rotation system on k vertices to within a factor of $\binom{k+4}{4}/5$ in time $O(m^{k+2})$ where $k=|V|$ and $m=|E|$.*

In [12] we showed that $\mathrm{cr}(G)\leq \mathrm{ocr}(G)\binom{k+4}{4}/5$, where $\mathrm{ocr}(G)$ is the *odd-crossing number* of G, that is, the smallest number of pairs of edges that cross an odd number of times in any drawing of G. In fact, the proof applies to a multigraph G with rotation system π, yielding $\mathrm{cr}(G,\pi)\leq \mathrm{ocr}(G,\pi)\binom{k+4}{4}/5$. The proof works by choosing a sequence of k edges $e_1,\dots,e_k$ and contracting G along those edges

[1] See [8, Exercises 5.1.1-6 and 5.2.4.-21]. Wagner's linear time algorithm [15] for computing the swapping distance of words is wrong.

obtaining a graph G' with rotation system π' on a single vertex. For graphs on a single vertex crossing number and odd crossing number are the same, hence, $\mathrm{cr}(G', \pi') = \mathrm{ocr}(G', \pi')$. Furthermore, the sequence of edges is chosen such that $\mathrm{cr}(G', \pi') \leq \mathrm{ocr}(G, \pi)\binom{k+4}{4}/5$. In other words, $\mathrm{ocr}(G, \pi) \geq \mathrm{cr}(G', \pi')/(\binom{k+4}{4}/5)$. The redrawing procedure of the proof establishes that $\mathrm{ocr}(G, \pi) \leq \mathrm{ocr}(G', \pi')$. Introducing $c := \mathrm{ocr}(G', \pi')$ allows us to summarize the discussion as

$$c/(\binom{k+4}{4}/5) \leq \mathrm{ocr}(G, \pi) \leq c.$$

Since $\mathrm{ocr}(G, \pi) \leq \mathrm{cr}(G, \pi) \leq \mathrm{ocr}(G, \pi)\binom{k+4}{4}/5$, we conclude that

$$c/(\binom{k+4}{4}/5) \leq \mathrm{cr}(G, \pi) \leq c\binom{k+4}{4}/5.$$

Now c can be computed in time $O(m^2)$ using the trivial algorithm for one-vertex graphs if we know G' and π'. The only remaining problem is that we do not know the sequence of edges that determines G' and π'. Hence we have to try all possible sequences, giving a running time of $O(m^{k+2})$.

References

1. Archdeacon, D.: Problems in topological graph theory (accessed Septmeber 15, 2006), http://www.emba.uvm.edu/~archdeac/problems/npcubic.htm
2. Bokal, D., Fijavž, G., Mohar, B.: Minor-monotone crossing number. In: Felsner, S. (ed.) EuroComb 2005. Discrete Mathematics and Theoretical Computer Science, vol. AE, pp. 123–128 (2005)
3. Buchheim, C., Jünger, M., Menze, A., Percan, M.: Directed crossing minimization. Technical report, Zentrum für Angewandte Informatik Köln, Lehrstuhl Jünger (August 2005)
4. Diaconis, P., Graham, R.L.: Spearman's footrule as a measure of disarray. J. Roy. Statist. Soc. Ser. B 39(2), 262–268 (1977)
5. Garey, M., Johnson, D.: Crossing number is NP-complete. SIAM Journal on Algebraic and Discrete Methods 4, 312–316 (1983)
6. Hliněný, P.: Crossing number is hard for cubic graphs. J. Combin. Theory Ser. B 96(4), 455–471 (2006)
7. Kedem, Z.M., Fuchs, H.: On finding several shortest paths in certain graphs. In: 18th Allerton Conference, pp. 677–686 (1980)
8. Knuth, D.E.: The art of computer programming. In: Sorting and searching, Addison-Wesley Series in Computer Science and Information Processing, vol. 3, Addison-Wesley Publishing Co., Reading (1973)
9. Lowrance, R., Wagner, R.A.: An extension of the string-to-string correction problem. J. Assoc. Comput. Mach. 22, 177–183 (1975)
10. Maes, M.: On a cyclic string-to-string correction problem. Inform. Process. Lett. 35(2), 73–78 (1990)
11. Marzal, A., Barrachina, S.: Speeding up the computation of the edit distance for cyclic strings. In: International Conference on Pattern Recognition, pp. 891–894 (2000)

12. Štefankovič, D., Pelsmajer, M.J., Schaefer, M.: Removing even crossings. In: Felsner, S. (ed.) EuroComb 2005, DMTCS Proceedings. Discrete Mathematics and Theoretical Computer Science, vol. AE, pp. 105–110 (2005)
13. Pelsmajer, M.J., Schaefer, M., Štefankovič, D.: Odd crossing number is not crossing number. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 386–396. Springer, Heidelberg (2006)
14. Pelsmajer, M.J., Schaefer, M., Štefankovič, D.: Removing even crossings. J. Combin. Theory Ser. B (to appear)
15. Wagner, R.A.: On the complexity of the extended string-to-string correction problem. In: Robert, A. (ed.) Seventh Annual ACM Symposium on Theory of Computing, Albuquerque, N.M., Assoc. Comput. Mach., New York, pp. 218–223 (1975)

A Bipartite Strengthening of the Crossing Lemma

Jacob Fox[1,⋆], János Pach[2,⋆⋆], and Csaba D. Tóth[3]

[1] Department of Mathematics, Princeton University, Princeton, NJ, USA
jacobfox@math.princeton.edu
[2] City College, CUNY and Courant Institute, NYU, New York, NY, USA
pach@cims.nyu.edu
[3] University of Calgary, Calgary, AB, Canada
cdtoth@ucalgary.ca

Abstract. The celebrated Crossing Lemma states that, in every drawing of a graph with n vertices and $m \geq 4n$ edges there are at least $\Omega(m^3/n^2)$ pairs of crossing edges; or equivalently, there is an edge that crosses $\Omega(m^2/n^2)$ other edges. We strengthen the Crossing Lemma for drawings in which any two edges cross in at most $O(1)$ points.

We prove for every $k \in \mathbb{N}$ that every graph G with n vertices and $m \geq 3n$ edges drawn in the plane such that any two edges intersect in at most k points has two disjoint subsets of edges, E_1 and E_2, each of size at least $c_k m^2/n^2$, such that every edge in E_1 crosses all edges in E_2, where $c_k > 0$ only depends on k. This bound is best possible up to the constant c_k for every $k \in \mathbb{N}$. We also prove that every graph G with n vertices and $m \geq 3n$ edges drawn in the plane with x-monotone edges has disjoint subsets of edges, E_1 and E_2, each of size $\Omega(m^2/(n^2 \operatorname{polylog} n))$, such that every edge in E_1 crosses all edges in E_2. On the other hand, we construct x-monotone drawings of bipartite dense graphs where the largest such subsets E_1 and E_2 have size $O(m^2/(n^2 \log(m/n)))$.

1 Introduction

The crossing number $\operatorname{cr}(G)$ of a graph[1] G is the minimum number of crossings in a drawing of G. A *drawing* of a graph G is a planar embedding which maps the vertices to distinct points in the plane and each edge to a simple continuous arc connecting the corresponding vertices but not passing through any other vertex. A *crossing* is a pair of curves and a common interior point between the two curves (the intersections at endpoints or vertices do not count as crossings). A celebrated result of Ajtai et al. [ACNS82] and Leighton [L84], known as

⋆ Research supported by an NSF Graduate Research Fellowship and a Princeton Centennial Fellowship.
⋆⋆ Supported by NSF Grant CCF-05-14079, and by grants from NSA, PSC-CUNY, Hungarian Research Foundation OTKA, and BSF.

[1] The graphs considered here are simple, having no loops or parallel edges.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 13–24, 2007.

the Crossing Lemma, states that the crossing number of every graph G with n vertices and $m \geq 4n$ edges satisfies

$$\mathrm{cr}(G) = \Omega\left(\frac{m^3}{n^2}\right). \tag{1}$$

The best known constant coefficient is due to [PRTT06]. Leighton [L84] was motivated by applications to VLSI design. Székely [S97] used the Crossing Lemma to give simple proofs of Szemerédi-Trotter bound on the number of point-line incidences [ST83], a bound on Erdős's unit distance problem and Erdős's distinct distance problem [E46]. The Crossing Lemma has since found many important applications, in combinatorial geometry [D98, KT04, PS98, PT02, STT02], and number theory [ENR00, TV06].

The *pairwise crossing number* pair-cr(G) of a graph G is the minimum number of pairs of crossing edges in a drawing of G. The lower bound (1) also holds for the pairwise crossing number with the same proof. It follows that in every drawing of a graph with n vertices and $m \geq 4n$ edges, there is an edge that crosses at least $\Omega(m^2/n^2)$ other edges. Conversely, if in every drawing of every graph with $m \geq 3n$ edges some edge crosses $\Omega(m^2/n^2)$ others, then we have pair-cr$(G) = \Omega(m^3/n^2)$ for every graph G with $m \geq 4n$ edges. Indeed, by successively removing edges that cross many other edges, we obtain the desired lower bound for the total number of crossing pairs. In this note, we prove a bipartite strengthening of this result for drawings where any two edges intersect in at most a constant number of points.

Theorem 1. *For every $k \in \mathbb{N}$, there is a constant $c_k > 0$ such that for every drawing of a graph $G = (V, E)$ with n vertices and $m \geq 3n$ edges, no two of which intersect in more than k points, there are disjoint subsets $E_1, E_2 \subset E$, each of size at least $c_k m^2/n^2$, such that every edge in E_1 crosses all edges in E_2.*

We have $k = 1$ in straight-line drawings, $k = (\ell + 1)^2$ if every edge is a polyline with up to ℓ *bend*s, and $k = d^2$ if the edges are sufficiently generic algebraic curves (e.g., splines) of degree at most d. Note also that every graph G has a drawing with cr(G) crossings in which any two edges cross at most once [V05].

The dependence on k in Theorem 1 is necessary: We show that one cannot expect bipartite crossing families of edges of size $\Omega(m^2/n^2)$ if any two edges may cross arbitrarily many times, even if the graph drawings are restricted to be x-monotone. An *x-monotone curve* is a continuous arc that intersects every vertical line in at most one point. A drawing of a graph is *x-monotone* if every edge is mapped to an x-monotone curve.

Theorem 2. *For every $n, m \in \mathbb{N}$ with $m \leq n^2/4$, there is a bipartite graph $G = (V, E)$ with n vertices, m edges, and an x-monotone drawing such that any two disjoint subsets $E_1, E_2 \subset E$ of equal size $|E_1| = |E_2| = t$, where every edge in E_1 crosses all edges in E_2, satisfy*

$$t = O\left(\frac{m^2}{n^2 \log(m/n)}\right).$$

We present the tools used for the bipartite strengthening of the Crossing Lemma in the next section. Theorem 1 is proved in Section 3. Our construction of x-monotone drawings are discussed in Section 4. Finally, Section 5 contains a weaker analogue of Theorem 1 for x-monotone drawings and a further strengthening of the Crossing Lemma for graphs satisfying some monotone property.

2 Tools

The proof of Theorem 1 relies on a recent result on the intersection pattern of k-intersecting curves. For a collection C of curves in the plane, the *intersection graph* is defined on the vertex set C, two elements of C are *adjacent* if the (relative) interiors of the corresponding curves intersect. A complete bipartite graph is *balanced* if the vertex classes differ in size by at most one. For brevity, we call a balanced complete bipartite graph a *bi-clique*.

Theorem 3. [FPT07a] *Given m curves in the plane such that at least εm^2 pairs intersect and any two curves intersect in at most k points, their intersection graph contains a bi-clique with at least $c_k \varepsilon^{64} m$ vertices where $c_k > 0$ depends only on k.*

If follows from the Crossing Lemma that in every drawing of a dense graph, the intersection graph of the edges is also dense. Therefore, Theorem 3 implies Theorem 1 in the special case that G is dense. This connection was first observed by Pach and Solymosi [PS01] who proved Theorem 1 for straight-line drawings of dense graphs.

If a graph G is *not* dense, we decompose G recursively into induced subgraphs with an algorithm reminiscent of [PST00] until one of the components is dense enough so that Theorem 3, like before, implies Theorem 1. The decomposition algorithm successively removes *bisectors*, and we use Theorem 4 below to keep the total number of deleted edges under control.

The *bisection width*, denoted by $b(G)$, is defined for every simple graph G with at least two vertices. It 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 $|E(V_1, V_2)| = b(G)$. Pach, Shahrokhi, and Szegedy [PSS96] gave an upper bound on the bisection width in terms of the crossing number and the L_2-norm of the degree vector (it is an easy consequence of the weighted version of the famous Lipton-Tarjan separator theorem [LT79, GM90]).

Theorem 4. [PSS96] *Let G be a graph with n vertices of degree $d_1, d_2, \ldots, d_n$. Then*

$$b(G) \leq 10\sqrt{\mathrm{cr}(G)} + 2\sqrt{\sum_{i=1}^{n} d_i^2(G)}. \tag{2}$$

3 Proof of Theorem 1

Let $G = (V, E)$ be a graph with n vertices and $m \geq 3n$ edges. Since a graph with more than $3n - 6$ edges cannot be planar, it must have crossing edges. Hence, as

long as $3n \le m < 10^6 n$, Theorem 1 holds with $|E_1| = |E_2| = 1 \ge 10^{-12} m^2/n^2$. We assume $m \ge 10^6 n$ in the remainder of the proof.

Let D be a drawing of G. To use the full strength of Theorem 4, we transform the drawing D into a drawing D' of a graph $G' = (V', E')$ with m edges, at most $2n$ vertices, and maximum degree at most $\lceil 2m/n \rceil$, so that the intersection graph of E' is isomorphic to that of E. If the degree of a vertex $v \in V$ is above the average degree $\bar{d} = 2m/n$, split v into $\lceil d/\bar{d} \rceil$ vertices $v_1, \ldots, v_{\lceil d/\bar{d} \rceil}$ arranged along a circle of small radius centered at v. Denote the edges of G incident to v by $(v, w_1), \ldots, (v, w_d)$ in clockwise order in the drawing D. In G', connect w_j with v_i if and only if $\bar{d}(i-1) < j \le \bar{d}i$, where $1 \le j \le d$ and $1 \le i \le \lceil d/\bar{d} \rceil$. Two edges of G' cross if and only if the corresponding edges of G cross. Also, letting $d(v)$ denote the degree of vertex v in G', the number of vertices of G' is

$$\sum_{v \in V} \lceil d(v)/\bar{d} \rceil < \sum_{v \in V} 1 + d(v)/\bar{d} = 2n.$$

Hence the resulting G' and D' have all the required properties.

We will decompose G' recursively into induced subgraphs until each induced subgraph is either a singleton or it has so many pairs of crossing edges that Theorem 3 already implies Theorem 1. Theorem 3 implies that the intersection graph of the edges of an induced subgraph H of G' contains a bi-clique of size at least $c_k \left(\frac{p(H)}{e(H)^2} \right)^{64} e(H)$, where $p(H)$ is the number of pairs of crossing edges in H in the drawing D', $e(H)$ is the number of edges of H, and $c_k > 0$ is the constant depending on k only in Theorem 3. So the intersection graph of the edge set of G' (and hence also of G) contains a bi-clique of size $\Omega_k(m^2/n^2)$ if there is an induced subgraph H of G' with

$$\varepsilon_k \frac{m^2}{n^2} \le \left(\frac{p(H)}{e(H)^2} \right)^{64} e(H), \tag{3}$$

where $\varepsilon_k > 0$ is any constant depending on k only. We use $\varepsilon_k = (10^9 k)^{-64}$ for convenience. Assume, to the contrary, that (3) does not hold for any induced subgraph H of G'.

Every induced subgraph H has at most $kp(H)$ crossings in the drawing D', hence $\mathrm{cr}(H) \le kp(H)$. It is enough to find an induced subgraph H for which

$$\frac{e(H)^{2-1/64}}{10^9} \left(\frac{m}{n} \right)^{\frac{1}{32}} \le \mathrm{cr}(H), \tag{4}$$

since this combined with $\mathrm{cr}(H) \le kp(H)$ implies (3).

Next, we decompose the graph G' of at most $2n$ vertices and m edges with the following algorithm.

Decomposition Algorithm

1. Let $S_0 = \{G'\}$ and $i = 0$.
2. While $(3/2)^i \le 4n^2/m$ and no $H \in S_i$ that satisfies (4), do
   Set $i := i + 1$. Let $S_i := \emptyset$. For every $H \in S_{i-1}$, do

- If $|V(H)| \le (2/3)^i\, 2n$, then let $S_i := S_i \cup \{H\}$;
- otherwise split H into induced subgraphs H_1 and H_2 along a bisector of size $b(H)$, and let $S_i := S_i \cup \{H_1, H_2\}$.

3. Return S_i.

For every i, every graph $H \in S_i$ satisfying the end condition has at most $|V(H)| \le (2/3)^i\, 2n$ vertices. Hence, the algorithm terminates in $t \le \log_{(3/2)} 2n$ rounds and it returns a set S_t of induced subgraphs. Let $T_i \subset S_i$ be the set of those graphs in S_i that have more than $(2/3)^i\, 2n$ vertices. Notice that $|T_i| \le (3/2)^i$. Denote by G_i the disjoint union of the induced subgraphs in S_i.

We use Theorem 4 for estimating the number of edges deleted throughout the decomposition algorithm. Substituting the upper bound for $\mathrm{cr}(H)$ and using Jensen's inequality for the concave function $f(x) = x^{1-1/128}$, we have for every $i = 0, 1, \ldots, t$,

$$\sum_{H\in T_i} \sqrt{\mathrm{cr}(H)} \le \sum_{H\in T_i} \sqrt{\frac{e(H)^{2-1/64}}{10^9}\left(\frac{m}{n}\right)^{\frac{1}{32}}} = 10^{-\frac{9}{2}}\left(\frac{m}{n}\right)^{\frac{1}{64}} \sum_{H\in T_i} e(H)^{1-\frac{1}{128}}$$
$$\le 10^{-\frac{9}{2}}\left(\frac{m}{n}\right)^{\frac{1}{64}} |T_i|^{\frac{1}{128}} m^{1-\frac{1}{128}} \le 10^{-\frac{9}{2}}\left(\frac{3}{2}\right)^{\frac{i}{128}} \frac{m^{1+1/128}}{n^{1/64}}.$$

Denoting by $d(v, H)$ the degree of vertex v in an induced subgraph H, we have

$$\sum_{H\in T_i} \sqrt{\sum_{v\in V(H)} d^2(v,H)} \le \sqrt{|T_i|}\sqrt{\sum_{v\in V(G_i)} d^2(v,G_i)}$$
$$\le \sqrt{(3/2)^i}\sqrt{n\cdot(\bar{d})^2} \le \frac{2m}{\sqrt{n}}\sqrt{(3/2)^i}.$$

In the first of the two above inequalities, we use the Cauchy-Schwartz inequality to get $\sum_{H\in T_i}\sqrt{x_H} \le \sqrt{|T_i|}\sqrt{\sum_{H\in T_i} x_H}$ with $x_H = \sum_{v\in V(H)} d^2(v,H)$.

By Theorem 4, the total number of edges deleted during this process is

$$\sum_{i=0}^{t-1}\sum_{H\in T_i} b(H) \le 10\sum_{i=0}^{t-1}\sum_{H\in T_i}\sqrt{\mathrm{cr}(H)} + 2\sum_{i=0}^{t-1}\sum_{H\in T_i}\sqrt{\sum_{v\in V(H)} d^2(v,H)}$$
$$\le 10^{-\frac{7}{2}}\frac{m^{1+1/128}}{n^{1/64}}\sum_{i=0}^{t-1}(3/2)^{\frac{i}{128}} + 4\frac{m}{\sqrt{n}}\sum_{i=0}^{t-1}\sqrt{(3/2)^i}$$
$$\le \frac{m^{1+1/128}}{4n^{1/64}}\left(\frac{n^2}{m}\right)^{1/128} + 100m^{1/2}n^{1/2} \le \frac{m}{2}.$$

The second inequality uses the earlier upper bounds for $\sum_{H\in T_i}\sqrt{\mathrm{cr}(H)}$ and $\sum_{H\in T_i}\sqrt{\sum_{v\in V(H)} d^2(v,H)}$, the third inequality uses the geometric series formula and the upper bound $t \le \log_{(3/2)} 2n$, while the last inequality follows from the fact that $m \ge 10^6 n$.

So at least $m/2$ edges survive and each of the induced subgraphs in S_t has at most $(2/3)^t\ 2n \le 2n/(4n^2/m) = m/2n$ vertices. Also G' has at most $2n$ vertices, so using Jensen's inequality for the convex function $g(x) = \binom{x}{2}$, the total number of vertex pairs lying in a same induced subgraph of S_t is less than

$$\frac{2n}{m/2n}\frac{(m/2n)^2}{2} = \frac{m}{2},$$

a contradiction. We conclude that the decomposition algorithm must have found an induced subgraph H satisfying (4). This completes the proof of Theorem 1. □

4 Drawings with Edges as x-monotone Curves

It is known that Theorem 3 does not hold without the assumption that any two curves intersect in at most a constant number of points. Using a construction from [F06], Pach and G. Tóth [PT06] constructed for every $n \in \mathbb{N}$, a collection of n x-monotone curves whose intersection graph is dense but every bi-clique it contains has at most $O(n/\log n)$ vertices. Theorem 2 shows a stronger construction holds: the curves are edges in an x-monotone drawing of a dense bipartite graph, where $\Theta(n^2)$ curves have only n distinct endpoints.

The proof of Theorem 3 builds on a crucial observation: Golumbic et al. [GRU83] noticed a close connection between intersection graphs of x-monotone curves and partially ordered sets. Consider n continuous functions $f_i : [0,1] \to \mathbb{R}$. The graph of every continuous real function is clearly an x-monotone curve. Define the partial order $\prec$ on the set of functions by $f_i \prec f_j$ if and only if $f_i(x) < f_j(x)$ for all $x \in [0,1]$. Two x-monotone curves intersect if and only if they are incomparable under this partial order $\prec$.

Lemma 1. [GRU83] *The elements of any partially ordered set $(\{1,2,\ldots,n\},\prec)$ can be represented by continuous real functions $f_1, f_2, \ldots, f_n$ defined on the interval $[0,1]$ such that $f_i(x) < f_j(x)$ for every x if and only if $i \prec j$ $(i \neq j)$.*

Proof. Let $(\{1,2,\ldots,n\},\prec)$ be a partial order, and let Π denote the set consisting of all of its extensions $\pi(1) \prec \pi(2) \prec \ldots \prec \pi(n)$ to a total order. Clearly, every element of Π is a permutation of the numbers $1,2,\ldots,n$. Let $\pi_1, \pi_2, \ldots, \pi_t$ be an arbitrary labeling of the elements of Π. Assign distinct points $x_k \in [0,1]$ to each π_k such that $0 = x_1 < x_2 < \ldots < x_t = 1$. For each i $(1 \le i \le n)$, define a continuous, piecewise linear function $f_i(x)$, as follows. For any k $(1 \le k \le t)$, set $f_i(x_k) = \pi_k^{-1}(i)$, and let $f_i(x)$ be linear over each interval $[x_k, x_{k+1}]$.

Obviously, whenever $i \prec j$ for some $i \neq j$, we have that $\pi_k^{-1}(i) \prec \pi_k^{-1}(j)$ for every k, and hence $f_i(x) < f_j(x)$ for all $x \in [0,1]$. On the other hand, if i and j are incomparable under the partial order $\prec$, there are indices k and k' $(1 \le k \ne k' \le m)$ such that $f_i(x_k) < f_j(x_k)$ and $f_i(x_k) > f_j(x_{k'})$, therefore, by continuity, the graphs of f_i and f_j must cross at least once in the interval $(x_k, x_{k'})$. This completes the proof. □

The following lemma is the key for the proof of Theorem 2. It presents a partially ordered set of size n^2 whose incomparability graph contains bi-cliques of size at most $O(n^2/\log n)$, yet it can be represented with x-monotone curves having only $2n$ endpoints.

Lemma 2. *For every $n \in \mathbb{N}$, there is a partially ordered set P with n^2 elements satisfying the following properties*

1. *every bi-clique in the incomparability graph of P has size at most $O(n^2/\log n)$,*
2. *there are equitable partitions $P = P_1 \cup \ldots \cup P_n$ and $P = Q_1 \cup \ldots \cup Q_n$ such that*
   (a) *for each i, there is a linear extension of P where the elements of P_i are consecutive,*
   (b) *there is a linear extension of P where the elements of each Q_j are consecutive, and*
   (c) *for every i and j, we have $|P_i \cap Q_j| = 1$.*

We now prove Theorem 2, pending the proof of Lemma 2. Note that it suffices to prove Theorem 2 in the case $m = n^2/4$, that is, when G is a bi-clique. By deleting some of the edges of this construction, we obtain a construction for every $m \leq n^2/4$, since edge deletions also decrease the intersection graph of the edges. So it is enough to prove the following.

Lemma 3. *There is an x-monotone drawing of $K_{n,n}$ such that every bi-clique in the intersection graph of the edges has size at most $O(n^2/\log n)$.*

Proof. Let P be a poset described in Lemma 2. Represent P with x-monotone curves as in the proof of Lemma 1 such that the last linear extension π_t has property (b) of Lemma 2, that is, the elements of each Q_j are consecutive in π_t.

We transform the n^2 x-monotone curves representing P into an x-monotone drawing of $K_{n,n}$. We introduce two vertex classes, each of size n, as follows. Along the line $x = 1$, the right endpoints of the x-monotone curves in each Q_j are consecutive. Introduce a vertex on $x = 1$ for each Q_j, and make it the common right endpoint of all curves in Q_j by deforming the curves over the interval $(x_{t-1}, 1]$ but keeping their intersection graph intact. These n vertices along the line $x = 1$ form one vertex class of $K_{n,n}$.

For each i, there is a vertical line $x = x_i$ along which the x-monotone curves in P_i are consecutive. Introduce a vertex for each P_i on line $x = x_i$, and make it the common left endpoint of all curves in P_i by deforming the curves over the interval $[x_i, x_{i+1})$ and erasing their portion over the interval $[0, x_i)$. These n vertices, each lying on a line $x = x_i$, form the second vertex class of $K_{n,n}$. After truncating and slightly deforming the n^2 curves representing P, we have constructed an x-monotone drawing of $K_{n,n}$.

Note that the intersection graph of the edges of this drawing of $K_{n,n}$ is a subgraph of the incomparability graph of P, so every bi-clique of the intersection graph of the edges has size at most $O(n^2/\log n)$. □

Proof of Lemma 2. We start out with introducing some notation for directed graphs. For a subset S of vertices in a directed graph G, let $N_+(S)$ denote the set of vertices x in G such that there is a vertex $s \in S$ with an edge (s, x) in G. Similarly, $N_-(S)$ is the set of vertices y in G such that there is a vertex $s \in S$ with an edge (y, s) in G. A directed graph has *path-girth* k if k is the smallest positive integer for which there are vertices x and y having at least two distinct walks of length k from x to y. Equivalently, denoting the adjacency matrix of G by A_G, it has path-girth k if $A_G^1, \ldots, A_G^{k-1}$ are all 0-1 matrices, but the matrix A_G^k has an entry greater than 1.

A *directed graph* $H = (X, E)$ is an ϵ-expander if both $N_+(S)$ and $N_-(S)$ has size at least $(1 + \epsilon)|S|$ for all $S \subset V$ with $1 \leq |S| \leq |V|/2$. An *expander* is a directed graph with constant expansion.

We will use that for every $v \in \mathbb{N}$, there is a constant degree expander with v vertices and path-girth $\Omega(\log v)$. This can be proved by a slight alteration of a random constant degree directed graph. We suppose for the remainder of the proof that $H = (X, E)$ is an ϵ-expander with v vertices, maximal degree at most d, and path-girth greater than $c \log v$, where ϵ, c, and d are fixed positive constants.

For every $a \in \mathbb{N}$, we define a poset $P(a, H)$ with ground set $X \times \{1, 2, \ldots, a\}$, generated by the relations $(j_1, k_1) \prec (j_2, k_2)$ whenever $k_2 = k_1 + 1$ and (j_1, j_2) is an edge of H.

Let $P_0 = P(a, H)$ with $a = \lfloor \min\left(c, (10 \log d)^{-1}\right) \cdot \log v \rfloor$. One can show, by essentially the same argument as in [F06], that the partially ordered set P_0 has the following three properties.

1. P_0 has $a|X| = \Theta(v \log v)$ elements,
2. each element of P_0 is comparable with fewer than $d^a \leq v^{1/10}$ other elements of P_0, and
3. the largest bi-clique in the incomparability graph of P_0 has size at most $O(|X|) = O(v)$.

Since the path-girth of H is greater than a, if $x, y, z, w \in P_0$ satisfy both $w \prec y \prec x$ and $w \prec z \prec x$, then y and z must be comparable. That is, the poset in Figure 1(a) cannot be a subposet of P_0. The poset P required for Lemma 2 will be a linear size subposet of P_0. We next describe the construction of P.

A *chain* is a set of pairwise comparable elements. The maximum chains in P_0 each have size a, having one element from each of $X \times \{i\}$, $i = 1, 2, \ldots, a$. Greedily choose as many disjoint chains of size a as possible from P_0, denote the set of chains by $\mathcal{C} = \{C_1, \ldots, C_w\}$, where w is the number of chains. By the expansion property of H, we have $w = \Theta(|X|) = \Theta(v)$.

We choose greedily disjoint subsets $P_1, \ldots, P_{ha}$ of P, each of which is the union of $h = \Theta\left(\sqrt{v}\right)$ chains of $\mathcal{C}$. Each P_i has the property that, besides the comparable pairs within each of the the h chains, there are no other comparable pairs in P_i. We can choose the h chains of each P_i greedily: after choosing the k^{th} chain in P_i, we have to choose the $(k+1)^{\text{th}}$ chain such that none of its

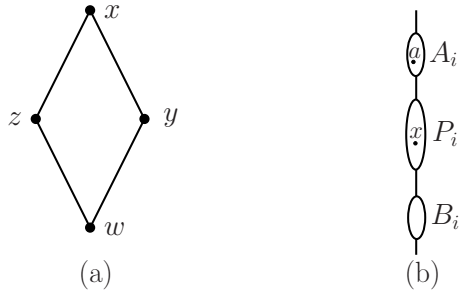


Fig. 1. (a) The Hasse diagram of a four element excluded subposet of P_0. (b) A linear extension of P where $B_i \prec P_i \prec A_i$.

elements are comparable with any element of the first k chains of P_i. Since at most $kav^{1/10} \leq hav^{1/10} = v^{3/5+o(1)}$ of the $w-(i-1)h-k = \Theta(v)$ remaining chains contain an element comparable with the first k chains of P_i, almost any of the remaining chains can be chosen as the $(k+1)^{\text{th}}$ chain of P_i. Finally, let $P = P_1 \cup \ldots \cup P_{ha}$. As mentioned earlier, we have $|P| = \Theta(|P_0|)$, and the largest bi-clique in the incomparability graph of P is of size $O(|P_0|/\log|P_0|) = O(|P|/\log|P|)$.

Since the poset in Fig. 1(a) is not a subposet of P_0, no element of $P_0 \setminus C_k$, $C_k \in \mathcal{C}$, can be both greater than an element of C_k and less than another element of C_k. By construction, if two elements of P_i are comparable, then they belong to the same chain. Therefore, no element of $P \setminus P_i$ can be both greater than an element of P_i and less than another element of P_i.

Consider the partition $P = A_i \cup P_i \cup B_i$, where an element $a \in P \setminus P_i$ is in A_i if and only if there is an element $x \in P_i$ such that $x \prec a$. There is a linear extension of P in which the elements of A_i are the largest, followed by the elements of P_i, and the elements of B_i are the smallest (see Fig. 1(b)). This is because no element of $P \setminus P_i$ can be both greater than an element of P_i and less than another element of P_i.

Partition P into subsets $P = X_1 \cup \ldots \cup X_a$, where X_j consists of the elements $(j,x) \in P$ with $x \in X$. Each X_j contains exactly h^2a elements, h elements from each P_i. Arbitrarily partition each X_j into h sets $X_j = Q_{(j-1)h+1} \cup \ldots \cup Q_{jh}$ such that each Q_k contains one element from each P_i. Since the elements in each X_j form an *antichain* (a set of pairwise incomparable elements), any linear order of the elements of P for which the elements of X_j are smaller than the elements of X_k for $1 \leq j < k \leq a$ is a linear extension of P. Hence, there is a linear extension of P such that, for each j, the elements of every Q_j are consecutive.

We have established that P has all the desired properties. We can choose v such that $n \leq ha$ and $ha = O(n)$, so $v = \Theta(n^2/\log n)$. If ha is not exactly n, we may simply take the subposet whose elements are $(P_1 \cup \ldots \cup P_n) \cap (Q_1 \cup \ldots \cup Q_n)$. This completes the proof of Lemma 2. □

5 Concluding Remarks

We can prove a weaker form of Theorem 1 for x-monotone curves, since our main tools (Theorems 3 and 4) are available in weaker forms in this case. It was recently shown in [FPT07b] that there is a constant $c > 0$ such that the intersection graph G of any n x-monotone curves, at least εn^2 pairs of which intersect, contains a bi-clique with at least $c\varepsilon^2 n/(\log \frac{1}{\varepsilon} \log n)$ vertices. The Crossing Lemma implies that the intersection graph of the edges of a dense topological graph is dense, so we have the following corollary.

Corollary 1. *For every x-monotone drawing of a graph $G = (V, E)$ with n vertices and $m = \Omega(n^2)$ edges, there are disjoint subsets $E_1, E_2 \subset E$, each of size at least $\Omega(n^2/\log n)$, such that every edge in E_1 crosses all edges in E_2.*

Corollary 1 is tight up to a constant factor by Theorem 2. Similar to Theorem 4, Kolman and Matoušek [KM04] proved an upper bound on the bisection width in terms of the *pairwise* crossing number and the L_2 norm of the degree sequence $d_1, d_2, \ldots, d_n$:

$$b(G) = O\left(\left(\sqrt{\text{pair-cr}(G)} + \sqrt{\sum_{i=1}^{n} d_i^2(G)}\right) \log n\right).$$

Using the same strategy as in the proof of Theorem 1, with the above mentioned tools instead of Theorems 3 and 4, it is straightforward to establish the following.

Theorem 5. *For every x-monotone drawing of a graph $G = (V, E)$ with n vertices and $m \geq 3n$ edges, there are disjoint subsets $E_1, E_2 \subset E$, each of cardinality at least $m^2/(n^2 \log^{5+o(1)} n)$, such that every edge in E_1 crosses every edge in E_2.*

In a special case, we can prove the same bound as in Theorem 1.

Proposition 1. *Given a bipartite graph G with n vertices and $m \geq 3n$ edges, and an x-monotone drawing where the vertices of the two vertex classes lie on the lines $x = 0$ and $x = 1$, respectively, then the intersection graph of the edges contains a bi-clique of size $\Omega(m^2/n^2)$.*

Proof. Consider the two dimensional partial order $\prec$ on the edges of G, where an edge e_1 is greater than another edge e_2 if and only if, for $j = 0, 1$ the endpoint of e_1 on the line $x = j$ lies above that of e_2. Two edges of G must cross if they are incomparable by the partial order $\prec$. Also notice that there is an x-monotone drawing of G with the vertices in the same position where two edges of G cross if and only if they are incomparable under $\prec$. Indeed, this is done by drawing the edges as straight line segments.

By the Crossing Lemma, there are at least $\Omega(m^3/n^2)$ pairs of crossing edges in this straight-line drawing of G. Hence, there are at least $\Omega(m^3/n^2)$ pairs of incomparable elements under the partial order $\prec$. In [FPT07b] (Theorem 3), we prove that any incomparability graph with m vertices and at least dm edges

contains a bi-clique of size at least d, so the intersection graph of the edges of G must contain a bi-clique of size $\Omega(m^2/n^2)$. □

Proposition 1 implies that Theorem 1 holds for x-monotone drawings if the vertex set lies in a bounded number of vertical lines. Indeed, an x-monotone drawing of a graph with all vertices contained in the union of d vertical lines can be partitioned into $\binom{d}{2}$ x-monotone drawings of bipartite graphs with each vertex class lying on a vertical line.

Monotone properties. If a graph is drawn with at most k crossings between any two edges and the graph has some additional property, then one may improve on the bound of Theorem 1.

A graph property $\mathcal{P}$ is *monotone* if whenever a graph G satisfies $\mathcal{P}$, every subgraph of G also satisfies $\mathcal{P}$, and whenever graphs G_1 and G_2 satisfy $\mathcal{P}$, then their disjoint union also satisfies $\mathcal{P}$. The *extremal number* $\mathrm{ex}(n, \mathcal{P})$ denotes the maximum number of edges that a graph with property $\mathcal{P}$ on n vertices can have. For graphs satisfying a monotone graph property, the bound (1) of the Crossing Lemma can be improved [PST00]. In particular, if $\mathcal{P}$ is a monotone graph property and $\mathrm{ex}(n, \mathcal{P}) = O(n^{1+\alpha})$ for some $\alpha > 0$, then there exist constants $c, c' > 0$ such that for every graph G with n vertices, $m \geq cn \log^2 n$ edges, and property $\mathcal{P}$, the crossing number is at least $\mathrm{cr}(G) \geq c' m^{2+1/\alpha}/n^{1+1/\alpha}$. Furthermore, if $\mathrm{ex}(n, \mathcal{P}) = \Theta(n^{1+\alpha})$, then this bound is tight up to a constant factor. A straightforward calculation shows, using the same strategy as in the previous section, the following strengthening of Theorem 1.

Theorem 6. *Let $\mathcal{P}$ be a monotone graph property such that $ex(n, \mathcal{P}) = O(n^{1+\alpha})$ for some $\alpha > 0$. For every $k \in \mathbb{N}$, there exist positive constants c and c_k such that for any drawing of a graph $G = (V, E)$ satisfying property $\mathcal{P}$, having n vertices and $m \geq cn \log^2 n$ edges, no two of which intersecting in more than k points, there are disjoint subsets $E_1, E_2 \subset E$, each of cardinality at least $c_k(m/n)^{1+1/\alpha}$, such that every edge in E_1 crosses all edges in E_2.*

Acknowledgment. We thank the anonymous referee for bringing a paper by Golumbic, Rotem, and Urrutia [GRU83] to our attention.

References

[ACNS82] Ajtai, M., Chvátal, V., Newborn, M., Szemerédi, E.: Crossing-free subgraphs. In: Theory and Practice of Combinatorics. Mathematical Studies, vol. 60, pp. 9–12. North-Holland, Amsterdam (1982)

[D98] Dey, T.K.: Improved bounds for planar k-sets and related problems. Discrete Comput. Geom. 19, 373–382 (1998)

[ENR00] Elekes, G., Nathanson, M.B., Ruzsa, I.Z.: Convexity and sumsets. J. Number Theory 83, 194–201 (2000)

[E46] Erdős, P.: On sets of distances of n points, Amer. Math. Monthly 53, 248–250 (1946)

[F06] Fox, J.: A bipartite analogue of Dilworth's theorem, Order 23, 197–209 (2006)

[FPT07a] Fox, J., Pach, J., Tóth, C.D.: Intersection patterns of curves, manuscript, Cf (2007), http://math.nyu.edu/~pach/publications/justcurves050907.pdf

[FPT07b] Fox, J., Pach, J., Tóth, C.D.: Turán-type results for partial orders and intersection graphs of convex sets, Israel J. Math. (to appear, 2007)

[GRU83] Golumbic, M.C., Rotem, D., Urrutia, J.: Comparability graphs and intersection graphs. Discrete Math. 43, 37–46 (1983)

[GM90] Gazit, H., Miller, G.L.: Planar separators and the Euclidean norm. In: Asano, T., Imai, H., Ibaraki, T., Nishizeki, T. (eds.) SIGAL 1990. LNCS, vol. 450, pp. 338–347. Springer, Heidelberg (1990)

[KT04] Katz, N.H., Tardos, G.: A new entropy inequality for the Erdős distance problem. In: Towards a theory of geometric graphs. Contemp. Math. AMS, vol. 342, pp. 119–126 (2004)

[KM04] Kolman, P., Matoušek, J.: Crossing number, pair-crossing number, and expansion. J. Combin. Theory Ser. B 92, 99–113 (2004)

[L84] Leighton, T.: New lower bound techniques for VLSI. Math. Systems Theory 17, 47–70 (1984)

[LT79] Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. 36, 177–189 (1979)

[LPS88] Lubotzky, A., Phillips, R., Sarnak, P.: Ramanujan graphs. Combinatorica 8, 261–277 (1988)

[PRTT06] Pach, J., Radoičić, R., Tardos, G., Tóth, G.: Improving the Crossing Lemma by finding more crossings in sparse graphs. Discrete Comput. Geom. 36, 527–552 (2006)

[PSS96] Pach, J., Shahrokhi, F., Szegedy, M.: Applications of the crossing number. Algorithmica 16, 111–117 (1996)

[PS98] Pach, J., Sharir, M.: On the number of incidences between points and curves. Combin. Probab. Comput. 7(1), 121–127 (1998)

[PS01] Pach, J., Solymosi, J.: Crossing patterns of segments. J. Combin. Theory Ser. A 96, 316–325 (2001)

[PST00] Pach, J., Spencer, J., Tóth, G.: New bounds on crossing numbers. Discrete Comput. Geom. 24(4), 623–644 (2000)

[PT00] Pach, J., Tóth, G.: Which crossing number is it anyway? J. Combin. Theory Ser. B 80, 225–246 (2000)

[PT02] Pach, J., Tardos, G.: Isosceles triangles determined by a planar point set. Graphs Combin. 18, 769–779 (2002)

[PT06] Pach, J., Tóth, G.: Comment on Fox News. Geombinatorics 15, 150–154 (2006)

[ST01] Solymosi, J., Tóth, C.D.: Distinct distances in the plane. Discrete Comput. Geom. 25, 629–634 (2001)

[STT02] Solymosi, J., Tardos, G., Tóth, C.D.: The k most frequent distances in the plane. Discrete Comput. Geom. 28, 639–648 (2002)

[S97] Székely, L.A.: Crossing numbers and hard Erdős problems in discrete geometry. Combin. Probab. Comput. 6, 353–358 (1997)

[ST83] Szemerédi, E., Trotter, W.T.: Extremal problems in discrete geometry. Combinatorica 3, 381–392 (1983)

[TV06] Tao, T., Vu, V.: Additive combinatorics. Cambridge Studies in Advanced Mathematics, 105. Cambridge University Press, Cambridge (2006)

[V05] Valtr, P.: On the pair-crossing number. In: Combinatorial and computational geometry, vol. 52, pp. 569–575. MSRI Publications, Cambridge Univ. Press (2005)

Improvement on the Decay of Crossing Numbers*

Jakub Černý[1], Jan Kynčl[1], and Géza Tóth[2]

[1] Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
kuba@kam.mff.cuni.cz, kyncl@kam.mff.cuni.cz
[2] Rényi Institute, Hungarian Academy of Sciences, Reáltanoda utca 13-15, H-1053, Budapest, Hungary
geza@renyi.hu

Abstract. We prove that the crossing number of a graph decays in a "continuous fashion" in the following sense. For any $\varepsilon > 0$ there is a $\delta > 0$ such that for n sufficiently large, every graph G with n vertices and $m \geq n^{1+\varepsilon}$ edges has a subgraph G' of at most $(1-\delta)m$ edges and crossing number at least $(1-\varepsilon)\text{CR}(G)$. This generalizes the result of J. Fox and Cs. Tóth.

1 Introduction

For any graph G, let $n(G)$ (resp. $m(G)$) denote the number of its vertices (resp. edges). If it is clear from the context, we simply write n and m instead of $n(G)$ and $m(G)$. The crossing number $\text{CR}(G)$ of a graph G is the minimum number of edge crossings over all drawings of G in the plane. In the optimal drawing of G, crossings are not necessarily distributed uniformly among the edges. Some edges could be more "responsible" for the crossing number than some other edges. For any fixed k, it is not hard to construct a graph G whose crossing number is k, but G has an edge e such that $G \setminus e$ is planar. Richter and Thomassen [RT93] started to investigate the following general problem. We have a graph G, and we want to remove a given number of edges. By *at least* how much does the crossing number decrease? They conjectured that there is a constant c such that every graph G with $\text{CR}(G) = k$ has an edge e with $\text{CR}(G \setminus e) \geq k - c\sqrt{k}$. They only proved that G has an edge with $\text{CR}(G \setminus e) \geq \frac{2}{5}\text{CR}(G) - O(1)$.

Pach, Radoičić, Tardos, and Tóth [PRTT06] proved that for every graph G with $m(G) \geq \frac{103}{16}n(G)$, we have $\text{CR}(G) \geq 0.032\frac{m^3}{n^2}$. It is not hard to see [PT00] that for *any* edge e, we have $\text{CR}(G-e) \geq \text{CR}(G) - m + 1$. These two results imply

* Jakub Černý and Jan Kynčl have been supported by the Phenomena in High Dimensions project, in the framework of the European Community's "Structuring the European Research Area" programme, Grant MRTN-CT-2004-511953 at Rényi Institute and by project 1M0021620808 (ITI) of Ministry of Education of the Czech Republic; Géza Tóth has been supported by the Hungarian Research Fund grants OTKA-T-038397, OTKA-T-046246, and OTKA-K-60427.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 25–30, 2007.

an improvement of Richter–Thomassen bound if $m \geq 8.1n$, and also imply the Richter–Thomassen conjecture for graphs of $\Omega(n^2)$ edges.

J. Fox and Cs. Tóth [FT06] investigated the case where we want to delete a *positive fraction* of the edges.

Theorem A. [FT06] *For every $\varepsilon > 0$, there is an n_ε such that every graph G with $n(G) \geq n_\varepsilon$ vertices and $m(G) \geq n(G)^{1+\varepsilon}$ edges has a subgraph G' with*

$$m(G') \leq \left(1 - \frac{\varepsilon}{24}\right) m(G)$$

and

$$\mathrm{cr}(G') \geq \left(\frac{1}{28} - o(1)\right) \mathrm{cr}(G).$$

In this note we generalize Theorem A.

Theorem. *For every $\varepsilon, \gamma > 0$, there is an $n_{\varepsilon,\gamma}$ such that every graph G with $n(G) \geq n_{\varepsilon,\gamma}$ vertices and $m(G) \geq n(G)^{1+\varepsilon}$ edges has a subgraph G' with*

$$m(G') \leq \left(1 - \frac{\varepsilon\gamma}{2394}\right) m(G)$$

and

$$\mathrm{cr}(G') \geq (1-\gamma)\mathrm{cr}(G).$$

2 Proof of the Theorem

Our proof is based on the argument of Fox and Tóth [FT06], the only new ingredient is Lemma 1.

Definition. Let $r \geq 2, p \geq 1$ be integers. A *$2r$-earring of size p* is a graph which is a union of an edge uv and p edge-disjoint paths between u and v, each of length at most $2r-1$. Edge uv is called the *main edge* of the $2r$-earring.

Lemma 1. *Let $r \geq 2, p \geq 1$ be integers. There exists n_0 such that every graph G with $n \geq n_0$ vertices and $m \geq 6prn^{1+1/r}$ edges contains at least $m/3pr$ edge-disjoint $2r$-earrings, each of size p.*

Proof. By the result of Alon, Hoory, and Linial [AHL02], for some n_0, every graph with $n \geq n_0$ vertices and at least $n^{1+1/r}$ edges contains a cycle of length at most $2r$.

Suppose that G has $n \geq n_0$ vertices and $m \geq 6prn^{1+1/r}$ edges. Take a *maximal* edge-disjoint set $\{E_1, E_2, \ldots, E_x\}$ of $2r$-earrings, each of size p. Let $E = E_1 \cup E_2 \cup \cdots \cup E_x$, the set of all edges of the earrings and let $G' = G \setminus E$. Now let E'_1 be a $2r$-earring of G' of maximum size. Note that this size is less than p. Let $G'_1 = G' \setminus E'_1$. Similarly, let E'_2 be a $2r$-earring of G'_1 of maximum size and let $G'_2 = G'_1 \setminus E'_2$. Continue analogously, as long as there is a $2r$-earring in the

remaining graph. We obtain the $2r$-earrings $E'_1, E'_2, \ldots, E'_y$, and the remaining graph $G'' = G'_y$ does not contain any $2r$-earring. Let $E' = E'_1 \cup E'_2 \cup \cdots \cup E'_y$.

We claim that $y < n^{1+1/r}$. Suppose on the contrary that $y \geq n^{1+1/r}$. Take the main edges of $E'_1, E'_2, \ldots, E'_y$. We have at least $n^{1+1/r}$ edges so by the result of Alon, Hoory, and Linial [AHL02] some of them form a cycle C of length at most $2r$. Let i be the smallest index with the property that C contains the main edge of E'_i. Then C, together with E'_i would be a $2r$-earring of G'_{i-1} of greater size than E'_i, contradicting the maximality of E'_i.

Each of the earrings $E'_1, E'_2, \ldots, E'_y$ has at most $(p-1)(2r-1)+1$ edges so we have $|E'| \leq y(p-1)(2r-1) + y < (2pr-1)n^{1+1/r}$. The remaining graph, G'' does not contain any $2r$-earring, in particular, it does not contain any cycle of length at most $2r$, since it is a $2r$-earring of size one. Therefore, by [AHL02], for the number of its edges we have $e(G'') < n^{1+1/r}$.

It follows that the set $E = \{E_1, E_2, \ldots, E_x\}$ contains at least $m - 2prn^{1+1/r} \geq \frac{2}{3}m$ edges. Each of $E_1, E_2, \ldots, E_x$ has at most $p(2r-1)+1 \leq 2pr$ edges, therefore, $x \geq m/3pr$. □

Lemma 2. [FT06] *Let G be a graph with n vertices, m edges, and degree sequence $d_1 \leq d_2 \leq \cdots \leq d_n$. Let ℓ be the integer such that $\sum_{i=1}^{\ell-1} d_i < 4m/3$ but $\sum_{i=1}^{\ell} d_i \geq 4m/3$. If n is large enough and $m = \Omega(n \log^2 n)$ then*

$$\mathrm{CR}(G) \geq \frac{1}{65} \sum_{i=1}^{\ell} d_i^2.$$

Proof of the Theorem. Let $\varepsilon, \gamma \in (0,1)$ be fixed. Choose integers r, p such that $\frac{1}{r} < \varepsilon \leq \frac{2}{r}$ and $\frac{132}{p} < \gamma \leq \frac{133}{p}$. Then there is an $n_{\varepsilon,\gamma}$ with the following properties: (a) $n_{\varepsilon,\gamma} \geq n_0$ from Lemma 1, (b) $(n_{\varepsilon,\gamma})^{1+\varepsilon} > 6pr \cdot (n_{\varepsilon,\gamma})^{1+1/r}$,

Let G be a graph with $n \geq n_{\varepsilon,\gamma}$ vertices and $m \geq n^{1+\varepsilon}$ edges.

Let $v_1, \ldots, v_n$ be the vertices of G, of degrees $d_1 \leq d_2 \leq \cdots \leq d_n$ and define ℓ as in Lemma 2, that is, $\sum_{i=1}^{\ell-1} d_i < 4m/3$ but $\sum_{i=1}^{\ell} d_i \geq 4m/3$. Let G_0 be the subgraph of G induced by $v_1, \ldots, v_\ell$. Observe that G_0 has at least $m/3$ edges. Therefore, by Lemma 1 G_0 contains at least $m/9pr$ edge-disjoint $2r$-earrings, each of size p.

Let M be the set of the main edges of these $2r$-earrings. We have $|M| \geq m/9pr \geq \frac{\varepsilon\gamma}{2394}m$. Let $G' = G \setminus M$ and $G'_0 = G_0 \setminus M$.

Take an optimal drawing $D(G')$ of the subgraph $G' \subset G$. We have to draw the missing edges to obtain a drawing of G. Our method is a randomized variation of the embedding method, which has been applied by Leighton [L83], Richter and Thomassen [RT93], Shahrokhi et al. [SSSV97], Székely [S04], and most recently by Fox and Tóth [FT06]. For every missing edge $e_i = u_iv_i \in M \subset G_0$, e_i is the deleted main edge of a $2r$-earring $E_i \subset G_0$. So there are p vertex-disjoint paths in G_0 from u_i to v_i. For each of these paths, draw a curve from u_i to v_i infinitesimally close to that path. Call these p curves *potential u_iv_i–edges* and call the resulting drawing D.

To get a drawing of G, for each $e_i = u_i v_i \in M$, choose one of the p potential $u_i v_i$–edges at random, independently and uniformly, with probability $1/p$, and draw the edge $u_i v_i$ as that curve.

There are two types of new crossings in the obtained drawing of G. First category crossings are infinitesimally close to a crossing in $D(G')$, second category crossings are infinitesimally close to a vertex of G_0 in $D(G')$.

The expected number of first category crossings is at most

$$\left(1 + \frac{2}{p} + \frac{1}{p^2}\right) \mathrm{CR}(G') = \left(1 + \frac{1}{p}\right)^2 \mathrm{CR}(G').$$

Indeed, for each edge of G', there can be at most one new edge drawn next to it, and that is drawn with probability at most $1/p$. Therefore, in the close neighborhood of a crossing in $D(G')$, the expected number of crossings is at most $(1 + \frac{2}{p} + \frac{1}{p^2})$. See figure 1(a).

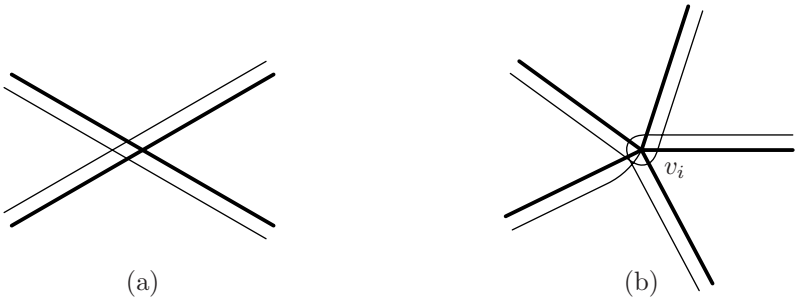


Fig. 1. The thick edges are edges of G', the thin edges are the potential edges. Figure shows (a) a neighborhood of a crossing in $D(G')$ and (b) a neighborhood of a vertex v_i in G'.

In order to estimate the expected number of second category crossings, consider the drawing D near a vertex v_i of G_0. In the neighborhood of vertex v_i we have at most d_i original edges. Since we draw at most one potential edge along each original edge, there can be at most d_i potential edges in the neighborhood. Each potential edge can cross each original edge at most once, and any two potential edges can cross at most twice. See figure 1(b). Therefore, the total number of first category crossings in D in the neighborhood of v_i is at most $2d_i^2$. (This bound can be substantially improved with a more careful argument, see e. g. [FT06], but we do not need anything better here.) To obtain the drawing of G, we keep each of the potential edges with probability $1/p$, so the expected number of crossings in the neighborhood of v_i is at most $\frac{1}{p} 2d_i^2$.

Therefore, the total expected number of crossings in the random drawing of G is at most $(1 + \frac{2}{p} + \frac{1}{p^2})\mathrm{CR}(G') + \frac{2}{p} \sum_{i=1}^{\ell} d_i^2$.

There exists an embedding with at most this many crossings, therefore, by Lemma 2 we have

$$\mathrm{CR}(G) \leq \left(1+\frac{1}{p}\right)^2 \mathrm{CR}(G') + \frac{2}{p}\sum_{i=1}^{\ell} d_i^2 \leq \left(1+\frac{1}{p}\right)^2 \mathrm{CR}(G') + \frac{130}{p}\mathrm{CR}(G).$$

It follows that

$$\left(1-\frac{130}{p}\right)\mathrm{CR}(G) \leq \left(1+\frac{1}{p}\right)^2 \mathrm{CR}(G')$$

so

$$\left(1-\frac{130}{p}\right)\left(1-\frac{1}{p}\right)^2 \mathrm{CR}(G) \leq \mathrm{CR}(G')$$

consequently

$$\mathrm{CR}(G') \geq \left(1-\frac{132}{p}\right)\mathrm{CR}(G) \geq (1-\gamma)\mathrm{CR}(G). \qquad \square$$

Remark. In the statement of our Theorem we cannot require that *every* subgraph G' with $(1-\delta)m(G)$ edges has crossing number $\mathrm{CR}(G') \geq (1-\gamma)\mathrm{CR}(G)$, instead of just *one* such subgraph G'. In fact, the following statement holds.

Proposition. *For every $\varepsilon \in (0,1)$ there exist graphs G_n with $n(G_n) = \Theta(n)$ vertices and $m(G_n) = \Theta(n^{1+\varepsilon})$ edges with subgraphs $G'_n \subset G_n$ such that*

$$m(G'_n) = (1-o(1))\, m(G_n)$$

and

$$\mathrm{CR}(G'_n) = o(\mathrm{CR}(G_n)).$$

Proof. Roughly speaking, G_n will be the disjoint union of a large graph G'_n with low crossing number and a small graph H_n with large crossing number. More precisely, let $G = G_n$ be a disjoint union of graphs $G' = G'_n$ and $H = H_n$, where G' is a disjoint union of $\Theta(n^{1-\varepsilon})$ complete graphs, each with n^{ε} vertices and H is a complete graph with $n^{(3+5\varepsilon)/8}$ vertices. We have $m(G) = \Theta(n^{1+\varepsilon})$ and $m(H) = \Theta(n^{(3+5\varepsilon)/4}) = o(m(G))$, since $\frac{3+5\varepsilon}{4} < 1+\varepsilon$. By the crossing lemma (see e. g. [PRTT06]), $\mathrm{CR}(G) \geq \mathrm{CR}(H) = \Omega(n^{(3+5\varepsilon)/2})$, but $\mathrm{CR}(G') = O(n^{1-\varepsilon}\cdot n^{4\varepsilon}) = O(n^{1+3\varepsilon}) = o(\mathrm{CR}(G))$, because $\frac{3+5\varepsilon}{2} > 1+3\varepsilon$. $\square$

On the other hand, we conjecture that we *can* require that a positive fraction of all subgraphs G' of G with $(1-\delta)m(G)$ edges has crossing number $\mathrm{CR}(G') \geq (1-\gamma)\mathrm{CR}(G)$. The simplest form of our conjecture is the following.

Conjecture. *For every $\varepsilon > 0$, there is an n_ε and δ such that every graph G with $n(G) \geq n_\varepsilon$ vertices and $m(G) \geq n(G)^{1+\varepsilon}$ edges has the following property. Let G' be a random subgraph of of G such that we choose each edge of G independently with probability $p = 1-\delta$. Then*

$$\Pr\left[\mathrm{CR}(G') \geq (1-\varepsilon)\mathrm{CR}(G)\right] > \delta.$$

References

[AHL02] Alon, N., Hoory, S., Linial, N.: The Moore bound for irregular graphs. Graphs and Combinatorics 18, 53–57 (2002)

[FT06] Fox, J., Tóth, Cs.: On the decay of crossing numbers. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 174–183. Springer, Heidelberg (2007)

[L83] Leighton, T.: Complexity issues in VLSI. MIT Press, Cambridge (1983)

[PRTT06] Pach, J., Radoičić, R., Tardos, G., Tóth, G.: Improving the Crossing Lemma by finding more crossings in sparse graphs. In: Proc. 19th ACM Symposium on Computational Geometry, pp. 68–75 (2004), Also in: Discrete and Computational Geometry 36, 527–552 (2006)

[PT00] Pach, J., Tóth, G.: Thirteen problems on crossing numbers. Geombinatorics 9, 194–207 (2000)

[RT93] Richter, B., Thomassen, C.: Minimal graphs with crossing number at least k. J. Combin. Theory Ser. B 58, 217–224 (1993)

[SSSV97] Shahrokhi, F., Sýkora, O., Székely, L., Vrt'o, I.: Crossing numbers: bounds and applications. In: Intuitive geometry, Budapest (1995) Bolyai Soc. Math. Stud. 6, 179–206 (1997)

[S04] Székely, L.: Short proof for a theorem of Pach, Spencer, and Tóth. In: Towards a theory of geometric graphs, Contemporary Mathematics, vol. 342, pp. 281–283. AMS, Providence, RI (2004)

Crossing Numbers and Parameterized Complexity

Michael J. Pelsmajer[1], Marcus Schaefer[2], and Daniel Štefankovič[3]

[1] Illinois Institute of Technology, Chicago, IL 60616, USA
pelsmajer@iit.edu
[2] DePaul University, Chicago, IL 60604, USA
mschaefer@cs.depaul.edu
[3] University of Rochester, Rochester, NY 14627, USA
stefanko@cs.rochester.edu

Abstract. The *odd crossing number* of G is the smallest number of pairs of edges that cross an odd number of times in any drawing of G. We show that there always is a drawing realizing the odd crossing number of G that uses at most 9^k crossings, where k is the odd crossing number of G. As a consequence of this and a result of Grohe we can show that the odd crossing number is fixed-parameter tractable.

1 Introduction

The *crossing number* of a graph G, denoted $\mathrm{cr}(G)$, is the smallest number of intersections in any drawing of G. There are many variants of this fundamental notion; in this paper we concentrate on the *odd crossing number* which counts pairs of edges that intersect an odd number of times. More formally, $\mathrm{ocr}(G)$ is the smallest number of pairs of edges in any drawing of G that cross an odd number of times. Similarly, we can define the *pair crossing number* of G, $\mathrm{pcr}(G)$, as the smallest number of pairs of edges that intersect in any drawing of G. For historical background and summary on different notions of crossing numbers, see the paper by Pach and Tóth [4].

From the definition we have

$$\mathrm{ocr}(G) \leq \mathrm{pcr}(G) \leq \mathrm{cr}(G).$$

We also know that $\mathrm{cr}(G) \leq 2\,\mathrm{ocr}(G)^2$ ([4], for a new proof, see [6]) and $\mathrm{cr}(G) \leq 2\,\mathrm{pcr}(G)^2/\log^2 \mathrm{pcr}(G)$ [9,8]. And while we do know that $\mathrm{ocr}(G) \neq \mathrm{cr}(G)$ in general [5], it is possible that $\mathrm{pcr}(G) = \mathrm{cr}(G)$ for all G.

This suggests the question of how close we can come to realizing this suspected equality in a drawing; that is, what can we say about the number of crossings needed in a pcr-optimal drawing? Maybe surprisingly, the best upper bounds we know are exponential [7] (see the end of Section 3 for a discussion).

To the extent that we believe that $\mathrm{pcr}(G) = \mathrm{cr}(G)$ this is a bit of an embarrassment, since the bound should be the identity. On the other hand, the pair crossing number does tie in very closely with the string graph problem, and a

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 31–36, 2007.

proof that $\mathrm{pcr}(G) = \mathrm{cr}(G)$ based on redrawing would have to change *which* pairs of edges intersect: if we restrict redrawing moves to those that do not change which pairs of edges intersect, there is an exponential separation between pair-crossing number and crossing number due to due to Kratochvíl and Matoušek [3] (they phrased their example for string graphs).

In this paper we address the question of how many crossings are needed to realize an ocr-optimal drawing. We prove an exponential upper bound, similar to what was shown in the case of pcr. It is not inconceivable that the actual gap is exponential; this would be a very interesting result indeed.

Grohe showed that $\mathrm{cr}(G) \leq k$ can be decided in quadratic time for any fixed k [1]. This means that the crossing number problem is *fixed-parameter tractable*: it can be solved in time $O(n^c)$ for some constant c not depending on the parameter k. In Section 3 we show how to combine our exponential upper on crossings in an ocr-optimal drawing with Grohe's proof to conclude that ocr can also be decided in quadratic time. This result is somewhat unsatisfactory in that it relies on Grohe's proof rather than establishing a reduction that would allows us to transfer the fixed-parameter tractability result from cr to ocr automatically (such reductions are known as *fpt-reductions*). If we had such a reduction, Grohe's result could be replaced when a better fixed-parameter algorithm for crossing number is found. Indeed, Grohe's result has very recently been improved from quadratic to linear time by Kawarabayashi and Reed [2]. Kawarabayashi and Reed also claim (albeit without supplying details) that ocr and pcr are fixed-parameter tractable. They do not have a reduction either, but have to verify that their constructions work for ocr and pcr in place of cr. We believe that their missing details can be filled in, for example, by using Theorem 1 and Theorem 3.2 from [7].

One motivation behind the introduction of the crossing number variants pcr and ocr was the hope that they would turn out to be easier objects to deal with than the crossing number itself. For example, the odd crossing number problem can be rephrased as a shortest vector problem in an appropriately chosen vector space. The hope remains that through these alternative approaches we might obtain feasible approximation algorithms or parameterized algorithms solving the crossing number problem (the results by Grohe, Kawarabayashi and Reed do not yield feasible algorithms).

2 ocr-Critical Drawings

In this section we show that a drawing of a graph realizing the odd crossing number has at most an exponential number of crossings.

Theorem 1. *For any graph G there is a drawing of G with odd-crossing number $c = \mathrm{ocr}(G)$ and crossing number at most 9^c.*

The core of the proof is a redrawing idea: consider a drawing of G, and a particular edge e of G. Imagine that e is drawn as a horizontal line segment, and consider an arbitrary subsegment I. Consider the intersections of e with other edges that occur within I. Without changing the odd-crossing number of the

drawing, we can rearrange these intersections within I such that for each edge $f \neq e$, the intersections of f and I are consecutive along I: We can do this by simply pushing intersections to the left or the right. Whenever an intersection of f with e is pushed past an intersection of f' with e, it yields two new intersections between f and f', which does not change the odd-crossing number of the drawing. Next, we claim that we can redraw G such that each edge $f \neq e$ has at most 2 intersections with I, without changing the odd-crossing number of the drawing. Consider every edge f that intersects I, one at a time. Split f at each intersection with I, creating a set of curves S_I with endpoints in I, except that two of the curves have one endpoint at an endpoint of f.

Let α and ω be the two curves in S_I that have one end at an endpoint of f. Let S_α be the set of curves in S_I that begin and end on the same side of I where α ends. Let S'_α be the set of curves in S_I that begin and end on the other side of I, and let S be the set of curves that begin and end at opposite sides of I.

Our goal is to reconnect the parts of f so that the resulting curve traverses all of the original parts of f except on a small neighborhood of I, and intersects I at most twice. We proceed as follows: Start by following α from an endpoint of f to its intersection with I. Continue by following all of the curves in S_α, one after the other, then the curves of S, then the curves of S'_α and end by following ω to the other endpoint of f. Move the endpoints of the curves at I slightly, and connect consecutive curves in a small neighborhood of I such that the resulting curve f' intersects I as few times as possible. (For the moment, we ignore self-intersections of f'.) The only steps at which intersecting I may be unavoidable occur when going from S to S'_α and when going from S'_α to ω. Thus f' redraws f using at most two intersections with I. Observe that the redrawing f' intersects I exactly once if and only if either 1) α and ω approach I from opposite sides and $|S|$ is even, or 2) α and ω approach I from the same side and $|S|$ is odd. Before redrawing, the number of intersections between f and I is $1 + 2|S_\alpha| + |S|$ if α and ω approach I from opposite sides and $2 + 2|S_\alpha| + |S|$ if α and ω approach I from the same side. Thus, the number of intersections between f and I is odd if and only if the number of intersections between f' and I is now one. Also, the parity of intersection of the redrawing f' with any other edge is the same as the parity of f with that edge, since f' and f agree except for in a small neighborhood of I, where f intersects only I.

As we mentioned earlier, the redrawing f' might contain self-intersections, however, these can easily be removed (see [6], for example). Repeating this process for each edge that intersects I results in at most $2i$ intersections of edges with I, where i is the number of edges $f \neq e$ that intersected I an odd number of times before the redrawing.

We now apply this idea to bound the number of crossings necessary to realize a particular odd crossing number.

We begin with a drawing of G achieving $\mathrm{ocr}(G)$. Applying Theorem 2.1 from [6] allows us to assume that all even edges are without intersections. Then there are at most $k := 2\,\mathrm{ocr}(G)$ edges, $e_1, \ldots, e_k$, involved in intersections in the drawing of G under consideration. We will redraw these edges such that

for $1 \leq i < j \leq k$, the number of intersections between e_i and e_j is at most $2(3^{i-1})$. We redraw the edges in order, as follows: Begin by applying the procedure described earlier to e_1; then each other edge intersects e_1 at most twice, as desired. We want to keep the intersections along e_1 now, so we should not apply our procedure to subsequent edges. Instead, during the jth step we split e_j into segments at every intersection with an edge e_i with $i < j$, and apply the procedure to each of those segments.

By induction, the number of intersections of e_j and all e_i with $i < j$ is at most $\sum_{i=1}^{j-1} 2(3^{i-1})$, which equals $3^{j-1} - 1$. Hence e_j is split up into at most 3^{j-1} segments, and after applying the procedure to each segment, each e_i with $i > j$ has at most $2(3^{j-1})$ intersections with e_j, as desired.

The total number of crossings is $\sum_{1 \leq i < j \leq k} 2(3^{i-1})$, or $\sum_{j=1}^{k} \sum_{i=1}^{j-1} 2(3^{i-1}) = \sum_{j=1}^{k} (3^{j-1} - 1) \leq 3^k$.

3 The Parameterized Complexity of ocr

In this section we will derive a quadratic time algorithm for computing ocr by adapting Grohe's result [1].

Grohe showed that for a fixed k it can be decided in quadratic time whether the crossing number of a graph G is at most k [1]. Grohe's algorithm proceeds as follows: for some function $w(k)$ only depending on k it tests whether the tree-width of G is at most $w(k)$; if that is not the case, then either the crossing number of G is larger than k or we can find a part of G that is not involved in any crossing in a cr-optimal drawing. If the crossing number is larger than k, we are done; otherwise we can replace G with a smaller graph and keep track of its crossing-free part. Repeating this procedure we will eventually reach a graph of bounded tree-width for which we can decide whether $\mathrm{cr}(G) \leq k$ using Courcelle's theorem (details to be explained below).

This central result of Grohe's paper is contained in his Corollary 8 [1] which we reproduce nearly verbatim below. Here, a *k-good drawing with respect to F* of G is a drawing of G with crossing number at most k in which none of the edges of F are involved in a crossing.

Proposition 1 (Grohe [1]). *There is a quadratic time algorithm that, given a graph G and an edge set $F \subseteq E(G)$, either recognizes that the crossing number of G is greater than k or computes a graph G' and an edge set $F' \subseteq E(G')$ such that the tree-width of G' is at most $w(k)$ and G has a k-good drawing with respect to F if and only if G' has a k-good drawing with respect to F'.*

We cannot immediately apply Grohe's result as stated to help us settle the parameterized complexity of computing the odd crossing number, since it is not clear how the odd crossing number of G' (with the planarity restriction on F') relates to the odd crossing number of G (with the planarity restriction on F). Fortunately, a closer look at Grohe's proof shows that a stronger version of the proposition is true.

For a graph G let a *(k, ℓ)-good drawing with respect to F* be a drawing of G with crossing number at most k and odd crossing number at most ℓ in which

none of the edges of F are involved in any crossings. An inspection of Grohe's proof of his Corollary 8 shows that it is true for (k, ℓ)-good drawing in place of k-good drawings. The reason is that in the core step of the proof [1, Lemma 5] the redrawing is local and does not increase the odd crossing number.

Lemma 1. *There is a quadratic time algorithm that, given a graph G and an edge set $F \subseteq E(G)$, either recognizes that the crossing number of G is greater than k or computes a graph G' and an edge set $F' \subseteq E(G')$ such that the tree-width of G' is at most $w(k)$ and G has a (k, ℓ)-good drawing with respect to F if and only if G' has a (k, ℓ)-good drawing with respect to F'.*

By Theorem 1, G has odd crossing number at most k if and only if G has a $(9^k, k)$-good drawing with respect to $F = \emptyset$. We can now proceed as in Grohe's algorithm to look for such a drawing of G. We either find that the crossing number of G is larger than 9^k, which implies that the odd crossing number is larger than k (actually, much larger by the quadratic bound between odd crossing number and crossing number due to Pach and Tóth [4]) or we obtain a graph G' of tree-width at most $w(k)$ and an edge set F' such that G has odd crossing number at most k if and only if G' has a $(9^k, k)$-good drawing in which none of the edges of F' are involved in an intersection.

If we can now show that "having a $(9^k, k)$-good drawing with respect to F" can be expressed in the second-order monadic logic of graphs, we can apply Courcelle's theorem which states that formulas of second-order monadic logic can be decided in linear time for graphs of bounded tree-width (remember that the tree-width $w(k)$ of G' depends on k only, and is therefore considered fixed). Consider a $(9^k, k)$-good drawing of G if it exists. Replacing every crossing with a new vertex yields a planar drawing; adding four more vertices and edges around this vertex we can ensure that a planar drawing of the resulting graph corresponds to a $(9^k, k)$-good drawing of G. (See Figure 1.)

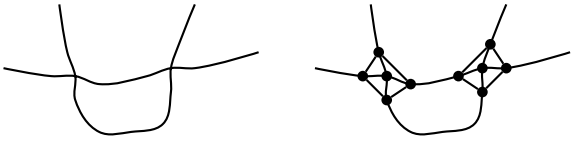


Fig. 1. Two crossings, before (*left*) and after (*right*)

Using monadic second order logic we can specify a set of at most $2k$ edges (not in F) and subdivide each of those $2k$ edges $3(9^k)$ times. These subdivided edges can now be used to express that there is a $(9^k, k)$-good drawing of G with respect to F: We can express that the ith intersection along edge e is also the jth intersection along edge f by identifying the $3i-1$st vertex along the subdivided e with the $3j-1$st vertex along the subdivided f and adding edges between vertices $3i-2$ and $3i$ on e and f to build the 4-cycle in the right half of Figure 1 to ensure e and f actually cross (rather than just touch) at their intersection point. Using this, we can write down explicitly a formula describing the order in which

edges cross every particular edge. While this leads to a formula exponentially large in 9^k, this is not a problem, since k is fixed. Since we are specifying how the crossings occur explicitly, we can restrict ourselves to those formulas describing a drawing with odd crossing number at most k.

Theorem 2. *For a fixed k we can decide* $\mathrm{ocr}(G) \leq k$ *in quadratic time.*

What about the pair-crossing number? A drawing of a graph can always be redrawn without making two pairs of edges intersect that did not intersect in the original drawing while reducing the crossing number of the drawing to at most $k2^k$ (where k is the number of edges involved in crossings) [7]. If we start with a drawing that realizes the pair-crossing number of the graph, this shows that we can always assume that a pair-crossing critical drawing has crossing number at most $k2^k$. With this result we can repeat the argument we used for odd crossing numbers, allowing us to conclude that the pair crossing number is fixed-parameter tractable.

Theorem 3. *For a fixed k we can decide* $\mathrm{pcr}(G) \leq k$ *in quadratic time.*

Acknowledgments. We would like to thank Martin Grohe for suggesting this problem, and Petr Hliněný for helpful discussions.

References

1. Grohe, M.: Computing crossing numbers in quadratic time. In: Proceedings of the 32nd ACM Symposium on Theory of Computing, pp. 231–236 (July 6–8, 2001)
2. Kawarabayashi, K.i., Reed, B.: Computing crossing number in linear time. STOC (accepted, 2007)
3. Kratochvíl, J., Matoušek, J.: String graphs requiring exponential representations. Journal of Combinatorial Theory, Series B 53, 1–4 (1991)
4. Pach, J., Tóth, G.: Which crossing number is it anyway? J. Combin. Theory Ser. B 80(2), 225–246 (2000)
5. Pelsmajer, M.J., Schaefer, M., Štefankovič, D.: Odd crossing number is not crossing number. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 386–396. Springer, Heidelberg (2006)
6. Pelsmajer, M.J., Schaefer, M., Štefankovič, D.: Removing even crossings. In: Felsner, S. (ed.) EuroComb 2005. 2005 European Conference on Combinatorics, Graph Theory and Applications of *DMTCS Proceedings*. Discrete Mathematics and Theoretical Computer Science, vol. AE, pp. 105–109 (April 2005)
7. Schaefer, M., Štefankovič, D.: Decidability of string graphs. In: STOC-2001. Proceedings of the 33th Annual ACM Symposium on Theory of Computing, pp. 241–246 (2001)
8. Tóth, G.: Note on the pair-crossing number and the odd-crossing number (Unpublished manuscript)
9. Valtr, P.: On the pair-crossing number. In: Combinatorial and Computational Geometry. Math. Sci. Res. Inst. Publ. vol. 52, pp. 569–575. Cambridge Univ. Press, Cambridge (2005)

Characterization of Unlabeled Level Planar Graphs

J. Joseph Fowler* and Stephen G. Kobourov*

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

Abstract. We present the set of planar graphs that always have a simultaneous geometric embedding with a strictly monotone path on the same set of n vertices, for any of the $n!$ possible mappings. These graphs are equivalent to the set of unlabeled level planar (ULP) graphs that are level planar over all possible labelings. Our contributions are twofold. First, we provide linear time drawing algorithms for ULP graphs. Second, we provide a complete characterization of ULP graphs by showing that any other graph must contain a subgraph homeomorphic to one of seven forbidden graphs.

1 Introduction

Simultaneous embedding enables the visualization of multiple graphs on the same set of vertices. In order to preserve the "mental map," graphs are overlaid so that corresponding vertices have the same location. The mapping between vertices may be fixed, or may not be given, or may change and dynamically evolve as in the case of colored simultaneous embeddings [1]. To accommodate this, we consider all possible 1-1 mappings between graphs. Embeddings that use no edge bends and in which no pair of edges of the same graph cross are known as simultaneous geometric embeddings [2].

Determining which graphs share a simultaneous geometric embedding has proved difficult. While Geyer *et al.* [7] have shown this cannot always be done for tree-tree pairs, the question remains open for tree-path pairs. Estrella *et al.* [5] partially answer this question by characterizing the set of trees that have a simultaneous geometric embedding with a strictly monotone path. We now extend those results by characterizing the set of all planar graphs that have a simultaneous geometric embedding with a strictly monotone path. The importance of this result lies in the fact that all positive results showing that certain pairs of graphs allow simultaneous geometric embeddings rely on reducing at least one of the graphs in the pair under consideration to a path which is realized in strictly monotone fashion. Thus, our result captures the largest possible class of graphs that can be embedded using this technique.

Rotating or stretching a drawing along a single direction does not affect crossings. As a result, we assume that the path will be drawn in a zig-zag fashion

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

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 37–49, 2007.

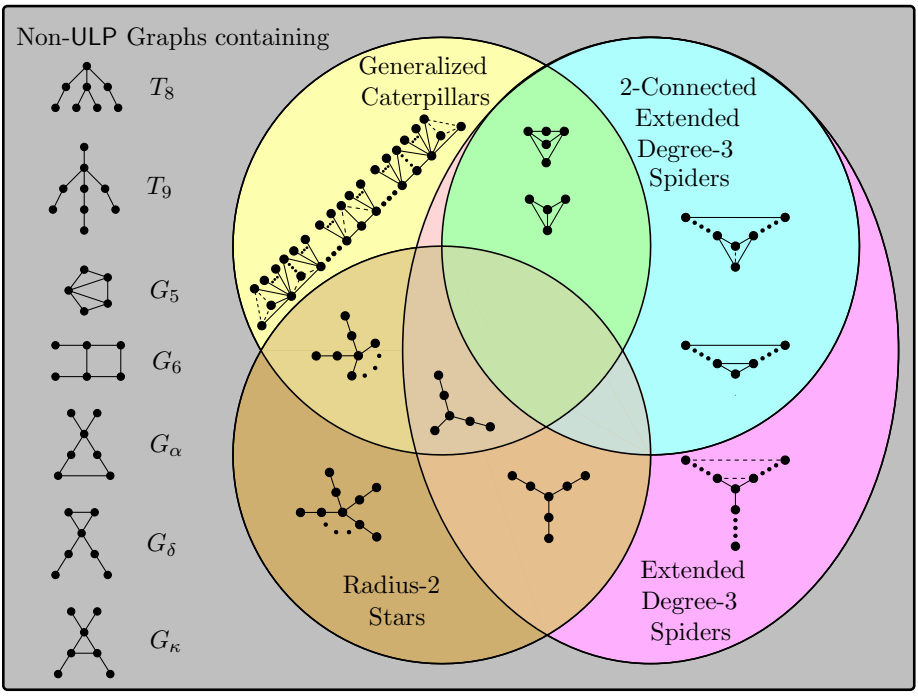


Fig. 1. A Venn diagram of the set of graphs characterized by the seven forbidden graphs T_8, T_9, G_5, G_6, G_α, G_δ, and G_κ in $\mathcal{F}$. Graphs that do not contain a subgraph homeomorphic to of any of these are generalized caterpillars, radius-2 stars, and extended degree-3 spiders.

with a difference of +1 between the y-coordinates of two successive vertices. This allows us to frame the problem of drawing the planar graph in terms of placing the vertices along a set of parallel horizontal lines, called tracks, with one vertex per track. For an n-vertex planar graph, we label the vertices from 1 to n in which the label is the y-coordinate. If a planar graph has a straight-line drawing without crossings for all $n!$ permutations of the labels, then it has a simultaneous geometric embedding with a strictly monotone path for any mapping.

A related problem is that of level planarity [9]. Our labeling forms a partition of vertices into levels with one vertex per level. If we consider a graph in which the y-coordinate of each level is distinct and all the edges are y-monotone, then we have a level drawing. If the drawing is planar, then the graph is level planar for that labeling. If this holds for each of the $n!$ labelings, then the graph is unlabeled level planar (ULP). ULP graphs are precisely those that have a simultaneous geometric embedding with strictly monotone paths for any labeling. Hence, we can also phrase our problem in terms of level planarity.

Any graph for which this cannot be done must have some subgraph homeomorphic to a forbidden graph, or obstruction, that will induce a crossing when drawn on tracks for a particular labeling. In this paper we show that ULP graphs

fall into three categories: *radius*-2 *stars*, *generalized caterpillars*, and *extended degree*-3 *spiders*. Furthermore, we show how to simultaneously embed any ULP graph with a monotone path in linear time. Finally, we complete the characterization in terms of a minimal set of seven forbidden graphs, $\mathcal{F} := \{T_8, T_9, G_5, G_6, G_\alpha, G_\delta, G_\kappa\}$; see Fig. 1.

2 Preliminaries

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

Let an n-vertex graph $G(V, E)$ have a labeling $\phi : V \mapsto [1..n]$ in which $\phi(u) \neq \phi(v)$ for all $(u, v) \in E$. A horizontal line $\ell_j = \{(x, j) \,|\, x \in \mathbb{R}\}$ for some $j \in [1..n]$ is *track* j. In a *realization* of G, each vertex $v \in V$ is placed along track $\phi(v)$ and each edge (u, v) is strictly y-monotone. Edge bends $b_1, b_2, \ldots, b_k$ may naturally occur at any point edge (u, v) intersects a track provided $\phi(u) < \phi(b_1) < \cdots < \phi(b_k) < \phi(v)$ or $\phi(u) > \phi(b_1) > \cdots > \phi(b_k) > \phi(v)$ in which b_1 is adjacent to u, b_k is adjacent to v, and b_i lies between b_{i-1} and b_{i+1} for $1 < i < k$.

A realization without crossings is a *planar realization* of G. A planar realization with one straight-line segment for each edge (u, v) is a *straight-line planar realization* of G. While any planar realization with bends can be "stretched out" in the x-direction to form a straight-line planar realization in $O(n)$ time as shown by Eades *et al.* [4], the area of the realization can become exponential.

A *level graph* $G(V, E, \phi)$ is a *directed* graph with *leveling* $\phi : V \mapsto [1..k]$ that assigns every vertex to one of k levels so that $\phi(u) < \phi(v)$ for every edge (u, v). In a *level drawing* all vertices in a level have the same y-coordinate and each edge is y-monotone. If the level drawing can be drawn without crossings, then G is *level planar*. The level planarity of G for a given leveling is independent of its orientation: First take an n-vertex undirected graph G. Then label G with labeling $\phi : V \mapsto [1..n]$. Next orient each edge (u, v) of G so that $\phi(u) < \phi(v)$ to form the level graph $\tilde{G}(V, \tilde{E}, \phi)$ with the leveling ϕ on n levels with one vertex per level. Then ask is $\tilde{G}$ level planar? If yes, repeat this process for all other labelings of G. If one never encounters a level nonplanar graph, the graph G is called *unlabeled level planar* (ULP). Hence, a ULP graph has a simultaneous embedding with a strictly y-monotone path for any labeling ϕ; see Fig 2.

The vertices placed along a track correspond to the levels in a level graph. An undirected graph with a labeling ϕ has a "planar realization" if and only if the corresponding level graph is "level planar". These two terms are interchangeable only if edge bends do not matter. If we need a simultaneous geometric embedding we use the more restrictive term "straight-line planar realization".

A *chain* C of G is a simple path denoted v_1–v_2– $\cdots$ –v_t. The vertices of C are denoted $V(C)$. A vertex v of C is ϕ-*minimal* (or ϕ-*maximal*) if it has a

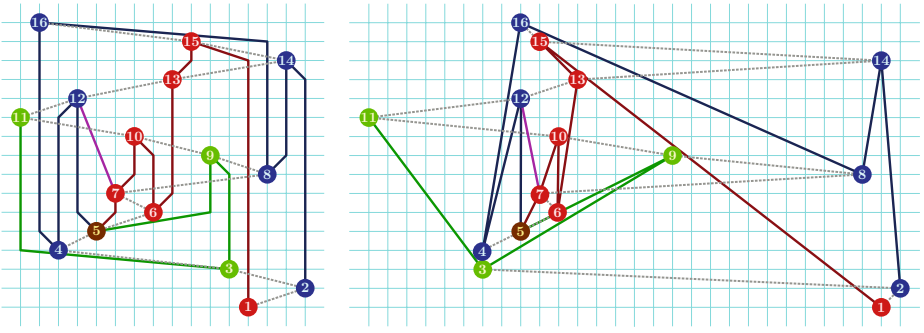


Fig. 2. Simultaneous embeddings of a path and a ULP tree with and without bends

minimal (or maximal) track number of all the vertices of $V(C)$. Such a vertex is *ϕ-extreme* if it is ϕ-minimal or ϕ-maximal.

In a graph $G(V, E)$, *subdividing* an edge $(u, v) \in E$ replaces edge (u, v) with the pair of edges (u, w) and (w, v) in E by adding w to V. A *subdivision* of G is a graph obtained by performing a series of subdivisions of G. A graph $G(V, E)$ is *isomorphic* to a graph $\tilde{G}(\tilde{V}, \tilde{E})$ if there exists a bijection $f : V \mapsto \tilde{V}$ such that $(u, v) \in E$ if and only if $\big(f(u), f(v)\big) \in \tilde{E}$. A graph $G(V, E)$ is *homeomorphic* to a graph $\tilde{G}(\tilde{V}, \tilde{E})$ if a subdivision of G is isomorphic to a subdivision of $\tilde{G}$. The *distance* between vertices u and v in a graph is the length of the shortest path from u to v. The *eccentricity* of a vertex v is the greatest distance to any other vertex. The *radius* of a graph is the minimum eccentricity of any vertex.

A *leaf vertex* is any degree-1 vertex. A *caterpillar* is a tree in which the removal of all leaf vertices yields a path (the empty graph is a special case of a path). The remaining path forms the *spine*. A *lobster* is a tree in which the removal of all leaf vertices yields a caterpillar. A *claw* is a $K_{1,3}$, whereas, a *star* is a $K_{1,k}$ for some $k \geq 3$. A *double star* is a star in which each edge has been subdivided once. A *radius*-2 *star* (R-2S) is any subgraph of a double star with radius 2. A *degree*-3 *spider* is an arbitrarily subdivided claw. The following six types of "edges" in Fig. 3 allow us to generalize a caterpillar and to extend a degree-3 spider to include cycles.

Definition 1

(a) A K_3 edge *is the cycle* u–v–w–u *on vertices* $\{u, v, w\}$
(b) A C_4 edge *is the cycle* u–s–v–t–u *on vertices* $\{u, v, s, t\}$.
(c) A kite edge *is the cycle* u–s–v–t–u *with edge* s–t *on vertices* $\{u, v, s, t\}$.
(d) A K_3^* edge *is set of cycles* u–v–w'–u *with edge* u–v *on vertices* $\{u, v\} \cup W$ *where* $w' \in W$ *for some possibly empty vertex set* W.
(e) A C_4^+ edge *is set of cycles* u–w–v–w'–u *on vertices* $\{u, v, w\} \cup W$ *where* $w' \in W$ *for some non-empty vertex set* W.
(f) A K_4 edge *is the complete graph on the vertices* $\{u, x, y, z\}$.

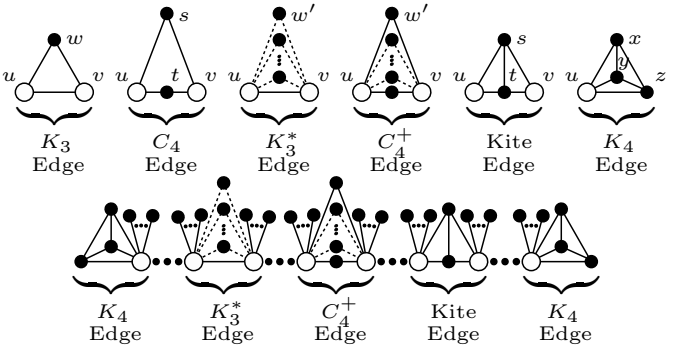


Fig. 3. The six types of H edges used to from a GC on the second line

Definition 2. *A* generalized caterpillar *(GC) is a caterpillar in which each edge u'–v' along the* spine *can be replaced with a K_3^*, C_4^+, or kite edge (and the two edges at the end of the spine can also be replaced by a K_4 edge) in which vertex u (and v if present) replaces vertex u' (and v'); see Fig. 3.*

Definition 3

(a) A 1-connected extended degree-3 spider *(1-CE3-S) is a degree-3 spider with two optional edges connecting*
  (i) two of three vertices adjacent to the degree-3 vertex and
  (ii) two of the three leaf vertices; see Fig. 4(a).
(b) A 2-connected extended degree-3 spider *(2-CE3-S) is a cycle or a cycle with one K_3, C_4 or kite edge, see Fig. 4(b).*
(c) A extended degree-3 spider *(E3-S) is either a 1-connected extended degree-3 spider or a 2-connected extended degree-3 spider.*

These definitions allows us to make the following observation.

Observation 4. *Every spanning tree of a GC is a caterpillar. Every spanning tree of a E3-S is a degree-3 spider or a path.*

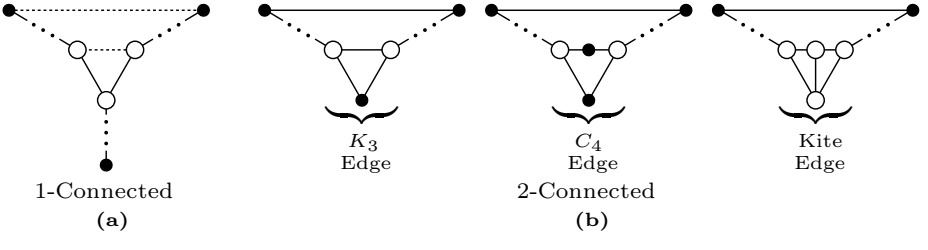


Fig. 4. A extended degree-3 spider is either (a) a 1-CE3-S or (b) a 2-CE3-S

3 Graphs with Planar Realizations on Tracks

In this section we show that radius-2 stars (R-2 S), generalized caterpillars (GC), and extended degree-3 spiders (E 3-S) are level planar for any labeling. We do this by presenting linear time algorithms for straight-line, crossings-free drawing of any such graph on the tracks determined by its labeling. More formally, we show that $\mathcal{P} = \{G : G$ is a R-2 S, GC, or E 3-S$\}$ is ULP.

The next lemma from [5] shows this for a R-2 S.

Lemma 5 ***(Lemma 4 of [5]).*** *An n-vertex radius-2 star can be straight-line planarly realized in $O(n)$ time on a $(2n+1) \times n$ grid for any labeling.*

The following lemmas show how a GC and the two types of a E 3-S also have compact planar realizations on tracks. We give a proof sketch for the next lemma; the full proof can be found in [6].

Lemma 6. *An n-vertex generalized caterpillar can be straight-line planarly realized in $O(n)$ time within an $n \times n$ grid for any labeling.*

Proof Sketch: We obtain the cut vertices of the GC using the vertices of its spanning tree, which must be a caterpillar by Observation 4, as candidates. With these we can draw each incident K_3^*, C_4^+, kite, and K_4 spine edge using at most $4 \times n$ space for each one proceeding left to right along the spine; see Fig. 5. If we were not constrained to an integer grid, one could place all the incident edges with leaf vertices in a sufficiently narrow region above and below each cut

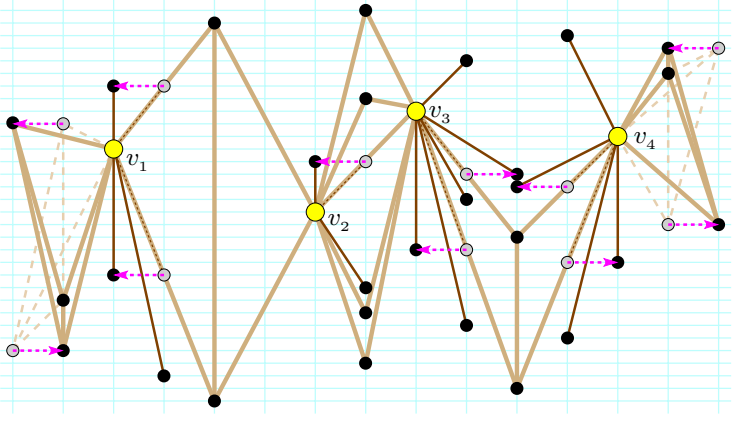


Fig. 5. The gray vertices are initial locations of vertices in a straight-line planar realization of a GC on a 14×32 grid. The arrows avoid crossings or overlapping edges. A leaf is initially placed to the right of its cut vertex except for the last one with its leaves placed to the left. Overlaps are eliminated by moving leaves left and right, e.g., the leaves between v_3 and v_4. The K_4 edges incident to v_1 and v_4 show initial locations with dashed edges leading to crossings that are eliminated by switching the location of the two incident vertices.

vertex. Being restricted to integer coordinates, we shift the endpoint of a leaf vertex left or right by one space as needed to avoid overlapping edges. □

Lemma 7. *An n-vertex 1-connected extended degree-3 spider can be planarly realized in $O(n)$ time on an $n \times n$ grid for any labeling.*

Proof. We show how to draw G on tracks with at most one bend per edge for a labeling ϕ. We first draw a subgraph that is a degree-3 spider T with an extra edge in G between two of three vertices adjacent to the root vertex r (the unique degree-3 vertex) of T. Next, we accommodate an extra edge in G connecting two leaf vertices of T.

Let T' be portion of the T drawn so far. We maintain two invariants:

(1) two of the leaf vertices $v_{\min}$ and $v_{\max}$ of T' are ϕ-extreme and
(2) T' only intersects the track of the third leaf vertex v_{mid} either to the left or right of v_{mid}.

Provided these invariants hold, we keep placing the next vertex v adjacent to v_{mid} in $T-T'$ one space to the left or right of T' at x-coordinate v_x depending on which side of the track of v_{mid} that T' intersects. By (2), T' does not intersect one side of the track of v_{mid}. Whenever we draw from v to w (in this case $w = v_{\text{mid}}$), we bend the edge at $\big(v_x,\, \phi(w)-1\big)$ if $\phi(v) < \phi(w)$ and at $\big(v_x,\, \phi(w)+1\big)$ otherwise. We keep doing this until v becomes ϕ-extreme. Either $v_{\min}$ or $v_{\max}$ becomes v_{mid}. Since that vertex was previously ϕ-extreme by invariant (1), T' now only intersects its track either to the left or right, maintaining invariant (2).

We start drawing T until both invariants hold for T'. Place r at $\big(0,\, \phi(r)\big)$. Let $\{u, v, w\}$ be the neighbors of r in T. Let $v_{\min}$, v_{mid} and $v_{\max}$ be these vertices such that $\phi(v_{\min}) < \phi(v_{\text{mid}}) < \phi(v_{\max})$. If $\phi(v_{\min}) < \phi(r) < \phi(v_{\max})$, drawing edges from r to vertices at $\big(-1,\, \phi(v_{\min})\big)$, $\big(1,\, \phi(v_{\max})\big)$, and $\big(2,\, \phi(v_{\text{mid}})\big)$ satisfies both invariants. In this case, we can also add a straight-line edge between any one pair of $\{u, v, w\}$. Otherwise, suppose w.l.o.g that $\phi(r) < \phi(v_{\min})$. Let $\{a, b, c\}$ be the ϕ-maximal vertices of the portions of the chains in T from r to the point each chain crosses the track of r such that $\phi(a) > \phi(b) > \phi(c)$. Assume w.l.o.g. that u is first vertex of the chain with a. There are two cases:

(i) If edge (v, w) is not in G, assume w.l.o.g. edge (u, w) is in G. Extend the chain starting with u to the right of r until it reaches a becoming $v_{\max}$. Place v one right of a with an edge bend at $\big(v_x,\, \phi(r) + 1\big)$.
(ii) If edge (v, w) is in G, then assume w.l.o.g. v is the first vertex of the chain with b. Extend this chain to the right until it reaches b. Place u one right of b with an edge bend at $\big(u_x,\, \phi(r) + 1\big)$ and continue to extend the chain to the right until it reaches a becoming $v_{\max}$.

Place w at $\big(-1,\, \phi(w)\big)$ and extend the chain to the left until it becomes $v_{\min}$. Edge (u, w) or (v, w) can be drawn with a straight-line edge since u or v is one right of r. In both cases, invariants (1) and (2) hold; see Fig. 6.

If an edge connects two leaf vertices to form a cycle C in T, we first draw subtree $\tilde{T}$ in which two leaf vertices $c_{\min}$ and $c_{\max}$ of $\tilde{T}$ are the ϕ-extreme vertices

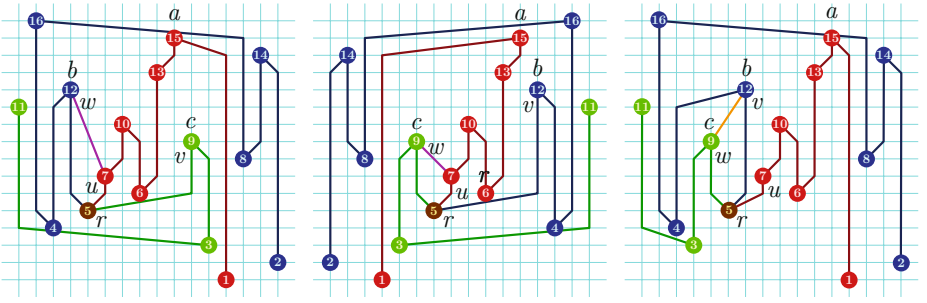


Fig. 6. Examples of three 1-C E 3-Ss on 16×16 grids. The only difference is the edge between one pair of the three vertices adjacent to the root. If this edge is incident to u, the first vertex along chain with the vertex a, case (i) applies as in (a) and (b). Otherwise, case (ii) applies as in (c).

of C. The above algorithm ensures the other chain of $\tilde{T}$ only intersects the tracks of $c_{\min}$ and $c_{\max}$ to the right or left, blocking one direction, but not both. Whichever $c_{\min}$ or $c_{\max}$ is leftmost or rightmost of $\tilde{T}$, say that $c_{\min}$ is rightmost, we extend the rest of C from $c_{\min}$ right until reaching v adjacent to $c_{\max}$. Then we draw an edge from v to $c_{\max}$ with a bend at $\big(v_x, \phi(c_{\max}) - 1\big)$. □

We next give a similar realization of a 2-C E 3-S with bends—the difference being that most edges are straight except for one or two edges that might require a bend.

Lemma 8. *An n-vertex 2-connected extended degree-3 spider can be planarly realized in $O(n)$ time on an $n \times n$ grid for any labeling.*

Proof. Let ϕ be a labeling of a 2-C E 3-S G. If G is merely a cycle C, then C can be planarly realized on an $n \times n$ grid with one edge bend. Begin with the ϕ-maximal vertex v_1 at the first position and proceed left to right placing each subsequent vertex in the cycle one to the right of the previous one until reaching the last vertex v_k that is also adjacent to v_1. The edge v_1–v_k requires only one bend directly above v_k routing the edge above all the other vertices.

By Definition 3, a 2-C E 3-S is at worst a cycle with a kite edge between u and v with common neighbors $\{s, t\}$ connected by edge s–t such that $\phi(s) > \phi(t)$. If s and t are ϕ-extreme, then we can draw the cycle without t starting from s and ending with v as above and place t below s drawing the straight edges s–t and t–u. Then we draw t–v with a bend directly below v and route the edge below all the others; see Fig. 7(a). Otherwise, either s or t is not ϕ-extreme in which case the other ϕ-extreme one is used to draw the cycle so as to not end with u or v; see Fig. 7(b). Suppose that s is not ϕ-maximal, then t can be placed directly below s and the three additional edges can be added as straight edges. □

We can remove the bends on the edges by stretching the layout which yields the next corollary; the full proof can be found in [6].

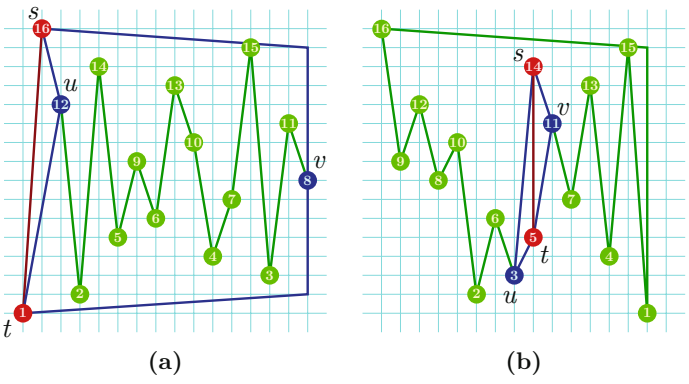


Fig. 7. Planar realizations of two 16-level 2-CE3-Ss on 16×16 grids illustrating the two cases in which s and t are ϕ-extreme

Corollary 9. *An n-vertex 1-connected extended degree-3 spider with radius r can be straight-line planarly realized in $O(n)$ time on an $O(r!\,3^r) \times n$ grid for any labeling, whereas, an n-vertex 2-connected extended degree-3 spider can be straight-line planarly realized in $O(n)$ time on an $n^2 \times n$ grid for any labeling.*

Combining Lemmas 5, 6, 7, 8, and Corollary 9, we have our first theorem.

Theorem 10. *Any graph from $\mathcal{P}$ has a simultaneous geometric embedding with a strictly monotone path for any labeling.*

4 Forbidden Graphs

We give seven forbidden graphs $\mathcal{F} := \{T_8, T_9, G_5, G_6, G_\alpha, G_\delta, G_\kappa\}$ that do not always have a simultaneous geometric embedding with a strictly monotone path; see Fig. 8. For each we provide a labeling that forces self-crossings. As noted previously for a given labeling, a graph has a straight-line planar realization if and only if it also has a planar realization that allows edge bends provided the edges remain strictly monotone [4]. Hence, it suffices to only consider straight-line edges in this section.

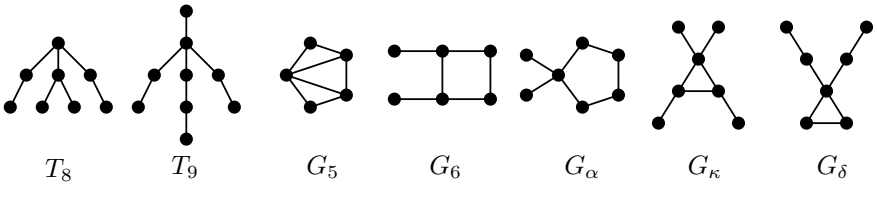


Fig. 8. The seven forbidden graphs of $\mathcal{F}$

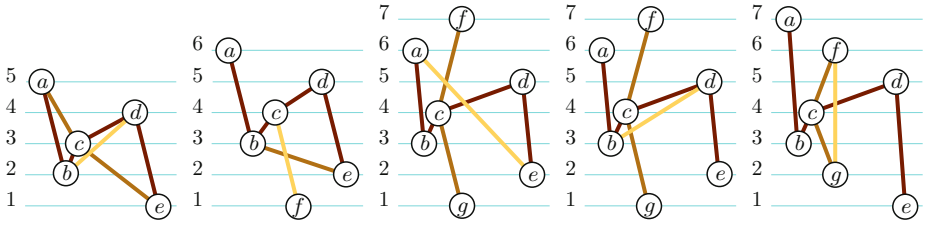


Fig. 9. Labelings that force self-crossings for G_5, G_6, G_α, G_κ, and G_δ

Lemma 11. *There exist labelings that prevent each graph in $\mathcal{F}$ from having planar realizations on tracks.*

Proof. The labelings of T_8 and T_9 were shown not to have planar realizations in [5]. We need to do the same for the labelings of the remaining five graphs in $\mathcal{F}$ given in Figure 9.

Let C denote the chain a–b–c–d–e, which is highlighted in each of the graphs in Figure 9. Observe that $\phi(a) > \phi(d) > \phi(c) > \phi(b) > \phi(e)$ in which C forms an backwards 'N'. If the rest of C intersects the track of c only on the left or right of c, then some part of the chain a–b–c must cross the chain c–d–e. Hence, we only need to consider embeddings in which c lies between the edge a–b and d–e, i.e., one of those edges intersect the track of c to the left, while the other intersects on the right. To avoid a self crossing of C, a–b must intersect the tracks of c and d on the same side of both vertices. The same goes for the d–e intersecting the tracks of b and c on the same side. So we can assume w.l.o.g. that a–b intersects the tracks of c and d to the their left while d–e intersects the tracks of b and c to the their right as is the case in all the figures.

For G_5, c and d being on the same side of a–b means that the edge b–d must also lie between the two edges. The only question is whether b–d intersects the track of c to the left or right. If it is to the left, then b–d must cross a–c, otherwise, it must cross c–e as in Fig. 9(a).

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

(i) a–b if it intersects the track of b to the left since c is right of a–b,
(ii) d–e if it intersects the track of e to the right since c is left of d–e,
(iii) b–e otherwise since it must intersect the track of b to the right and e to the left as in Fig. 9(b).

In G_α, G_δ and G_κ for c–f and c–g to avoid crossing C, c–f must intersect the track of d to the left while c–g must intersect the track of b to the right. Since $\phi(f) > \phi(a) > \phi(e) > \phi(g)$ in G_α and G_κ, c–f must intersect the track of a to the right while c–g must intersect the track of e to the left. However, in G_δ $\phi(a) > \phi(f) > \phi(g) > \phi(e)$ so that a–b must intersect the track of f to the right while d–e must intersect the track of g to the left.

This means in G_α for a–e to avoid crossing C, as in Fig. 9(c), it must either intersect the track of d to the right in which case it must cross c–f or b to the left in which case it must cross c–g.

This also means in G_κ if b–d intersects the track of c to the right as in Fig. 9(d), it will cross c–g. Otherwise, b–d will cross c–f.

Finally, in G_δ if f–g intersects the track of c to the right as in Fig. 9(e), it will cross c–d–e. Otherwise, f–g will cross a–b–c. □

Corollary 12. *A graph containing a subgraph homeomorphic to a graph in $\mathcal{F}$ does not have a simultaneous geometric embedding with a strictly monotone path for all labelings.*

Proof. We provide a labeling ϕ of a graph G containing a subgraph homeomorphic to a graph $\tilde{G} \in \mathcal{F}$. Let h be the homeomorphism that maps an edge in $\tilde{G}$ to a path in G and a vertex in $\tilde{G}$ to the endpoint of such a path in G. Label the vertices of $\tilde{G}$ using the appropriate labeling ϕ' from Lemma 11 that forces a self-crossing in $\tilde{G}$. We maintain the same relative ordering of the labels in G as in $\tilde{G}$. In particular, we want $\phi\big(h(u)\big) < \phi\big(h(v)\big)$ if and only if $\phi'(u) < \phi'(v)$ for each edge (u, v) in $\tilde{G}$. For each path $h\big((u, v)\big) = p_{(u,v)} = v_1$–$v_2$–$\cdots$–$v_k$ in G that corresponds to an edge (u, v) in $\tilde{G}$, we want $\phi(v_1) < \phi(v_2) < \cdots < \phi(v_k)$ if $\phi'(u) < \phi'(v)$. We can assign the other vertices of G not in the image of h arbitrary labels. Then every edge (u, v) in $\tilde{G}$ corresponds to a strictly monotone path $p_{(u,v)}$ in G preserving the nonplanarity of the realization of $\tilde{G}$. □

5 Completing the Characterization

The next lemma shows that the seven forbidden graphs of $\mathcal{F}$ are minimal; the removal of any edge from any of the seven yields a graph from $\mathcal{P}$.

Lemma 13. *Each forbidden graph is minimal, in that the removal of any edge yields one or more GCs, R-2Ss, or E3-Ss.*

Proof. Showing that the removal of any edge from T_8 or T_9 yielded a caterpillar, radius-2 star, or degree-3 spider, all members of $\mathcal{P}$, was done in [5]. For G_5 in which a–b–d–e–c–a, a–b–c–a, b–c–d–b, c–d–e–c all form cycles shown in Fig. 9(a), the removal of edges b–c or c–d forms a 2-CE3-S, while removing of any other edge forms a GC. For G_6 in which b–e–d–c forms a 4-cycle shown in Fig. 9(b), the removal of any edge leaves a GC. For G_α shown in Fig. 9(c), the removal of c–f or c–g leaves a E3-S. Removing any other edge yields a GC. For G_κ in which b–c–d–b forms a 3-cycle shown in Fig. 9(d), the removal of c–f or c–g leaves a 1-CE3-S, while removing any other edge leaves a GC. For G_δ in which c–f–g–c forms a 3-cycle shown in Fig. 9(e), the removal of c–b or c–d leaves a GC and a lone edge. Removing a–b, d–e, or f–g leaves a GC, and removing c–f or c–g leaves a degree-3 spider. □

Finally, the next theorem completes our characterization.

Theorem 14. *Every connected graph either contains a subgraph homeomorphic to one of the seven forbidden graphs of $\mathcal{F}$, or it is a generalized caterpillar, radius-2 star, or a extended degree-3 spider, which form the collection of graphs $\mathcal{P}$ that have simultaneous geometric embeddings with strictly monotone paths for any labelings, the set of ULP graphs.*

Proof Sketch: We sketch out the proof here; the complete proof can be found in [6]. The high-level proof idea is to use induction on the number of edges in which we have as an inductive hypothesis that any connected graph with fewer than m edges that does not contain one of the seven forbidden subgraphs of $\mathcal{F}$ is a GC, a R-2 S, or a E 3-S. As a base case are all connected graphs with two edges, which is only the path of length 2, which is clearly a GC. Let $G(V, E)$ be some connected graph then with m edges. Remove a single edge e to form $G' = G - \{e\}$ and the inductive hypothesis holds for G'. We then need to consider all the ways of adding back in the edge e to form G'' showing that no matter what G'' is a GC, a R-2 S, or a E 3-S or contains a copy of one of the seven graphs of $\mathcal{F}$. □

6 Previous and Future Work

Level planar graphs are historically studied in the context of directed graphs, which restricts the types of levelings that can be assigned. Additionally, they are generally considered in the context of a particular leveling such as ones given by hierarchical relationships with an emphasis on minimizing the number of levels required to maintain planarity. In contrast, our application of level planarity has been in terms of the underlying undirected graph with one vertex per level with no consideration given to minimizing levels.

Many of the problems regarding level planarity have been addressed, including the ability to recognize a level planar graph and produce an embedding in linear time [9,10]. However, all of these results are for a particular leveling and do not generalize to the context of considering the level planarity of all the level graphs induced by all possible $n!$ labelings of a given undirected graph. Running either of these linear time algorithms for each possible level graph leads to an exponential running time. Using our approach we achieve this in linear time.

We gave a characterization of ULP graphs akin to Kuratowksi's characterizations of planar graphs [11]; we provided a forbidden set of graphs $\mathcal{F}$ that play the same role with respect to ULP graphs that K_5 and $K_{3,3}$ play with respect to planar graphs. Just as Kuratowksi's theorem states that a graph is planar if and only if it does not contain a subgraph that is a subdivision of K_5 or $K_{3,3}$, we show a graph is ULP if and only if it does not contain a subgraph homeomorphic to a forbidden graph of $\mathcal{F}$.

The analogue of Kuratowksi's theorem for level planar graphs are minimum level non-planar patterns [8]. These are based on the characterization of hierarchies by Di Battista and Nardelli [3]. Unlike our characterization, these patterns are not solely based upon the underlying graph, but also upon the given leveling. The same graph with two different levelings that is level non-planar for each may

very well match two distinct patterns since the reasons that a crossing is forced in each can be entirely different.

Estrella *et al.* [5] presented linear time recognition algorithms for the class of ULP trees. Providing the equivalent algorithms for general ULP graphs remains for future work.

References

1. Brandes, U., Erten, C., Fowler, J.J., Frati, F., Geyer, M., Gutwenger, C., Hong, S., Kaufmann, M., Kobourov, S.G., Liotta, G., Mutzel, P., Symvonis, A.: Colored simultaneous geometric embeddings. In: COCOON 2007. LNCS, vol. 4598, pp. 254–263. Springer, Heidelberg (2007)
2. Brass, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous graph embedding. Computational Geometry: Theory and Applications 36(2), 117–130 (2007)
3. Di Battista, G., Nardelli, E.: Hierarchies and planarity theory. IEEE Transactions on Systems, Man, and Cybernetics 18(6), 1035–1046 (1988)
4. Eades, P., Feng, Q., Lin, X., Nagamochi, H.: Straight-line drawing algorithms for hierarchical graphs and clustered graphs. Algorithmica 44(1), 1–32 (2006)
5. Estrella-Balderrama, A., Fowler, J.J., Kobourov, S.G.: Characterization of unlabeled level planar trees. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 367–369. Springer, Heidelberg (2007)
6. Fowler, J.J., Kobourov, S.G.: Characterization of unlabeled planar graphs. Technical Report TR06-04, University of Arizona (2006), ftp://ftp.cs.arizona.edu/reports/2006/TR06-04.pdf
7. Geyer, M., Kaufmann, M., Vrto, I.: Two trees which are self-intersecting when drawn simultaneously. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 201–210. Springer, Heidelberg (2006)
8. Healy, P., Kuusik, A., Leipert, S.: A characterization of level planar graphs. Discrete Math. 280(1-3), 51–63 (2004)
9. Jünger, M., Leipert, S.: Level planar embedding in linear time. J. Graph Algorithms Appl. 6(1), 67–113 (2002)
10. Jünger, M., Leipert, S., Mutzel, P.: Level planarity testing in linear time. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 224–237. Springer, Heidelberg (1999)
11. Kuratowski, C.: Sur les problèmes des courbes gauches en Topologie. Fundamenta Mathematicae 15, 271–283 (1930)

Cyclic Level Planarity Testing and Embedding
(Extended Abstract)

Christian Bachmaier, Wolfgang Brunner, and Christof König

University of Passau, Germany
{bachmaier,brunner,koenig}@fim.uni-passau.de

Abstract. In this paper we introduce cyclic level planar graphs, which are a planar version of the recurrent hierarchies from Sugiyama et al. [8] and the cyclic extension of level planar graphs, where the first level is the successor of the last level. We study the testing and embedding problem and solve it for strongly connected graphs in time $\mathcal{O}(|V| \log |V|)$.

1 Introduction

Cyclic level planar graphs receive their motivation from two sources: level planar graphs and recurrent hierarchies.

A level graph is a directed acyclic graph with a level assignment for each node. Nodes on the same level are placed on a horizontal line and edges are drawn downwards from the upper to the lower end node. Level planarity has been studied intensively in recent years. Jünger and Leipert [6] completed this series and established a linear time algorithm for the level planarity testing and embedding problem. Bachmaier et al. [1] extended level planarity to radial level planarity. Here the levels are concentric circles and the edges are directed from inner to outer circles. Again there are linear time algorithms for the testing and embedding problem. Radial level planar graphs can also be drawn on a cylinder where each level is a circle on the surface.

Recurrent hierarchies were introduced by Sugiyama et al. [8] over 25 years ago. A recurrent hierarchy is a level graph with additional edges from the last to the first level. Here two possible drawings are natural: The first is a 2D drawing where the levels are rays from a common center, and are sorted counterclockwise by their number, see Fig. 1. All nodes of one level are placed on different positions on the corresponding ray and an edge $e = (u, v)$ is drawn as a monotone counterclockwise curve from u to v wrapping around the center at most once. The second is a 3D drawing of a level graph on a cylinder, see Fig. 2. A planar recurrent hierarchy is shown on the cover of the book by Kaufmann and Wagner [7], in which it is stated that recurrent hierarchies are "unfortunately [...] still not well studied". This paper will improve this situation.

We consider cyclic k-level graphs with edges spanning many levels. First, observe that every (undirected) planar graph with a given embedding and any level assignment is a cyclic level planar graph, if the edges are arbitrary Jordan curves. These curves can even be monotone, such that every edge goes either

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 50–61, 2007.

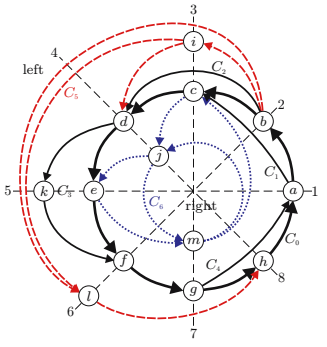


Fig. 1. 2D drawing of a cyclic 8-level graph G

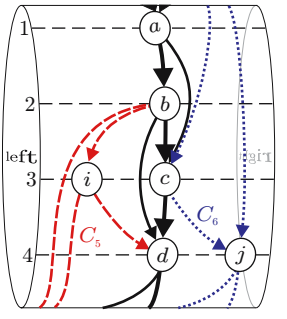


Fig. 2. Drawing of G on a cylinder

clockwise or counterclockwise around the center in the 2D drawing. This drawing can be obtained by a variation of the algorithm of de Fraysseix et al. [3], wrapping the graph $|V|$ times round the center and successively moving each node counterclockwise to its level. Thus we limit the edges as described above.

Healy and Kuusik [4] have presented an algorithm for level planarity testing and embedding using the *vertex-exchange graph*. For proper graphs the algorithm finds an embedding in $\mathcal{O}(|V|^3)$. Every non-proper graph can be made proper by adding at most $\mathcal{O}(|V|^2)$ dummy nodes on the edges which leads to a running time of $\mathcal{O}(|V|^6)$ for non-proper graphs. We claim that this algorithm can be used for testing and embedding cyclic k-level graphs without major modifications as the algorithm can handle edges from level k to level 1 as any other edge.

In this paper we improve this result and show that cyclic level planarity testing and embedding can be solved in $\mathcal{O}(|V| \log |V|)$ time for strongly connected non-proper graphs.

2 Preliminaries

A *cyclic k-level graph* $G = (V, E, \phi)$ ($k \geq 2$) is a directed graph without self-loops with a given surjective level assignment of the nodes $\phi\colon V \to \{1, 2, \ldots, k\}$. For two nodes $u, v \in V$ let $\text{span}(u, v) := \phi(v) - \phi(u)$ if $\phi(u) < \phi(v)$ and $\text{span}(u, v) := \phi(v) - \phi(u) + k$ otherwise. For an edge $e = (a, b) \in E$ we define $\text{span}(e) := \text{span}(a, b)$. A graph is *proper* if for all edges $e \in E$ $\text{span}(e) = 1$ holds. For a simple path or simple cycle P we define $\text{span}(P) := \sum_{e \in E(P)} \text{span}(e)$. A drawing is *(cyclic level) plane* if the edges do not cross except on common endpoints. A cyclic k-level graph is *(cyclic level) planar* if such a drawing exists. The *right outer face* is the face of the 2D drawing containing the center and the *left outer face* is the unbounded face. A *cyclic level planar embedding* consists of two lists $N^-(v)$ and $N^+(v)$ for each node $v \in V$ which contain the end nodes of ingoing and outgoing edges, respectively, which are both ordered from left to right.

Proposition 1 (Euler, [4]). *Let G be a planar cyclic k-level graph. Then $|E| \leq 3|V| - 6$. If G is proper, $|E| \leq 2|V| - k$. Both inequalities are tight.*

3 Testing Strongly Connected Graphs

In this section we present our algorithm embedCyclicLevelPlanar(G) for cyclic level planarity testing and embedding of strongly connected graphs. The algorithm is quite technical; this seems to be inherent to level planarity and its extensions. Algorithm 1 has some similarities to the planarity testing algorithm by Hopcroft and Tarjan [5] and consists of three phases. The first phase (see lines 1–2 and Sect. 3.1) searches for a simple cycle C_0 in G and splits $G \setminus C_0$ into its "connected" *components* $C_1, \ldots, C_p$ which correspond to *segments* in [5]. The second phase (lines 3–12, Sect. 3.2) tries to find a cyclic level planar embedding for each C_i, s.t. all nodes in $V(C_i) \cap V(C_0)$ lie on the same border of the embedding. If a component does not wrap around the center completely, a level planarity test is applied. Otherwise the test is applied to each of its subcomponents. The third phase (lines 13–23, Sect. 3.3) decides for each C_i whether it will be embedded on the left or right side of C_0.

Algorithm 1: embedCyclicLevelPlanar

Input: G: a cyclic k-level graph
Output: a cyclic level planar embedding H of G or abort

```
1  Let C_0 be a simple cycle in G with embedding H   // abort if span(C_0) > k
2  Let C := {C_1, ..., C_p} be the components of G sorted by increasing span
3  foreach C_i ∈ C do
4      if span(C_i) ≤ k then embedLevelPlanar(C_i')   // and thus C_i, abort if it fails
5      else
6          initialize NEXT_PAIRS
7          while NEXT_PAIRS ≠ ∅ do
8              (u, v) := remove(NEXT_PAIRS)
9              S_i := findSubcomponent(u, v)          // abort if it fails
10             embedLevelPlanar(S_i')                 // and thus S_i, abort if it fails
11             add S_i to the left side of the embedding of C_i
12             update NEXT_PAIRS
13 build the set R by constructing a rigid component R for each virtual edge of C_0
14 foreach C_i ∈ C do
15     traverse the border of R for consecutive nodes in link(C_i)   // abort if it fails
16     update R
17 foreach R ∈ R do
18     traverse the tree of R formed by rigid components and for each node R_j with
19         C_i = component(R_j) set d_i to the number of RIGHT entries on its path
20 foreach C_i ∈ C do
21     if d_i is uninitialized then embed C_i to the side of H where its link nodes are
22     else if d_i is even then embed C_i to the left side of H
23     else embed C_i to the right side of H
24 return H
```

3.1 Splitting the Graph

The first step of the algorithm is to find a simple cycle C_0 in G. Such a cycle exists as G is strongly connected. If $\text{span}(C_0) > k$, the cycle C_0 is not cyclic level planar and the algorithm aborts. Otherwise $\text{span}(C_0) = k$ holds and C_0 has exactly one possible embedding.

Definition 1. *Let C_0 be a simple cycle of G. Two edges $e_1, e_2 \in E \setminus E(C_0)$ are part of the same* component *C if there exists an undirected path P connecting an end node of e_1 to an end node of e_2 s.t. $V(P) \cap V(C_0) = \emptyset$. C has at most two levels with exactly one node of C_0 and no other nodes of C on them and no edges of C crossing these levels. If C has exactly two such levels, one of the nodes on them has no ingoing edges and one no outgoing edges. We call these nodes* $\text{upper}(C)$ *and* $\text{lower}(C)$*, respectively. If C has exactly one such node, we call it* $\text{upper}(C) = \text{lower}(C)$*. In both cases we call C* open *and define* $\text{span}(C) := \text{span}(\text{upper}(C), \text{lower}(C))$*. Let* $\text{link}(C)$ *be the set $V(C) \cap V(C_0)$ sorted from* $\text{upper}(C)$ *to* $\text{lower}(C)$ *by increasing level. If* $\text{upper}(C) = \text{lower}(C)$*, the first and the last element in* $\text{link}(C)$ *is this node. If C has no such levels, we call C* closed *and define* $\text{span}(C) := \infty$ *and* $\text{link}(C)$ *as all nodes in $V(C) \cap V(C_0)$ sorted by increasing level with the (arbitrary) first and last node being the same.*

Next all components $C_1, \ldots, C_p$ are computed by a connectivity test which can be done in time $\mathcal{O}(|V|)$. For each C_i we add a *virtual edge* for each pair of consecutive nodes in $\text{link}(C_i)$. Each virtual edge corresponds to a path in C_0. The virtual edges ensure that in the computed embedding of C_i all nodes in $\text{link}(C_i)$ are on the same side of the border. This is obviously necessary to obtain a cyclic level planar embedding of $C_0 \cup C_i$ as C_i is connected. The virtual edges are deleted after an embedding of C_i is found.

See Fig. 1 as an example. Let (a, b, c, d, e, f, g, h) be the cycle C_0. There are components $C_1, \ldots, C_6$ with $E(C_1) = \{(a, c)\}$, $E(C_2) = \{(b, d)\}$, $E(C_3) = \{(d, k), (k, f)\}$ and $E(C_4) = \{(g, a)\}$. C_5 and C_6 consist of the dashed and dotted edges, respectively. C_1 through C_5 are open components and C_6 is a closed component, $\text{upper}(C_5) = b$, $\text{lower}(C_5) = h$, $\text{link}(C_5) = [b, d, h]$ and $\text{span}(C_5) = 6$. For C_6 $\text{span}(C_6) = \infty$ and $\text{link}(C_6) = [c, e, c]$ hold and $\text{upper}(C_6)$ and $\text{lower}(C_6)$ are undefined. Without the edge (m, j) C_6 would be an open component with $\text{upper}(C_6) = \text{lower}(C_6) = c$ and $\text{span}(C_6) = 8$.

3.2 Embedding the Components

If C is an open component with $\text{span}(C) < k$, we set $C' = C$. If $\text{span}(C) = k$, we construct C' by duplicating the level of $\text{upper}(C) = \text{lower}(C)$ with $\text{upper}(C')$ receiving all outgoing and $\text{lower}(C')$ all ingoing edges of the node $\text{upper}(C) = \text{lower}(C)$. After adding an edge $(\text{upper}(C'), \text{lower}(C'))$ the last phase of the linear time level planarity embedding algorithm of [6] is applied to the *st-graph* C'. In the remaining case C is a closed component. We decompose C into *subcomponents* and apply the last phase of the algorithm of [6] to each subcomponent. This decomposition is possible because G is strongly connected.

Definition 2. *Let C be a closed component. The* subcomponents $S_0, \ldots, S_q$ *are an edge disjoint decomposition of C. S_0 consists of the nodes in* $\text{link}(C)$ *and the virtual edges of C. Let $H_j = \bigcup_{i=0}^{j} S_i$ $(0 \leq j \leq q)$. We construct S_j $(1 \leq j \leq q)$ s.t. $1 \leq |V(S_j) \cap V(H_{j-1})| \leq 2$. If $|V(S_j) \cap V(H_{j-1})| = 2$, we call the two nodes* $\text{upper}(S_j)$ *and* $\text{lower}(S_j)$*. If $|V(S_j) \cap V(H_{j-1})| = 1$ holds, we call this node* $\text{upper}(S_j) = \text{lower}(S_j)$*. In both cases S_j consists of all edges lying on a path P from* $\text{upper}(S_j)$ *to* $\text{lower}(S_j)$ *with* $\text{span}(P) = \text{span}(\text{upper}(S_j), \text{lower}(S_j))$*. Let* $V'(S_j) := V(S_j) \setminus \{\text{upper}(S_j), \text{lower}(S_j)\}$*. We call $v \in V'(S_j)$* externally active *if $deg_{S_j}(v) < deg_C(v)$ and S_j* externally active *if $V'(S_j)$ contains such a node. We call a node $v \in V(H_{j-1})$* externally active *if $deg_{H_{j-1}}(v) < deg_C(v)$.*

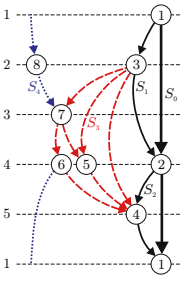


Fig. 3. Embedding a closed component

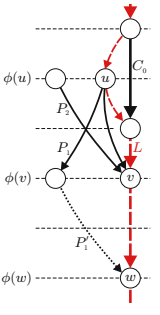


Fig. 4. Aborting case in findSubcomponent

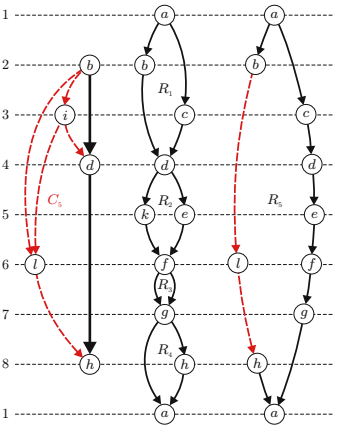


Fig. 5. Arranging component C_5

The closed component in Fig. 3 is split into the subcomponents $S_0, \ldots, S_4$ with S_0 consisting of the virtual edges $(1, 2)$ and $(2, 1)$. $E(S_1) = \{(1, 3), (3, 2)\}$ and $E(S_2) = \{(2, 4), (4, 1)\}$ hold. S_3 and S_4 consist of the dashed and dotted edges, respectively. $\text{upper}(S_3) = 3$ and $\text{lower}(S_3) = 4$ hold and S_3 is externally active because the nodes 6 and 7 are externally active. Thus 6 and 7 have to be placed on the left side of the embedding of S_3.

To compute a cyclic level planar embedding for a closed component C, we start with an embedding of H_0 for the cycle of virtual edges. We then repeat the following steps as long as there are edges to embed:

1. Find two (not necessarily different) nodes u and v on the left border of the embedding of H_{j-1} with unembedded outgoing and ingoing nodes, respectively s.t. no externally active nodes lie between u and v.
2. Find the subcomponent S_j with $\text{upper}(S_j) = u$ and $\text{lower}(S_j) = v$.
3. Try to embed the subcomponent to the left side of H_{j-1}, s.t. all externally active nodes appear on the left border.

We maintain a set NEXT_PAIRS of pairs of nodes to store the end nodes for possible next subcomponents to embed. We initialize NEXT_PAIRS with those virtual edges $e = (u, v) \in E(C)$, where u and v have unembedded outgoing and ingoing edges, respectively. We can now choose an arbitrary element $(u, v) \in$ NEXT_PAIRS and determine whether there really are paths from u to v.

findSubcomponent(u, v) tries to find the subcomponent S with upper$(S) = u$ and lower$(S) = v$ in time $\mathcal{O}(|E(S)| \log |E(S)|)$ as follows: We examine untraversed paths from u downwards and from v upwards by taking edges alternately. If the downwards phase finds a visited node, it starts again with the next highest node with unvisited outgoing edges to which a path from u and v has been found (thus priority queues and the logarithmic overhead are needed). The downwards phase aborts the current path if it runs below the lowest node with unvisited ingoing edges (at the latest v) or if no such node below the current path exists. The upwards phase is symmetric. We ensure that the starting node of a path of the downwards phase lies above the starting node of the upwards phase. If one phase follows a path that will not belong to S, the running time can be accounted to the path found by the other phase. If both phases follow such paths, Lemma 2 shows that a crossing is then inevitable.

The next step is to find an embedding for the subcomponent S. If span$(S) < k$, the subcomponent does not wrap around the center and we actually have the problem of finding a level planar embedding for $S' = S$. If span$(S) = k$, we create a level planarity problem instance S' by duplicating the level of upper$(S) =$ lower(S) and the node itself. One node (which we call upper(S') from now on) receives the outgoing edges and the other (lower(S')) the ingoing edges. In both cases we now have a level planarity problem instance with span$(S)+1$ levels. But we also have to ensure that all externally active nodes of S' lie on the left border of the embedding. (We show in Lemma 1 that C is not cyclic level planar if such an embedding does not exist.) To do so we add a node f to the level of lower(S') and connect all externally active nodes to f. We also add a node w below f and lower(S') and add the edges (f, w), (lower$(S'), w)$ and (upper$(S'), w)$ to obtain an *st-graph*. Therefore, again the last phase of the embedding algorithm for level planar graphs as described in [6] suffices. If the result is an embedding with all externally active nodes lying on the right border, we flip the embedding. If the level planarity testing algorithm fails, then S is not cyclic level planar and the algorithm aborts. If it succeeds, we add the embedding of S to the left side of the partial embedding of C.

As a last step we have to update the set NEXT_PAIRS. If the last so far embedded subcomponent S was not externally active, we follow the left border of the partial embedding of C from upper(S) upwards and search for the first externally active node e_1. Note that upper(S) itself can be externally active. If we do not find such a node, the component has been embedded completely. We also search from lower(S) downwards for the first externally active node e_2. If e_1 has unembedded outgoing edges and e_2 unembedded ingoing edges, we add (e_1, e_2) to NEXT_PAIRS. Otherwise we add a short cut edge (e_1, e_2) to the left border of the embedding to ensure that this path will not be traversed a second

time. Short cut edges are removed as the last step of this phase. If S is externally active, the same search is performed twice: from upper(S) and from lower(S) in each case upwards and downwards. Note that both searches can find the same pair of nodes which is added to NEXT_PAIRS only once.

In Fig. 3 NEXT_PAIRS is initialized with $\{(1,2),(2,1)\}$. Let $(1,2)$ be the first taken pair. After embedding S_1 both searches fail and after S_2 the pair $(3,4)$ is added. After treating S_3 both searches find the pair $(6,7)$ which is added once.

Definition 3. *We call a planar embedding of a subgraph of C* (cyclic level) planarity preserving *if it can be expanded to a planar embedding of C without changing the relative order in the N^- and N^+ lists if C is cyclic level planar.*

Lemma 1. *For a closed component the algorithm constructs only planarity preserving embeddings. In particular each partial embedding can be extended, s.t. new subcomponents are only added to the left outer face with all externally active nodes lying on the left side.*

Proof. We give the proof by induction over the number of embedded subcomponents j. If $j = 0$, the cycle of virtual edges has only one embedding. Suppose that the algorithm has embedded the subgraph H_{j-1} and is about to embed S_j. We have to show now that the chosen embedding for S_j is either the only one possible or does not influence the planarity preserving property.

Let L be the left border of H_{j-1} strictly between upper(S_j) and lower(S_j). The algorithm will embed S_j to the left of L. Let us assume for contradiction that it is possible to embed (a part of) S_j to the right of L (and the left of S_0). With induction assumption H_{j-1} is planarity preserving. Thus a face F in the current embedding H_{j-1} on the right side of L has to exist on which upper(S_j) and lower(S_j) lie. Note that the left border of F has to belong completely to an already embedded subcomponent S_i ($i < j$). But then S_j would be a part of S_i. A contradiction. If upper(S_j) and lower(S_j) both lie on S_0, then embedding S_j to the right of S_0 seems to be an option. But all subcomponents of one component have to be embedded on the same side as a component is connected.

The second possibility of choice regards the subcomponent itself: The subcomponent S_j can have several different embeddings but only the position of the externally active nodes are important. The algorithm places all these nodes on the left side of the subcomponent. Theoretically it would be possible to place an externally active node to the right side of the subcomponent or in the middle of it. But in both cases the path from the externally active node to either the level of upper(S_j) or lower(S_j) could then not be embedded in a planar way. □

See Fig. 3 as an example: Embedding S_1 or S_2 to the right side of S_0 is not possible as the component is connected. Embedding (a part of) S_3 to the right side of $L = \{2\}$ is not possible as no face between (S_1, S_2) and S_0 exists to which 3 and 4 belong. The nodes 6 and 7 must lie on the left outer face to be able to embed S_4.

Lemma 2. *If* findSubcomponent(u, v) *aborts while searching for a subcomponent S_j of a component C, C is not cyclic level planar.*

Proof. The subalgorithm aborts when it finds two disjoint paths P_1 and P_2, s.t. P_1 is a path from u downwards to level $\phi(v)$ but misses v and P_2 is a path from v upwards to level $\phi(u)$ but misses u. (This has to be the case if there is no path from u to v with length $\text{span}(u, v)$ at all.) Let L be the current left border of the partial embedding of C. The only possible way to embed P_1 and P_2 in a cyclic level planar way is to put one path to the left and one to the right of L. If v lies on C_0, P_2 has to be embedded on the same side as the rest of the component as the component is connected (see Fig. 4). The case for P_1 is analogous.

We will now show that P_1 has to be embedded on the left of L if u does not lie on C_0 (see Fig. 4). The proof for P_2 is analogous. Let P_1' be one shortest extension of P_1 to a node on L. If $\text{span}(P_1') > k$, then P_1' cannot be embedded on the right of L obviously. Otherwise let w be the end node of P_1'. Assume for contradiction that a face to the right of L exists to which u and w belong, s.t. P_1' fits into it. The left border of the face belongs completely to an already embedded subcomponent S_i $(i < j)$. But then P_1' would have been found by the same call of findSubcomponent as S_i. A contradiction. □

3.3 Arranging the Components

This phase combines ideas from [2] and [5]. We have to decide which components are embedded on the left side of C_0 and which to the right side. Therefore we first sort the components by increasing span. If there are several components with the same span, then the components with exactly two link nodes are considered last. The p components can be sorted by bucket sort in $\mathcal{O}(p + k)$.

We add one component at a time to the left side of C_0. Let C_i be the component to be embedded next. To do so it can be necessary to flip some already embedded components to the other side of C_0. C_i and all components which have overlapping spanned levels with C_i form a *rigid component* (see the data structure below) and are flipped simultaneously from now on. Among these a component C_j could be, s.t. C_j is embraced by the new component C_i in such a way, that C_j could lie on both sides of C_0. In this case we now decide on which side C_j will lie relative to C_i, too, by choosing an arbitrary side.

Definition 4. *A rigid component R is a recursive data structure consisting of a main component called* $\text{component}(R)$ *and all other already constructed rigid components $R_1, \ldots, R_r$ with overlapping levels with* $\text{component}(R)$*. For each R_i a flag $o_i \in \{LEFT, RIGHT\}$ is stored, which indicates on which side R_i lies relative to* $\text{component}(R)$*, which is assumed to lie on the left side of C_0.* $\text{rigidComponents}(R) = \{(R_1, o_1), \ldots, (R_r, o_r)\}$ *stores this information. R also has two nodes* $\text{upper}(R)$ *and* $\text{lower}(R)$*, which are its upper and lower end nodes. We define* $\text{span}(R) := \text{span}(\text{upper}(R), \text{lower}(R))$*. Furthermore, R stores four pointers to the nodes under* $\text{upper}(R)$ *and over* $\text{lower}(R)$ *on either side of the border called* border pointers *and one pointer to the next rigid component* $\text{next}(R)$*.*

Each node on the border has pointers to the predecessor and successor node on the border. Note that when traversing a border we can determine on which side of which rigid component we are when we encounter a border pointer.

Definition 5. *Let $C_1, \ldots, C_p$ be all components sorted in the way described above and let C_i be the component to be embedded next. We call a node $v \in V(C_0)$ pertinent if $v \in \text{link}(C_i)$. We call $v \in V(C_0)$ strongly externally active if the following conditions hold:*

1. *There exists a component $C_j (j > i)$ s.t. $v \in \text{link}(C_j)$.*
2. *The node v is not the upper end node of a rigid component.*
3. *If C_i is an open component, then v is strictly between $\phi(\text{upper}(C))$ and $\phi(\text{lower}(C))$.*

Note that all nodes satisfying the first condition have to be reachable from one outer face after embedding C_i. But only nodes for which the second and third condition hold can possibly be enclosed by C_i. A node can be pertinent and strongly externally active at the same time. When embedding C_5 in Fig. 1 the nodes b, d and h are pertinent and c and e are strongly externally active.

We do not embed a component into another one. Therefore, we have to make sure that all strongly externally active nodes of C_0 stay reachable from at least one outer face. To embed a component C_i all pertinent nodes have to be reachable from the same side. After embedding an open component no node of C_0 between the levels of $\text{upper}(C_i)$ and $\text{lower}(C_i)$ (both not included) is reachable from the left side. After embedding a closed component no node of C_0 is reachable from the left side.

We now have to flip all R_j which have overlapping spanned levels with C_i s. t. all pertinent nodes on the border of R_j lie on the left side and all strongly externally active nodes on the border of R_j lie on the right side.

Just looking through both sides of the border for each such rigid component could yield a quadratic running time. But for each R we examine, one side of the border of R will be enclosed by the component C_i we want to embed and will therefore never be traversed again. So we can always traverse the shorter of the two sides of the border of R. Further, we do not really flip a rigid component, but store how often it should be flipped only.

For each edge $e = (u, v) \in E(C_0)$ we initialize a rigid component R with $\text{upper}(R) = u$, $\text{lower}(R) = v$. Let C_i be the component to be embedded next. We have to find a path from $\text{upper}(C_i)$ to $\text{lower}(C_i)$ along the borders of the rigid components on which all pertinent nodes lie. Additionally no strongly externally active nodes may lie on this path.

We use a method searchOneSide(source, target) if we already know which side of a rigid component we have to follow and searchBothSides(source, target) if we do not. We then follow both borders alternately. We search for paths connecting each consecutive pair of nodes (u, v) in $\text{link}(C_i)$: We call searchOneSide(u, v) if u lies on the border of a rigid component and searchBothSides(u, v) if u is the upper end of a rigid component. If searchOneSide(u, v) finds a strongly externally active node, the algorithm aborts (even if the node is v). If it finds the node v, a path from u to v has been found. If we reach $\phi(v)$ or a level below without finding v, we know we are on the wrong side of the border and the algorithm aborts. If it finds the lower end node w of a rigid component, searchBothSides(w, v) is called.

Due to performance restrictions in searchBothSides(u, v) we have to make sure to completely traverse the side which will be enclosed only. We start taking alternately one edge of the left and one of the right side of the rigid component R whose upper end is u. If $\phi(v)$ is below $\phi(\text{lower}(R))$, then we just have to find one side which has no strongly externally active nodes on it. Thus, if we reach the lower end of the rigid component R from one side, we take this side and start searchBothSides(upper(next$(R)), v)$. If we reach a strongly externally active node, we only follow the other side from now on. If we encounter a strongly externally active node on the other side as well, the algorithm aborts.

If $\phi(v)$ lies between $\phi(\text{upper}(R))$ and $\phi(\text{lower}(R))$, we additionally test the following: If we find the node v from one side, we have found the path. Again, if we miss v on one side, we know that v lies on the other side and we do not follow this path any further.

If the algorithm did not abort, we have now found a path from upper(C_i) to lower(C_i) on the sides of the rigid components. Due to the border pointers we know for each R_i which side we have used (except a special case discussed below). We can therefore test if upper(C_i) and lower(C_i) lie on different sides of the same rigid component and abort if it is the case. We now create a new rigid component R containing all visited R_i. Let R_{lower} be the rigid component s.t. lower(C_i) is the lower end node of R_{lower} or lies on the border of it. (We can identify R_{lower} if we encountered a border pointer). We set lower(R) := lower(R_{lower}) and upper(R) for upper(R_{upper}) accordingly. We set component(R) := C_i and next(R) := next(R_{lower}). For each R_j between upper(R) and lower(R) we add $(R_j, LEFT)$ to rigidComponents(R) if the left side of R_j was used and $(R_j, RIGHT)$, otherwise. At last we have to construct the left and right border of R. The left border is the path from upper(R) to upper(C_i) with the left border of C_i and the path from lower(C_i) to lower(R). The right border is the path from upper(R) to lower(R) which was not used. To build this path we do not have to run through this path completely. It suffices to update the pointers at the connections between two (old) rigid components. If we have to merge a rigid component with itself, we only maintain a pseudo rigid component with two cyclic lists for the borders from there on.

One special case remains: Let C_i be the component to be embedded next and all nodes in link(C_i) lie on the same side of the same rigid component and no strongly externally active nodes lie between them. We then know that C_i can be embedded. But we do not know to which side, as we have not encountered a border pointer. So we do not construct a new rigid component for C_i, but update the border only.

Lemma 3. *If the search for the paths on the borders of the rigid components aborts for a component C_i, C_i cannot be added in a planar way.*

Proof. In this case the link nodes of C_i cannot be reached from the same side and so C_i cannot be added in the current situation. We show that all decisions the arranging algorithm makes are planarity preserving. Choosing an arbitrary side of a rigid component if both sides are not strongly externally active cannot have an influence on later components. In the chosen order of the components

embedding a component C_i to the inner side of an already embedded component C_j is possible only if both are open and have the same upper and lower end nodes. If C_j has more than 2 link nodes, C_i cannot be embedded on the inner side. C_i having more than two link nodes and C_j having exactly two is not possible due to our sorting. In the last case C_i and C_j have exactly two link nodes. This cannot happen as then C_i could still be embedded on the outer side of C_j. □

After all components have been processed, the rigidComponents lists form a set of trees. For each R we count with d how often the value $RIGHT$ is stored on the path from the root of its tree to R. If d is odd, we know we have to embed component(R) to the right otherwise to the left. In the end, we go through the list of components and embed it to the calculated side. If we find a component for which we do not know the side, we embed it to the side on which its link nodes lie.

Figure 5 shows the situation of embedding C_5: On the left the component C_5 is shown with bold virtual edges. In the middle we see the current rigid components. The algorithm starts with searchOneSide(b, d) and finds d by using the left side of R_1. Then searchBothSides(d, h) is called and both sides of R_2 are searched. The right is not followed below e as e is strongly externally active. But the left side can be used and searchBothSides(f, h) is called which will choose, e. g., the left side of R_3. The call of searchBothSides(g, h) will find h on the right side. So we have found a path from b to h. In the new rigid component R_5 we set rigidComponents$(R_5) = \{(R_1, LEFT), (R_2, LEFT), (R_3, LEFT), (R_4, RIGHT)\}$. R_5 is shown on the right in Fig. 5. Now we embed C_6 and search for paths from c to e and from e to c. We obtain a pseudo rigid component.

4 Correctness and Running Time

Theorem 1. *Let G be a strongly connected cyclic k-level graph. G is cyclic level planar if and only if* embedCyclicLevelPlanar(G) *does not abort.*

Proof. If embedCyclicLevelPlanar(G) does not abort, the returned embedding H is cyclic level planar as due to the construction of the algorithm no crossing can be inserted.

We will now show that in all cases in which embedCyclicLevelPlanar(G) aborts, G is not cyclic level planar. The cases are:

- span$(C_0) > k$: Such a (simple) cycle can obviously not be cyclic level planar.
- embedLevelPlanar(C_i') fails for a component C_i: As C_i' is a subgraph of G in which paths on C_0 are replaced by virtual edges, G cannot be planar then.
- findSubcomponent(u, v) fails: see Lemma 2.
- embedLevelPlanar(S') fails for a subcomponent S: In this case S does not have a level planar embedding with all externally active nodes on the same border which we have shown to be necessary in Lemma 1.
- searching the borders of the rigid components fails for a component C_i: see Lemma 3. □

Theorem 2. embedCyclicLevelPlanar(G) *runs in time* $\mathcal{O}(|V|\log|V|)$.

Proof. Finding a cycle and splitting G into its components can be done in $\mathcal{O}(|V|)$.

Next we consider the embedding of a component C_i. If $\text{span}(C_i) \leq k$, a linear time level planarity embedding algorithm is applied. Otherwise the following holds: Maintaining the set NEXT_PAIRS can be done in linear time. The method findSubcomponent runs in time $\mathcal{O}(|E(S)|\log|E(S)|)$ for a subcomponent S. Finding an embedding for S is again done by a level planarity algorithm. This yields a running time of $\mathcal{O}(|V(C_i)|\log|V(C_i)|)$ for embedding a component C_i. If findSubcomponent aborts, its running time is in $\mathcal{O}(|V|\log|V|)$.

Deciding for each C_i to which side of C_0 it will be embedded is possible in $\mathcal{O}(|P|)$ with P being the path on the partial embedding of G which will be enclosed by C_i. This and the arranging itself is therefore possible in $\mathcal{O}(|V|)$. □

5 Conclusion

In this paper we claim that the problem of finding a planar embedding can be solved in $\mathcal{O}(|V|^3)$ for proper cyclic k-level graphs and in $\mathcal{O}(|V|^6)$ for non-proper graphs by an algorithm presented in [4]. Our main result is a new algorithm which solves the testing and embedding problem for non-proper and strongly connected graphs in time $\mathcal{O}(|V|\log|V|)$.

The major open problem is to improve this algorithm to linear running time and to find algorithms with (near) linear running time for a larger class of graphs. Combining the problems for radial level planarity and cyclic level planarity would yield drawings on a torus and could also be a topic for further research.

References

1. Bachmaier, C., Brandenburg, F.J., Forster, M.: Radial level planarity testing and embedding in linear time. Journal of Graph Algorithms and Applications 9(1), 53–97 (2005)
2. Boyer, J., Myrvold, W.: On the cutting edge: Simplified $\mathcal{O}(n)$ planarity by edge addition. Journal of Graph Algorithms and Applications 8(3), 241–273 (2004)
3. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10(1), 41–51 (1990)
4. Healy, P., Kuusik, A.: Algorithms for multi-level graph planarity testing and layout. Theoretical Computer Science 320(2–3), 331–344 (2004)
5. Hopcroft, J.E., Tarjan, R.E.: Efficient planarity testing. Journal of the ACM 21(4), 549–568 (1974)
6. Jünger, M., Leipert, S.: Level planar embedding in linear time. Journal of Graph Algorithms and Applications 6(1), 67–113 (2002)
7. Kaufmann, M., Wagner, D.: Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
8. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics 11(2), 109–125 (1981)

Practical Level Planarity Testing and Layout with Embedding Constraints

Martin Harrigan* and Patrick Healy

Department of Computer Science and Information Systems,
University of Limerick, Ireland
{martin.harrigan, patrick.healy}@ul.ie

Abstract. We describe a practical method to test a leveled graph for level planarity and provide a level planar layout of the graph if the test succeeds, all in quadratic running-time. Embedding constraints restricting the order of incident edges around the vertices are allowed.

1 Introduction

A *leveled* graph, or one whose vertices have predetermined y-coordinates, is level planar if and only if it can be drawn in the plane satisfying the predetermined y-coordinates using straight-line edges without any edge crossings. We improve a previous test for level planarity [4] so that it provides a level planar layout of the graph if the test succeeds in quadratic running-time. We also handle a family of embedding constraints [3] that restrict the order of incident edges around the vertices.

There exists a level planarity testing and layout algorithm with linear running-time [2,5,6]. However, it is quite complicated, involving iterative updating of a set of PQ-trees, graph augmentations and an embedding algorithm for general planar graphs [1]. Our method, while requiring quadratic running-time is much simpler to understand and implement and can be naturally extended to handle embedding constraints.

2 Preliminaries

A *leveling* of a graph $G = (V, E)$ is a surjective mapping $\phi : V \rightarrow \{1, 2, \ldots, k\}$ such that $\phi(u) \neq \phi(v), \forall \{u, v\} \in E$. A leveling partitions the vertex set $V = V_1 \cup V_2 \cup \ldots \cup V_k$ such that $V_i = \phi^{-1}(i)$ and the edge set $E = E_1 \cup E_2 \cup \ldots \cup E_{k-1}$ such that $E_i \subseteq V_i \times V_{i+1}$. A leveling is *proper* if $\forall \{u, v\} \in E : |\phi(u) - \phi(v)| = 1$. In the following we assume all levelings to be proper.

The *vertex-exchange graph* or *ve-graph* $\mathcal{VE}(G, \phi) = (\mathcal{V}, \mathcal{E})$ of a graph $G = (V, E)$ with leveling ϕ is a graph with vertex set $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \ldots \cup \mathcal{V}_k$ where

* Supported by the Irish Research Council for Science, Engineering and Technology.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 62–68, 2007.

$\mathcal{V}_j = \{\langle u, v\rangle | u, v \in V_j\}$ and edge set $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \cup \mathcal{E}_{k-1}$ where $\mathcal{E}_j = \{\{\langle t, w\rangle, \langle u, v\rangle\} | \{t, u\}, \{w, v\} \in E_j, \langle t, w\rangle \in \mathcal{V}_j, \langle u, v\rangle \in \mathcal{V}_{j+1}\}$.

In other words, $\mathcal{VE}(G, \phi)$ is constructed by taking the distinct pairs of vertices on the same level of G as vertices of $\mathcal{VE}(G, \phi)$ and joining two vertices in $\mathcal{VE}(G, \phi)$ whenever two pairs from the four corresponding vertices in G are joined by independent edges (see Fig. 1).

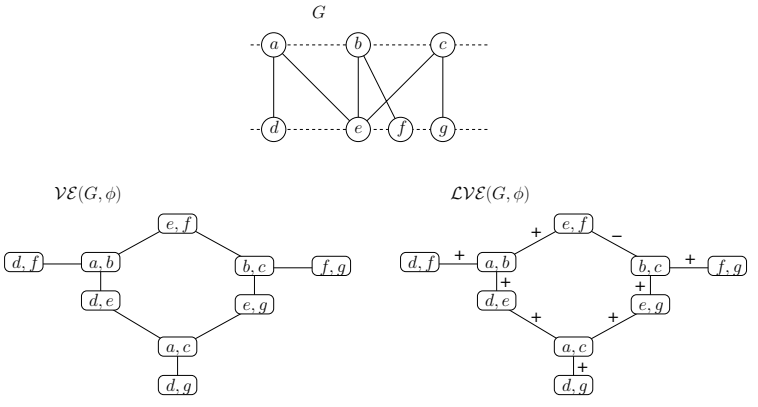


Fig. 1. A leveled graph, its ve-graph, and lve-graph

We augment the ve-graph $\mathcal{VE}(G, \phi) = (\mathcal{V}, \mathcal{E})$ with an edge labeling $\lambda : \mathcal{E} \to \{\text{'+'}, \text{'-'}\}$ to produce the *labeled vertex-exchange graph* or *lve-graph* $\mathcal{LVE}(G, \phi)$ as follows. Choose some initial level layout $\mathcal{L}$ of G. For every edge $e \in \mathcal{E}$ we set $\lambda(e) =$ '-' if the corresponding edges in G cross and $\lambda(e) =$ '+' if they do not (see Fig. 1).

3 Testing and Layout

The *ve-operation* $\texttt{ve}(\langle u, v\rangle)$ switches the labeling of every edge incident to $\langle u, v\rangle$ in $\mathcal{LVE}(G, \phi)$, *i.e.* '-' becomes '+' and '+' becomes '-'. This loosely corresponds to exchanging the position of the vertices u and v in the level layout $\mathcal{L}$ of G.

Theorem 1. [4]
A graph G with leveling ϕ is level planar if and only if there exists some sequence of ve-operations that removes all '-'-labeled edges from $\mathcal{LVE}(G, \phi)$, or equivalently, $\mathcal{LVE}(G, \phi)$ does not contain a cycle with an odd number of '-'-labeled edges.

Level planarity can therefore be tested in quadratic running-time using a simple depth-first search (DFS) traversal on the lve-graph. However, if the test succeeds, we need to solve the 3-*cycle problem* in order to produce a level planar layout within the same asymptotic running-time.

3.1 The 3-Cycle Problem

Suppose three vertices $\langle u, v\rangle$, $\langle v, w\rangle$ and $\langle u, w\rangle$ in some lve-graph representing the three vertices u, v and w in some leveled graph are not in the same connected component of the lve-graph. If we perform ve-operations to remove all the '-'-labeled edges using a DFS traversal then it may arise that $u \prec v \prec w \prec u$ where $\prec$ denotes the required order of the vertices along their respective level. Clearly, this is impossible.

Consider the leveled graph G and its lve-graph $\mathcal{LVE}(G, \phi)$ in Fig. 2. Note that $\langle e, f\rangle$, $\langle f, g\rangle$ and $\langle e, g\rangle$ are in different connected components of $\mathcal{LVE}(G, \phi)$. Suppose we begin the DFS traversal at $\langle e, f\rangle$, then visit $\langle b, c\rangle$ and perform $\texttt{ve}(\langle b, c\rangle)$ so that $e \prec f$. Suppose we continue the DFS traversal at $\langle f, g\rangle$, then visit $\langle a, b\rangle$ and perform $\texttt{ve}(\langle a, b\rangle)$ so that $f \prec g$. Now, suppose we continue the DFS traversal at $\langle a, c\rangle$, then visit $\langle e, g\rangle$ and perform $\texttt{ve}(\langle e, g\rangle)$ so that $g \prec e$. We have removed all '-'-labeled edges. However, $e \prec f \prec g \prec e$.

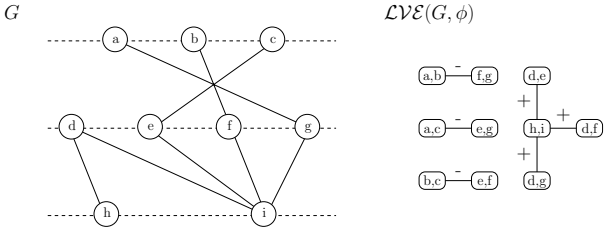


Fig. 2. An instance of the 3-cycle problem

A book-keeping solution for every such triple of vertices in the level graph has been suggested [4]. Every time we perform a ve-operation that results in a single remaining vertex being constrained by another two, we queue that vertex and its respective connected component to be traversed once we are finished with all the vertices in the current connected component. Unfortunately there are $\mathcal{O}(|V|^3)$ such triples leading to cubic running-time.

Randerath *et al.* [7] have reduced the level planarity testing and layout problems to satisfiability problems. They reduced the level planarity testing problem to a 2-SAT problem that is quadratic in the size of the leveled graph. However, if a level planar layout is required, the solution must be 'enhanced' to avoid the 3-cycle problem (or, in their terminology, to satisfy the transitivity clauses). They show that such an enhancement is always possible but they do not show how to find it within the same asymptotic running-time.

3.2 The 3-Cycle Solution

We solve the 3-cycle problem using a combination of a DFS traversal and a level-by-level traversal. In this case the DFS traversal (Algorithm 1), given an initial level layout $\mathcal{L}$, constructs a mapping π that tells us the required *relative* order of the vertices. For example, suppose $\langle u, v\rangle$ and $\langle w, x\rangle$ are in the same connected

component of the ve-graph. The algorithm may decide that $\pi(\langle u, v\rangle) = [v, u]$ and $\pi(\langle w, x\rangle) = [x, w]$. In other words, $v \prec u \Leftrightarrow x \prec w$ and $u \prec v \Leftrightarrow w \prec x$. It does not change the initial layout $\mathcal{L}$. If it returns `false` then it has found a cycle with an odd number of '-'-labeled edges (Lines 19 and 22) and the leveled graph is not level planar. If it returns `true` then it is level planar and we can proceed to the level-by-level traversal to produce a level planar layout.

Algorithm 1: `dfsTraversal`

```
   Input: VE(G, φ), L, π passed by reference
 1 visited ← {};
 2 Initialize a stack S;
 3 foreach C ∈ connectedComponents(VE(G, φ)) do
 4     Choose some vertex ⟨u, v⟩ in C;
 5     if u ≺_L v then π(⟨u, v⟩) ← [u, v];
 6     else π(⟨u, v⟩) ← [v, u];
 7     push(S, ⟨u, v⟩, false);
 8 while S not empty do
 9     ⟨u, v⟩, value ← pop(S);
10     visited(⟨u, v⟩) ← true;
11     foreach ⟨w, x⟩ ∈ neighbors(⟨u, v⟩) do
12         if λ({⟨u, v⟩, ⟨w, x⟩}) = '+' then
13             p ← [w, x], q ← [x, w];
14             if visited(⟨w, x⟩) = false then push(S, ⟨w, x⟩, value);
15         else
16             p ← [x, w], q ← [w, x];
17             if visited(⟨w, x⟩) = false then push(S, ⟨w, x⟩, ¬value);
18         if (w ≺_L x ∧ value) ∨ (x ≺_L w ∧ ¬value) then
19             if visited(⟨w, x⟩) = true ∧ π(⟨w, x⟩) ≠ p then return false;
20             π(⟨w, x⟩) ← p;
21         else
22             if visited(⟨w, x⟩) = true ∧ π(⟨w, x⟩) ≠ q then return false;
23             π(⟨w, x⟩) ← q;
24 return true;
```

The level-by-level traversal (Algorithm 2) decides on the *absolute* order of the vertices so that everything remains consistent between the connected components of the ve-graph. The vertices of the ve-graph are grouped by the level of the vertices in the leveled graph they represent. We traverse the vertices in each group $1, \ldots, k$. Within each group the vertices are traversed in descending order according to the distance between the vertices they represent in the layout $\mathcal{L}$ of the leveled graph. This traversal proceeds left-to-right and top-to-bottom along the dotted lines in Fig. 3(b). It is controlled by `rows`($\mathcal{L}$) (the number of dotted lines), `cols`($\mathcal{L}, i$) (the number of vertices on the ith dotted line) and `vertexAt`($\mathcal{L}, i, j$) (the vertex at the jth position on the ith dotted line). Note

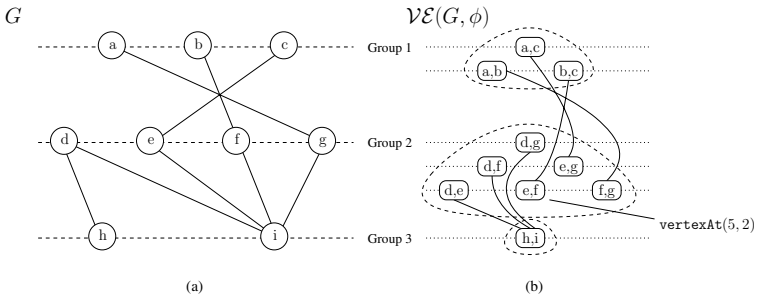


Fig. 3. (a) The initial level layout $\mathcal{L}$ of G and (b) the level-by-level traversal of $\mathcal{VE}(G,\phi)$)

that the position of some vertices may change after each update of $\mathcal{L}$. However, the traversal proceeds in this direction irrespectively.

On traversing a vertex $\langle u,v\rangle$, we count the number of neighbors it has in the previous group that are joined by '+' (`plusCnt`) and '-'-labeled edges (`minusCnt`). If $\langle u,v\rangle$ belongs to a hitherto unvisited connected component C of the ve-graph then we mark C as visited and record whether $u \prec_{\mathcal{L}} v$ or $v \prec_{\mathcal{L}} u$. If C has been previously visited and `plusCnt`+`minusCnt` $= 0$ then we use π to decide whether or not we need to exchange u and v in $\mathcal{L}$. Let $\langle w,x\rangle$ be the first vertex visited in C. Our test `match`$(\mathcal{L},\pi,visited,C,\langle u,v\rangle)$ returns `true` if any of the following are true:

- $\pi(\langle w,x\rangle) = visited(C)$ and $u \prec_{\mathcal{L}} v$ and $\pi(\langle u,v\rangle) = [u,v]$
- $\pi(\langle w,x\rangle) = visited(C)$ and $v \prec_{\mathcal{L}} u$ and $\pi(\langle u,v\rangle) = [v,u]$
- $\pi(\langle w,x\rangle) \neq visited(C)$ and $u \prec_{\mathcal{L}} v$ and $\pi(\langle u,v\rangle) = [v,u]$
- $\pi(\langle w,x\rangle) \neq visited(C)$ and $v \prec_{\mathcal{L}} u$ and $\pi(\langle u,v\rangle) = [u,v]$

and `false` otherwise. Finally, if `minusCnt` > 0 (and hence `plusCnt` $= 0$), we exchange u and v in $\mathcal{L}$ and proceed to the next vertex in the traversal. Note that it cannot arise that `minusCnt` > 0 and `plusCnt` > 0 since this would mean the presence of a cycle with an odd number of '-'-labeled edges. Algorithm 2 makes $\mathcal{L}$ level planar one level at a time. To keep its running-time linear in the size of the ve-graph we determine the label of an edge (Line 8 of Algorithm 2) dynamically instead of precomputing a lve-graph and updating the labels every time we change $\mathcal{L}$.

4 Embedding Constraints

Embedding constraints, restricting the order of incident edges around the vertices, can be specified using *constraint trees* [3]. A constraint tree $T(v)$ (see Fig. 4) is an ordered tree rooted at v where inner vertices (except the root), called *c-vertices*, impose embedding constraints on their children and the leaves are v's incident edges between v and the next successive level. There are three types of c-vertices:

Algorithm 2: `levelByLevelTraversal`

```
   Input: VE(G, φ), L passed by reference, π
 1 visited ← {};
 2 for i from 1 to rows(L) do
 3     for j from 1 to cols(L, i) do
 4         ⟨u, v⟩ ← vertexAt(L, i, j);
 5         plusCnt ← 0, minusCnt ← 0;
 6         foreach w ∈ neighbors(⟨u, v⟩) do
 7             if row(w) < i ∨ (row(w) = i ∧ col(w) <= j) then
 8                 if λ({⟨u, v⟩, w}) = '+' then plusCnt ← plusCnt + 1;
 9                 else minusCnt ← minusCnt + 1;
10         C ← connectedComponent(VE(G, φ), ⟨u, v⟩);
11         if C ∉ domain(visited) then
12             if u ≺_L v then visited(C) ← [u, v];
13             else visited(C) ← [v, u];
14         else if plusCnt + minusCnt = 0 ∧ ¬match(L, π, visited, C, ⟨u, v⟩) then
15             Exchange u and v in L;
16         if minusCnt > 0 then
17             Exchange u and v in L;
```

- *gc-vertices* (grouping) allow children to be arbitrarily permuted,
- *mc-vertices* (mirroring) allow children to be reversed, and
- *oc-vertices* (orientation) fix the order of children in $T(v)$.

$T(v)$ constrains the order of its leaves (from left to right) and thus v's incident edges between v and the next successive level. For every constraint tree $T(v)$, we expand the graph G by replacing the subgraph induced by v and its successive neighboring vertices with $T(v)$ and assigning the c-vertices to sub-levels as shown in Fig. 4. Note that this new leveling may not be proper so we add 'dummy' vertices to the long edges at the bottommost sub-level and only allow edge crossings involving the long edges to occur between this sub-level and the next successive level. This suffices since constraint trees do not share c-vertices. The ve-graph of the expanded leveled graph is constructed as before, treating c-vertices and sub-levels as regular vertices and levels respectively and with the following additions:

- A loop is added to every vertex in the ve-graph whose corresponding vertices in the leveled graph are children of the same oc-vertex.
- Every subgraph of the ve-graph induced by vertices whose corresponding vertices in the leveled graph are children of the same mc-vertex is made biconnected.

Constraining the order of v's incident edges between v and the previous level is handled analogously. When choosing an initial layout $\mathcal{L}$ of the expanded leveled graph it must satisfy the constraint trees. To begin with, the additional loops

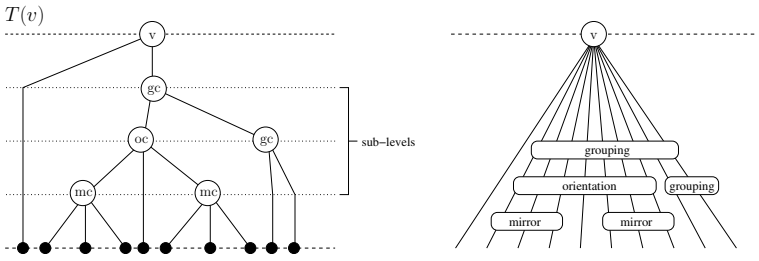


Fig. 4. A constraint tree $T(v)$ and the corresponding restriction on the order of v's incident edges between v and the next successive level

and edges are labeled '+'. The additional loops preserve the order of the children of the oc-vertices while the additional edges allow the order of the children of the mc-vertices to be reversed altogether or none at all. The level planarity testing and layout algorithms remain unchanged. If we find a level planar layout we then contract the expanded graph back to G.

References

1. Chiba, N., Nishizeki, T., Abe, S., Ozawa, T.: A Linear Algorithm for Embedding Planar Graphs using PQ-Trees. Journal of Comp. and Sys. Sci. 30(1), 54–76 (1985)
2. Di Battista, G., Nardelli, E.: Hierarchies and Planarity Theory. IEEE Trans. Sys., Man and Cybern. 18(6), 1035–1046 (1988)
3. Gutwenger, C., Klein, K., Mutzel, P.: Planarity Testing and Optimal Edge Insertion with Embedding Constraints. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 126–137. Springer, Heidelberg (2007)
4. Healy, P., Kuusik, A.: Algorithms for Multi-Level Graph Planarity Testing and Layout. Theor. Comp. Sci. 320(2-3), 331–344 (2004)
5. Heath, L., Pemmaraju, S.: Recognizing Leveled-Planar DAGs in Linear Time. In: Brandenburg, F.J. (ed.) GD 1995. LNCS, vol. 1027, pp. 300–311. Springer, Heidelberg (1996)
6. Jünger, M., Leipert, S.: Level Planar Embedding in Linear Time. Journal of Graph Alg. and App. 6(1), 67–113 (2002)
7. Randerath, B., Speckenmeyer, E., Boros, E., Hammer, P., Kogan, A., Makino, K., Simeone, B., Cepek, O.: A Satisfiability Formulation of Problems on Level Graphs. Technical report 40-2001, Rutgers Center for Operations Research, Rutgers University (2001)

Minimum Level Nonplanar Patterns for Trees

J. Joseph Fowler* and Stephen G. Kobourov*

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

Abstract. Minimum level nonplanar (MLNP) patterns play the role for level planar graphs that the forbidden Kuratowksi subdivisions K_5 and $K_{3,3}$ play for planar graphs. We add two MLNP patterns for trees to the previous set of tree patterns given by Healy *et al.* [4]. Neither of these patterns match any of the previous patterns. We show that this new set of patterns completely characterizes level planar trees.

1 Introduction

Level graphs model hierarchical relationships. A level drawing has all vertices in the same level with the same y-coordinates and has all edges strictly y-monotone. Level planar graphs have level drawings without edge crossings. Hierarchies are special cases in which every vertex is reachable via a y-monotone path from a source in the top level. Planar graphs are characterized by forbidden subdivisions of K_5 and $K_{3,3}$ by Kuratowksi's Theorem [5]. The counterpart of this characterization for level planar graphs proposed by Healy, Kuusik, and Liepert [4] are the minimum level nonplanar (MLNP) patterns. These are minimal obstructing subgraphs with a set of level assignments that force one or more crossings.

Di Battista and Nardelli [1] provided three level nonplanar patterns for hierarchies (HLNP patterns); cf. Fig. 2. Healy *et al.* adapted these HLNP patterns to MLNP patterns for level graphs. However, the completeness of their characterization was based on the claim that all MLNP patterns must contain a

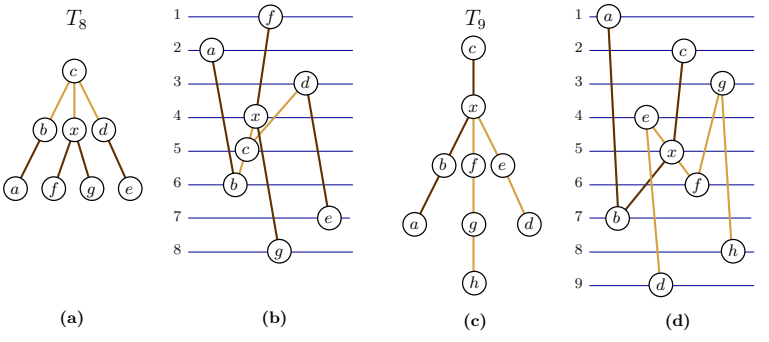


Fig. 1. Labelings preventing the forbidden ULP trees T_8 and T_9 from being level planar

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

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 69–75, 2007.

HLNP pattern. We provide a counterexample to this claim based on the level nonplanar assignment for the forbidden tree T_9 used by Estrella *et al.* [2] to characterize the set of unlabeled level planar (ULP) trees; cf. Fig. 1. Healy *et al.* provide two of the MLNP patterns, P_1 and P_2, for trees that are also HLNP patterns; cf. Fig. 3(a) and (b). We provide two more MLNP patterns, P_3 and P_4 for level nonplanar trees; cf. Fig. 3(c) and (d) using our counterexample.

2 Preliminaries

A *k-level graph* $G(V, E, \phi)$ on n vertices has *leveling* $\phi : V \to [1..k]$ where every $(u, v) \in E$ either has $\phi(u) < \phi(v)$ if G is directed or $\phi(u) \neq \phi(v)$ if G is undirected. This leveling partitions V into $V_1 \cup V_2 \cup \cdots \cup V_k$ where the *level* $V_j = \phi^{-1}(j)$ and $V_i \cap V_j = \varnothing$ if $i \neq j$. A *proper level graph* only has *short edges* in which $\phi(v) = \phi(u) + 1$ for every $(u, v) \in E$. Edges spanning multiple levels are *long*. A *hierarchy* is a proper level graph in which every vertex $v \in V_j$ for $j > 1$ has at least one incident edge $(u, v) \in E$ to a vertex $u \in V_i$ for some $i < j$.

A *path* p is a non-repeating ordered sequence of vertices $(v_1, v_2, \ldots, v_t)$ for $t \geq 1$. Let $\text{MIN}(p) = \min\{\phi(v) : v \in p\}$, $\text{MAX}(p) = \max\{\phi(v) : v \in p\}$, and $\mathcal{P}(i, j) = \{p : p \text{ is a path where } i \leq \text{MIN}(p) < \text{MAX}(p) \leq j\}$ are the paths between levels V_i and V_j. A *linking path*, or *link*, $L \in \mathcal{L}(i, j)$ is a path $x \rightsquigarrow y$ in which $i = \text{MIN}(L) = \phi(x)$ and $\text{MAX}(L) = \phi(y) = j$, and $\mathcal{L}(i, j) \subseteq \mathcal{P}(i, j)$ are all paths linking the *extreme levels* V_i and V_j. A *bridge* b is a path $x \rightsquigarrow y$ in $\mathcal{P}(i, j)$ connecting links $L_1, L_2 \in \mathcal{L}(i, j)$ in which $b \cap L_1 = x$ and $b \cap L_2 = y$.

A *level drawing* of G has all of its *level-j* vertices in the j^{th} level V_j placed along the *track* $\ell_j = \{(x, k - j) \,|\, x \in \mathbb{R}\}$, and each edge $(u, v) \in E$ is drawn as a continuous strictly y-monotone sequence of line segments. A level drawing drawn without edge crossings shows that G is *level planar*. A *pattern* is a set of level nonplanar graphs sharing structural similarities. Removing any edge from the underlying graph matching a *minimum level nonplanar* (MLNP) *pattern* gives a level planar graph. A *hierarchy level nonplanar* (HLNP) *pattern* is a level nonplanar pattern in which every matching graph is a hierarchy. The next theorem gives the set of the three distinct HLNP patterns.

Theorem 1. *[Di Battista and Nardelli [1]] A hierarchy $G(V, E, \phi)$ on k levels is level planar if and only if there does not exist three paths $L_1, L_2, L_3 \in \mathcal{L}(i, j)$ linking levels V_i and V_j for $1 \leq i < j \leq k$ where one of the following hold:*

(P_A) *L_1, L_2, and L_3 are completely disjoint and pairwise connected by bridges b_1, b_2, b_3 where $b_1 \cap L_2 = b_2 \cap L_1 = b_3 \cap L_1 = \varnothing$; cf. Fig. 2(a).*

(P_B) *L_1 and L_2 share a path $C = L_1 \cap L_2$ from $p \in V_i \cup V_j$ where $L_1 \cap L_3 = L_2 \cap L_3 = \varnothing$ are connected by bridges b_1 from L_1 to L_3 and b_2 from L_1 to L_3 such that $b_1 \cap L_2 = b_2 \cap L_1 = \varnothing$; cf. Fig. 2(b).*

(P_C) *L_1 and L_2 share a path $C_1 = L_1 \cap L_2$ from $p \in V_i$ and L_2 and L_3 share a path $C_2 = L_2 \cap L_3$ from $q \in V_j$ such that $C_1 \cap C_2 = \varnothing$. Bridge b connects L_1 and L_3 where $b \cap L_2 = b \cap C_1 = b \cap C_2 = \varnothing$; cf. Fig. 2(c).*

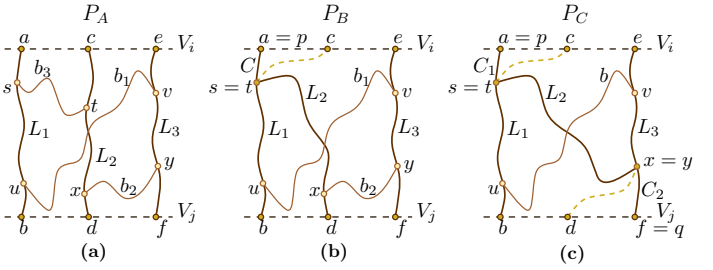


Fig. 2. The three patterns characterizing hierarchies

3 MLNP Patterns for Trees

Theorem 2. *A level tree $T(V, E, \phi)$ on k levels is minimum level nonplanar if*

(1) there are three disjoint paths $L_1, L_2, L_3 \in \mathcal{L}(i, j)$ for $1 \leq i < j \leq k$ where P_A of Theorem 1 applies and the union of the three bridges $b_1 \cup b_2 \cup b_3$ forms a subdivided $K_{1,3}$ subtree S with vertex c of degree 3 where either
(P_1) $c \in V_i$ (or V_j) and there is a leaf of S in V_j (or V_i) as in Fig. 3(a) or
(P_2) one leaf of S is in V_i and another leaf of S is in V_j as in Fig. 3(b), or

(2) there are four paths $L_1, L_2, L_3, L_4 \in \mathcal{L}(i, j)$ for $1 \leq i < j \leq k$ where $L_1 \cap L_4 = \varnothing$, $L_1 \cap L_2 \in V_j$ (or V_i) and $L_3 \cap L_4 \in V_i$ (or V_j) where $L_1 \cup L_2$ and $L_3 \cup L_4$ form paths with both endpoints in V_i and V_j (or V_j and V_i), resp., and there exist levels V_l and V_m for some $i < l < m < j$ in which either L_2 or L_3 consists of subpaths $C_1 \in \mathcal{L}(i, m)$, $C_2 \in \mathcal{L}(l, m)$, and $C_3 \in \mathcal{L}(l, j)$ where either
(P_3) $L_2 \cap L_3 = x$ where $l \leq \phi(x) \leq m$ as in Fig. 3(c), or
(P_4) $L_2 \cap L_3$ is path $x \rightsquigarrow y$ where $l \leq \{\phi(x), \phi(y)\} \leq m$ and $L_2 = c \rightsquigarrow x \rightsquigarrow y \rightsquigarrow b$ where $c \in V_i$ (or V_j) and $b \in V_j$ (or V_i) as in Fig. 3(d).

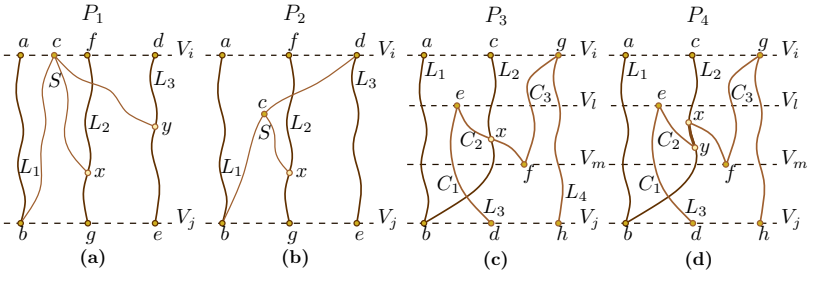


Fig. 3. Four MLNP patterns for trees

Proof. P_1 and P_2 are MLNP given they match T1 and T2 of Healy *et al.* The argument in [2] used by Estrella *et al.* to show T_9 is level nonplanar generalizes for P_3 and P_4. To see that P_3 is minimal (P_4 is similar), we try the seven distinct ways of removing an edge; cf. Fig. 4. In each case crossings can be avoided. □

The proof of Theorem 15 of Healy *et al.* [4] argues that every MLNP pattern must match some HLNP pattern. We show why this argument fails for P_3.

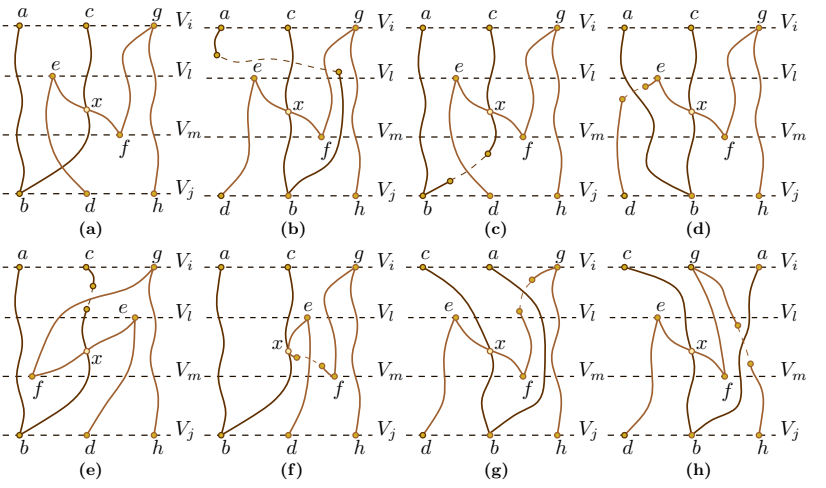


Fig. 4. The seven cases of deleting an edge from pattern P_3 in (a)

Lemma 3. *P_3 augmented to form a hierarchy has a subtree matching P_2.*

Proof. Fig. 5 shows the highlighted subtrees that match P_2 when P_3 is augmented to form a hierarchy. However, P_2 does not match P_3 by Theorem 2. □

The next lemma gives the minimal conditions for a MLNP tree pattern.

Lemma 4. *A level nonplanar tree $T(V, E, \phi)$ on k levels contains three disjoint paths $L_1, L_2, L_3 \in \mathcal{L}(i, j)$ linking levels V_i and V_j for $1 \leq i < j \leq k$ with bridges b_1 from L_1 to L_2 and b_2 from L_2 to L_3 with $x = b_1 \cap L_2$ and $y = b_2 \cap L_2$ so that either (P_α) $x = y$, (P_β) $L_2 = c \rightsquigarrow y \rightsquigarrow x \rightsquigarrow d$, or (P_γ) $L_2 = c \rightsquigarrow x \rightsquigarrow y \rightsquigarrow d$ hold where $c \in V_i$ and $d \in V_j$ as in Fig. 6(a), (b), (c).*

Proof. Assume that P is an MLNP pattern between levels V_i and V_j in which $|i - j|$ is minimum and there are at most two disjoint paths $L_1, L_2 \in \mathcal{L}(i, j)$.

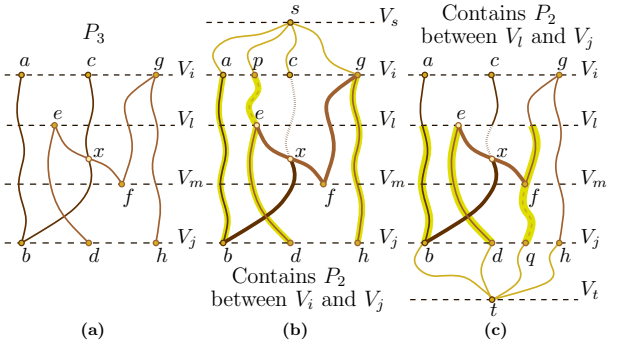


Fig. 5. Augmenting P_3 in (a) from above (b) and below (c) to form hierarchies

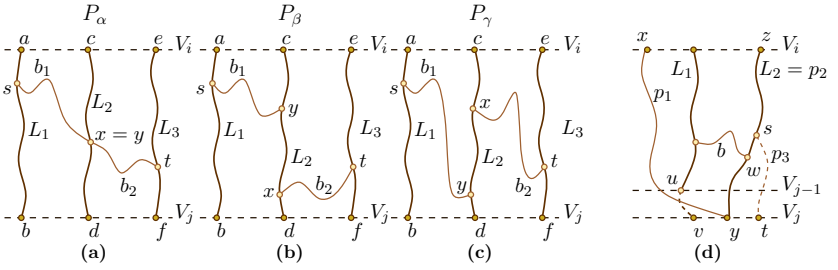


Fig. 6. Minimal patterns for Lemma 4

There could be at most one bridge b joining L_1 and L_2 without forming a cycle. Let w be the endpoint of b in L_2. Let P' be $P - (u, v)$ where (u, v) is the short edge connecting L_1 to V_j in which $v \in V_j$. In order for P to be MLNP, there must exist two linking paths $p_1, p_2 \in \mathcal{L}(i, j)$ in P' with endpoints $x, z \in V_i$ and common endpoint $y \in V_j$ such that for any level planar embedding of P', u is contained in the region bounded by p_1, p_2 and the track ℓ_i; cf. Fig. 6(d). Assume w.l.o.g. that L_2 is p_2. In order for p_1 not to be embeddable on the other side of p_2 (allowing edge (u, v) to be drawn in P without crossing), there must be a path p_3 from s in L_2 to $t \in V_j$ in which s lies between z and w blocking this direction. Then there are at least three disjoint paths in P in $\mathcal{L}(i, j)$: p_1, L_1 and the path $z \rightsquigarrow s \rightsquigarrow t$, contradicting our assumption of there only being two.

Let $L_1, L_2, L_3 \in \mathcal{L}(i, j)$ be three disjoint paths. At least one of the three paths, say it is L_2, must be joined by bridges b_1 and b_2 to the other two paths L_1 or L_3, respectively, or P would be disconnected contradicting the minimality of P. If $b_1 \cap b_2$ form a nonempty path, then $b_1 \cup b_2$ would form a subtree homeomorphic to $K_{1,3}$, yielding pattern P_1 or P_2 of Theorem 2. Thus, b_1 and b_2 can share at most one vertex as in P_α of Fig. 6(a). Otherwise there must have been endpoints $x = b_1 \cup L_2$ and $y = b_2 \cup L_2$ along the path $c \rightsquigarrow d$ forming L_2 where either y proceeds x as in P_β of Fig. 6(b) or x proceeds y as in P_γ of Fig. 6(c). □

We next show that P_4 is easily derived from P_3.

Lemma 5. *P_4 is the only distinct MLNP pattern for trees that can be formed from P_3 (by splitting the degree-4 vertex) not containing a subtree matching P_2.*

Proof. Fig. 7 shows the three ways the degree-4 vertex of P_3 can be split into two degree-3 vertices. Two contain subtrees that match P_2. □

Finally we complete our characterization for level nonplanar trees.

Theorem 6. *A level tree T is level nonplanar if and only if T has a subtree matching one of the minimum level nonplanar patterns P_1, P_2, P_3, or P_4.*

Proof Sketch: We sketch proof for the simplest case here; the full proof can be found in [3]. Once a MLNP pattern P is augmented to form a hierarchy, one of the HLNP patterns must apply. Since this augmentation does not introduce a

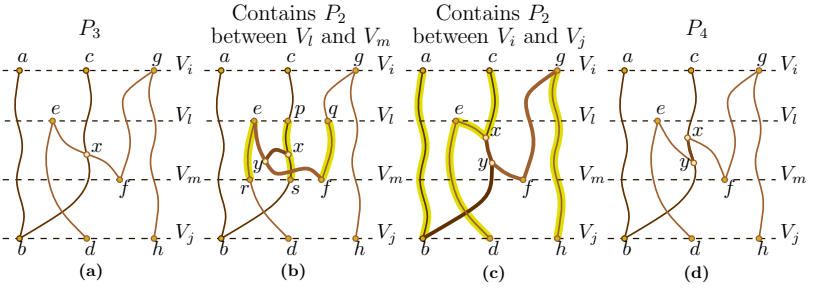


Fig. 7. The three ways of splitting the degree-4 vertex of P_3 into two vertices of degree 3

cycle between levels V_i and V_j, either pattern P_1 or P_2 must match a subtree of the augmented pattern by Lemma 5 of [4].

Assume there is a MLNP tree pattern P containing P_α of Lemma 4 that does not match P_1 or P_2. We consider the simplest case of how the bridges of P_α in P could spans levels between V_i and V_j. We augment P to form a hierarchy to illustrate how either P must match P_1 or P_2 or contain a cycle.

Suppose that a bridge of P_α in P is not strictly y-monotone. Then P could either have a bend at e in level V_l in one bridge or a bend at f in level V_m in the other as in Fig. 8(a) for some $i < l < m < j$. Each bend would require augmentation to a path from the source when forming a hierarchy from above or below as was the case with P_3 in Fig. 5.

We augment P with a path $p \rightsquigarrow e$ from V_i to V_l to form P', a hierarchy, that must match P_1 or P_2. We observe that between levels V_i and V_m, we have four linking paths. A third bridge $u \rightsquigarrow v$ must be present in P' that is part of a subtree S homeomorphic to $K_{1,3}$. Fig. 8(b) gives one such example. While P' matches P_2 between levels V_i and V_m, we see that between levels V_i and V_j, P must have had the cycle $u \rightsquigarrow v \rightsquigarrow e \rightsquigarrow b \rightsquigarrow u$, contradicting P being a tree pattern. By inspection, any other placement of $u \rightsquigarrow v$ to connect three of the four linking paths to form P_1 or P_2 similarly implies a cycle in P.

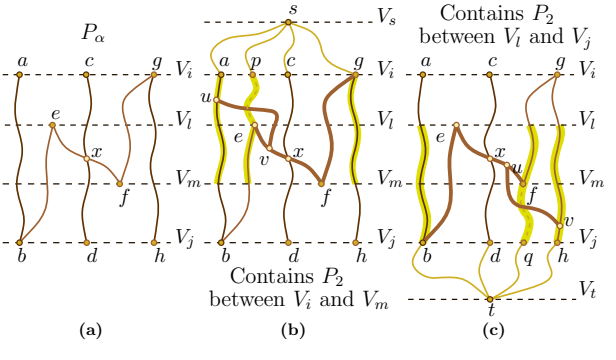


Fig. 8. Examples of pattern P_α in (a) being augmented to form a hierarchy in (b) and (c)

Hence, P cannot contain any more edges than those of P_α without matching P_1 or P_2. We observe that P_α consists of two paths sharing a common vertex x. Given the minimality of P in minimizing $|i-j|$, one path has both endpoints in V_i with one vertex in V_j that can be split into linking paths $L_1, L_2 \in \mathcal{L}(i,j)$. Similarly, the other has both endpoints in V_j with one vertex in V_i that can also be split into the linking paths $L_3, L_4 \in \mathcal{L}(i,j)$. In P_3 of Fig. 8(a), L_1 is $a \rightsquigarrow b$, L_2 is $b \rightsquigarrow e \rightsquigarrow x \rightsquigarrow c$, L_3 is $d \rightsquigarrow x \rightsquigarrow f \rightsquigarrow g$, and L_4 is $g \rightsquigarrow h$.

For P to be level nonplanar, a crossing must be forced between these two paths. This is done by having L_2 or L_3 meet the condition of P_3 of three subpaths $C_1 \in \mathcal{L}(i,m)$ linking V_i to V_m, $C_2 \in \mathcal{L}(l,m)$ linking V_l to V_m, and $C_3 \in \mathcal{L}(l,j)$ linking V_l to V_j. This is not the case for P_α in Fig. 8(a) since the $x \rightsquigarrow c$ portion of L_2 does not reach level V_m, and the $x \rightsquigarrow d$ portion of L_3 does not reach level V_l. So for P not to match P_3, at least one subpath of both L_2 and L_3 from x to V_i or V_j must strictly monotonic as was the case in Fig. 8(a). However, in this case P can be drawn without crossings. This leaves P_3 as the only possibility of a MLNP pattern matching P_α that does not match P_1 or P_2. □

References

1. Di Battista, G., Nardelli, E.: Hierarchies and planarity theory. IEEE Trans. Systems Man Cybernet 18(6), 1035–1046 (1988)
2. Estrella-Balderrama, A., Fowler, J.J., Kobourov, S.G.: Characterization of unlabeled level planar trees. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 367–369. Springer, Heidelberg (2007)
3. Fowler, J.J., Kobourov, S.G.: Minimum level nonplanar patterns for trees. Technical Report TR07-04, University of Arizona (2007), ftp://ftp.cs.arizona.edu/reports/2007/TR07-04.pdf
4. Healy, P., Kuusik, A., Leipert, S.: A characterization of level planar graphs. Discrete Math. 280(1-3), 51–63 (2004)
5. Kuratowski, C.: Sur les problèmes des courbes gauches en Topologie. Fundamenta Mathematicae 15, 271–283 (1930)

Straight-Line Orthogonal Drawings of Binary and Ternary Trees*

Fabrizio Frati

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

Abstract. In this paper we provide upper and lower bounds on the area requirement of straight-line orthogonal drawings of n-node binary and ternary trees. Namely, we show algorithms for constructing order-preserving straight-line orthogonal drawings of binary trees in $O(n^{1.5})$ area, straight-line orthogonal drawings of ternary trees in $O(n^{1.631})$ area, and straight-line orthogonal drawings of complete ternary trees in $O(n^{1.262})$ area. As far as we know, the ones we present are the first algorithms achieving sub-quadratic area for these problems. Further, for upward order-preserving straight-line orthogonal drawings of binary trees and for order-preserving straight-line orthogonal drawings of ternary trees we provide $\Omega(n^2)$ area lower bounds, that we also prove to be tight.

1 Introduction

The design of algorithms for constructing orthogonal and straight-line drawings of binary and ternary trees, that are trees whose maximum degree is bounded by three and four, respectively, has attracted considerable research efforts in the Graph Drawing community. Orthogonal and straight-line planar drawings are easily readable by the viewer and hence they are among the most studied drawing standards. When dealing with orthogonal or straight-line tree drawings, it is common to consider area minimization as an important aesthetic requirement to satisfy. The study of area minimization for binary and ternary tree drawings has been motivated by VLSI circuits design and it is still attractive for the sake of rendering acyclic relationships on a screen limited by a finite resolution rule. Nevertheless, the beauty of some combinatorial and geometric open problems concerning area minimization of straight-line and orthogonal drawings of trees justifies their study even looking at them from a purely theoretical point of view.

Almost thirty years ago, Valiant proved in [12] that every n-node ternary tree admits a $\Theta(n)$ area orthogonal drawing. Such a result was strengthened in [5], where Dolev and Trickey proved that ternary trees admit $\Theta(n)$ area order-preserving orthogonal drawings. A $\Theta(n \log \log n)$ optimal bound for upward orthogonal drawings of binary trees was proved by Garg et al. in [6], while in [9] Kim showed that $\Theta(n \log n)$ area is an optimal bound for upward orthogonal

* Work partially supported by MUR under Project MAINSTREAM Algorithms for Massive Information Structures and Data Streams.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 76–87, 2007.

drawings of ternary trees. Concerning the area requirement of planar straight-line drawings, Garg and Rusu proved in [8] that linear area suffices for bounded degree trees, while $\Theta(n \log n)$ area is asymptotically optimal if the drawing is required to be upward and order-preserving [7].

Drawings that are simultaneously straight-line and orthogonal provide extremely high readability of the combinatorial structure of a tree, and hence it is a serious lack in the literature that only few results concerning area minimization of straight-line orthogonal drawings of binary and ternary trees are known. Chan et al. in [1], and Shin et al. in [11] have shown that $O(n \log \log n)$ area suffices for straight-line orthogonal drawings of binary trees. Further, it has been shown in [3,1] that binary trees admit upward straight-line orthogonal drawings in $O(n \log n)$ area. Such an area bound is worst-case optimal, as proved in [1].

In this paper we present the following results: (i) order-preserving straight-line orthogonal drawings of binary trees can be constructed in $O(n^{1.5})$ area (Section 3); (ii) upward order-preserving straight-line orthogonal drawings of binary trees require (and can be realized in) $\Omega(n^2)$ area (Section 3); (iii) straight-line orthogonal drawings of ternary trees can be constructed in $O(n^{1.631})$ area (Section 4); (iv) order-preserving straight-line orthogonal drawings of ternary trees require (and can be realized in) $\Omega(n^2)$ area (Section 4); (v) straight-line orthogonal drawings of complete ternary trees can be constructed in $O(n^{1.262})$ area (Section 5); and (vi) there exist ternary trees for which the minimum side of any straight-line orthogonal drawing is $\Omega(n^{0.438})$ and, for complete ternary trees, such a bound is tight (Section 5).

Table 1. Summary of the best known area bounds for straight-line orthogonal drawings of binary and ternary trees. For complete trees the order-preserving column is not considered, since such trees are symmetric. Straight-line orthogonal upward drawings of ternary trees cannot generally be constructed.

	Upward	*Order-preserving*	*Upper Bound*	*Ref.*	*Lower Bound*	*Ref.*
Complete Binary	✓		$O(n)$	[3]	$\Omega(n)$	trivial
Complete Binary			$O(n)$	[10]	$\Omega(n)$	trivial
Binary	✓	✓	$O(n^2)$	[3]	$\Omega(n^2)$	Th. 1
Binary	✓		$O(n \log n)$	[3,1]	$\Omega(n \log n)$	[1]
Binary		✓	$O(n^{1.5})$	Th. 2	$\Omega(n)$	trivial
Binary			$O(n \log \log n)$	[1,11]	$\Omega(n)$	trivial
Complete Ternary	✓		**non-drawable**			
Complete Ternary			$O(n^{1.262})$	Th. 6	$\Omega(n)$	trivial
Ternary	✓	✓	**non-drawable**			
Ternary	✓		**non-drawable**			
Ternary		✓	$O(n^2)$	Th. 4	$\Omega(n^2)$	Th. 3
Ternary			$O(n^{1.631})$	Th. 5	$\Omega(n)$	trivial

2 Preliminaries

We assume familiarity with trees and their drawings (see also [4]).

A *rooted tree* T is a tree with one distinguished node, called *root* and denoted by $r(T)$. In the following we assume that binary and ternary trees are rooted

at any node of degree at most two and three, respectively. A *spine* in T is a path connecting $r(T)$ to a leaf. A *double-spine* in T is a path connecting two leaves and passing through $r(T)$. A tree is *ordered* if an order of the children of each node is specified. For an ordered binary tree we talk about *left* and *right child*. For an ordered ternary tree we talk about *left*, *middle*, and *right child*. The subtrees rooted at the left, middle, and right child of a node u are the *left*, *middle*, and *right subtree* of u, respectively. The subtree of a given tree rooted at node u is denoted by $T(u)$. Removing a path $\mathcal{P}$ from a tree disconnects the tree into connected components. The ones containing children of nodes in $\mathcal{P}$ are subtrees of $\mathcal{P}$. If the tree is ordered, then each component is a left, middle, or right subtree of $\mathcal{P}$, depending on whether the root of such subtree is a left, middle, or right child of a node in $\mathcal{P}$. We denote by $|T|$ the number of nodes in a tree T. The *heaviest* tree in a set of trees is the one with the greatest number of nodes. A *complete tree* is such that all non-leaf nodes have the same degree and all spines have the same number of nodes, called the *height* of the tree.

A *straight-line orthogonal grid drawing* of a binary or ternary tree is a mapping of its nodes to distinct points with integer coordinates and of its edges to horizontal or vertical segments between such points. A drawing is *planar* if no two segments cross, but, possibly, at common end-points. In the following we use *SO-drawing* as short for straight-line orthogonal planar grid drawing. An SO-drawing is *upward* if every node is drawn not below its children. An SO-drawing Γ is *order-preserving* if, for every node u, the segments connecting u to its left child, middle child, right child and parent appear in Γ in this order around u. When we talk about order-preserving drawings, we suppose that trees are ordered. Consider an SO-drawing Γ of a rooted tree T. Denote by l the vertical half-line starting at $r(T)$ and directed upward. Then Γ has the *top visibility property* if no node, but for $r(T)$, is placed on l and no edge crosses l. Denote by r the horizontal line through $r(T)$. Then Γ has the *side visibility property* if no node, but for $r(T)$, is placed on r and no edge crosses r. The *width* (*height*) of a drawing is the number of vertical (horizontal) grid lines intersecting it. The *area* of a drawing is its height multiplied by its width.

3 Straight-Line Orthogonal Order-Preserving Drawings of Binary Trees

First, we show that order-preserving upward SO-drawings of binary trees generally require quadratic area. Such a bound is matched by an $O(n^2)$ upper bound obtained by using the well-known *h-v layout* (see, e.g., [3]).

Theorem 1. *There exists an n-node binary tree T requiring $\Omega(n^2)$ area in any upward order-preserving SO-drawing.*

Proof: Assume $n \equiv 0 \bmod 6$. Tree T is composed of (see Fig. 1(a)): (i) an $n/6$-node spine $C_1 : (m_0 = r(T), m_1, \ldots, m_{\frac{n}{6}-2}, m_{\frac{n}{6}-1})$, with m_i left child of m_{i-1}, for $1 \leq i \leq \frac{n}{6} - 1$; (ii) an $n/6$-node spine $C_2 : (p_0 = r(T), p_1, \ldots, p_{\frac{n}{6}-2}, p_{\frac{n}{6}-1})$, with p_i right child of p_{i-1}, for $1 \leq i \leq \frac{n}{6} - 1$; (iii) the right child m_i^r of each

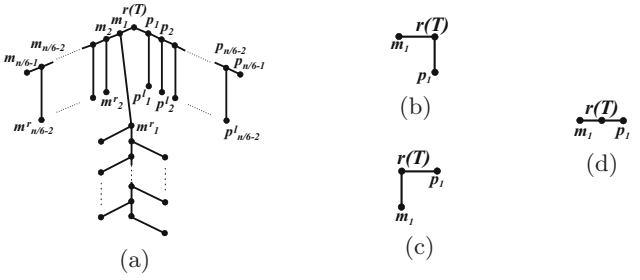


Fig. 1. (a) Tree T providing the lower bound of Theorem 1. (b)-(c)-(d) Possible placements of $r(T)$ and its children.

node m_i of C_1, with $1 \leq i \leq \frac{n}{6} - 2$; (iv) the left child p_i^l of each node p_i of C_2, with $1 \leq i \leq \frac{n}{6} - 2$; (v) a path C_3 of $n/6 + 3$ nodes, alternating between right and left children, such that one end-vertex of C_3 is m_1^r; and (vi) $n/6 + 3$ leaves attached to C_3, alternating between left and right children.

Consider any upward order-preserving SO-drawing Γ of T. In [6] it is shown that C_3 and its attached leaves require $\Omega(n)$ height in any upward order-preserving drawing. Consider the relative position of $r(T)$ and its children in Γ. Three are the cases; either m_1 is to the left of $r(T)$ and p_1 is below $r(T)$ (see Fig. 1.b), or m_1 is below $r(T)$ and p_1 is to the right of $r(T)$ (see Fig. 1.c), or m_1 is to the left of $r(T)$ and p_1 is to the right of $r(T)$ (see Fig. 1.d). Suppose m_1 is to the left of $r(T)$. We prove by induction that each node m_i of C_1, with $1 \leq i \leq \frac{n}{6} - 1$, is drawn at least one unit to the left of its parent. The claim holds in the base case by the assumption that m_1 is to the left of $m_0 = r(T)$. If m_i is to the left of m_{i-1}, then the edges from m_i to its children are drawn towards the left and the bottom. Since the drawing is order-preserving, m_i^r must be below m_i and m_{i+1} to the left of m_i. So each node m_i, with $1 \leq i \leq \frac{n}{6} - 1$, is drawn at least one unit to the left of its parent, implying a linear lower bound on the width of Γ. If m_1 is not to the left of $r(T)$ then p_1 is to the right of $r(T)$ and a similar argument shows that each node p_i, with $1 \leq i \leq \frac{n}{6} - 1$, is at least one unit to the right of its parent, again implying a linear lower bound on the width of Γ. Hence both the height and the width of Γ are $\Omega(n)$. □

Now we turn to non-upward drawings, showing that sub-quadratic area suffices for order-preserving SO-drawings:

Theorem 2. *Any n-node binary tree T admits an $O(n^{1.5})$ area order-preserving SO-drawing.*

Proof: We describe an inductive algorithm constructing an order-preserving SO-drawing Γ of T satisfying the side visibility property. If $n = 1$, then Γ is trivially constructed. Suppose $n > 1$. Select a double-spine $\pi = (u_k, u_{k-1}, \ldots, u_1, u_0 = r(T) = v_0, v_1, \ldots, v_m)$ in T. How to choose π is discussed later. Denote by p_i the non-spine child of a node $u_i \in \pi$ and by q_j the non-spine child of a node $v_j \in \pi$.

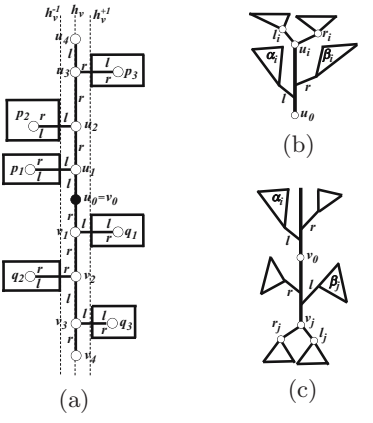


Fig. 2. Illustrations for the algorithm in the proof of Theorem 2. Left (right) edges are labeled l (r). Label l (r) inside a subtree shows the direction of the edge from the root to its left (right) child.

Recursively construct drawings $\Gamma(p_i)$ of $T(p_i)$ and $\Gamma(q_j)$ of $T(q_j)$ satisfying the side visibility property, for $1 \leq i < k$ and $1 \leq j < m$. Let h_v, h_v^{-1}, and h_v^1 be vertical grid lines with h_v^{-1} (h_v^1) one unit to the left (to the right) of h_v. Draw $r(T)$ on h_v. For $i = 1, 2, \ldots, k-1$, if p_i is the left child of u_i rotate $\Gamma(p_i)$ of π and place it so that the rightmost vertical line intersecting it is h_v^{-1} and with the lowest horizontal line intersecting it one unit above the highest horizontal line intersecting $\Gamma(p_{i-1})$ or u_{i-1}; otherwise (p_i is the right child of u_i), place $\Gamma(p_i)$ so that the leftmost vertical line intersecting it is h_v^1 and with the lowest horizontal line intersecting it one unit above the highest horizontal line intersecting $\Gamma(p_{i-1})$ or u_{i-1}. Draw u_i on h_v on the same horizontal line of its already drawn child (or one unit above the highest horizontal line intersecting $\Gamma(p_{i-1})$ or u_{i-1} if no child of u_i has been drawn). Draw u_k on h_v one unit above the highest horizontal line intersecting $\Gamma(p_{k-1})$ or u_{k-1}. For $j = 1, 2, \ldots, m-1$, if q_j is the right child of v_j rotate $\Gamma(q_j)$ of π and place it so that the rightmost vertical line intersecting it is h_v^{-1} and with the highest horizontal line intersecting it one unit below the lowest horizontal line intersecting $\Gamma(q_{j-1})$ or v_{j-1}; otherwise (q_j is the left child of v_j), place $\Gamma(q_j)$ so that the leftmost vertical line intersecting it is h_v^1 and with the highest horizontal line intersecting it one unit below the lowest horizontal line intersecting $\Gamma(q_{j-1})$ or v_{j-1}. Draw v_j on h_v on the same horizontal line of its already drawn child (or one unit below the lowest horizontal line intersecting $\Gamma(q_{j-1})$ or v_{j-1} if no child of v_j has been drawn). Draw v_m on h_v one unit below the lowest horizontal line intersecting $\Gamma(q_{m-1})$ or v_{m-1} (see Fig. 2.a).

It's easy to see that the constructed drawing Γ is an order-preserving SO-drawing satisfying the side visibility property. Let's analyze the area requirement of Γ. Concerning its height, there is at least one node of T on each horizontal grid line intersecting Γ, hence the height of Γ is $O(n)$. Denote by $W(T)$ the width of the drawing constructed by the described algorithm when its input is binary tree T. Let also $W(n) = \max\{W(T)\}$ over all binary trees T with n

nodes. Since all subtrees drawn to the left (to the right) of π are aligned on their right side (on their left side) and since $W(n)$ is a non-decreasing function of n, then $W(n) = W(n_l) + W(n_r) + 1$, where n_l (n_r) is the number of nodes in the heaviest subtree drawn to the left (to the right) of π. To get a good bound for $W(n)$ we need to carefully choose π. A technique similar to the one we present was introduced in [2] for selecting (single) spines. π is composed of two spines $\mathcal{U} = (u_0, u_1, \ldots, u_k)$ and $\mathcal{V} = (v_0, v_1, \ldots, v_m)$. Spine $\mathcal{U}$ is iteratively selected as follows: $u_0 = r(T)$, u_1 is the left child of u_0. Denote by l_i and by r_i the left and right child of u_i, respectively. Denote also by α_i and by β_i the heaviest left subtree and the heaviest right subtree of path $(u_1, \ldots, u_{i-1})$ (see Fig. 2.b). If $|\alpha_i| + |T(r_i)| \leq |\beta_i| + |T(l_i)|$ then set $u_{i+1} = l_i$, otherwise set $u_{i+1} = r_i$. Spine $\mathcal{V}$ is iteratively selected as follows: $v_0 = r(T)$, v_1 is the right child of u_0. Denote by l_j and by r_j the left and right child of v_j, respectively. Denote by α_j the one between the heaviest right subtree of path $(v_1, \ldots, v_{j-1})$ and the heaviest left subtree of $\mathcal{U} \setminus u_0$ that has the greatest number of nodes. Denote also by β_j the one between the heaviest left subtree of path $(v_1, \ldots, v_{j-1})$ and the heaviest right subtree of $\mathcal{U} \setminus u_0$ that has the greatest number of nodes (see Fig. 2.c). If $|\alpha_j| + |T(l_j)| \leq |\beta_j| + |T(r_j)|$ then set $v_{j+1} = r_j$, otherwise set $v_{j+1} = l_j$. Similarly to [2], we get the following:

Lemma 1. *For any left subtree α of $\mathcal{U} \setminus u_0$ or right subtree α of $\mathcal{V} \setminus v_0$ and for any right subtree β of $\mathcal{U} \setminus u_0$ or left subtree β of $\mathcal{V} \setminus v_0$, $|\alpha| + |\beta| \leq n/2$.*

Proof: If α and β are both subtrees of $\mathcal{U} \setminus u_0$ or if are both subtrees of $\mathcal{V} \setminus v_0$, then the statement follows as in Lemma 4.1 of [2]. Otherwise, suppose α is a left subtree of $\mathcal{U} \setminus u_0$ and β is a left subtree of $\mathcal{V} \setminus v_0$. Let v_j be the parent of β's root. Denote by l_j and r_j the left and right child of v_j, respectively. Notice that $r_j = v_{j+1}$. Denote by α_j the one between the heaviest right subtree of path $(v_1, \ldots, v_{j-1})$ and the heaviest left subtree of $\mathcal{U} \setminus u_0$ that has the greatest number of nodes, and denote by β_j the one between the heaviest left subtree of path $(v_1, \ldots, v_{j-1})$ and the heaviest right subtree of $\mathcal{U} \setminus u_0$ that has the greatest number of nodes. By construction $|\alpha_j| + |T(l_j)| \leq |\beta_j| + |T(r_j)|$. Moreover, $|\alpha_j| + |T(l_j)| + |\beta_j| + |T(r_j)| \leq n$. Therefore, $\alpha_j + |T(l_j)| \leq n/2$. Since $\alpha \leq \alpha_j$ and $\beta = T(l_j)$, the statement follows. The case in which α is a right subtree of $\mathcal{V} \setminus v_0$ and β is a right subtree of $\mathcal{U} \setminus u_0$ is analogous. □

Selecting π as just described, we get $W(n) \leq \max_{n_1+n_2 \leq n/2} W(n_1) + W(n_2) + 1$. As already noticed in [2], by Hölder's inequality $n_1 + n_2 \leq n/2$ implies $\sqrt{n_1} + \sqrt{n_2} \leq \sqrt{n}$ and, by induction, $W(n) \leq c\sqrt{n} - 1$, for some constant c depending only on the values of $W(n)$ with n small. □

4 Straight-Line Orthogonal Drawings of Ternary Trees

In this section we consider SO-drawings of ternary trees. First, we show that if an order of the children of each node is fixed, then quadratic area is necessary in the worst case.

Theorem 3. *There exists an n-node ternary tree T requiring $\Omega(n^2)$ area in any order-preserving SO-drawing.*

Proof: Assume $n \equiv 4 \bmod 9$. Tree T is composed of (see Fig. 3.a): (i) a spine $C_1 : (m_0 = r(T), m_1, \ldots, m_{\frac{n-4}{9}}, m_{\frac{n+5}{9}})$, with m_1 left child of $r(T)$ and m_i middle child of m_{i-1}, for $i = 2, 3, \ldots, \frac{n+5}{9}$; (ii) a spine $C_2 : (p_0 = r(T), p_1, \ldots, p_{\frac{n-4}{9}}, p_{\frac{n+5}{9}})$, with p_i middle child of p_{i-1}, for $i = 1, 2, \ldots, \frac{n+5}{9}$; (iii) a spine $C_3 : (q_0 = r(T), q_1, \ldots, q_{\frac{n-4}{9}}, q_{\frac{n+5}{9}})$, with q_1 right child of $r(T)$ and q_i middle child of q_{i-1}, for $i = 2, 3, \ldots, \frac{n+5}{9}$; and (iv) a left and a right child for each node m_i, p_i, and q_i, for $i = 1, 2, \ldots, \frac{n-4}{9}$. Consider any order-preserving SO-drawing of C_1 and of the children of nodes m_i, with $1 \leq i \leq \frac{n-4}{9}$. Suppose that m_1 is to the left of m_0. Then, to preserve the order of the children of m_1, m_i is to the left of m_{i-1}, for $i = 2, 3, \ldots, \frac{n+5}{9}$. Analogously, if m_1 is to the right, above, or below m_0, then m_i is to the right, above, or below m_{i-1}, respectively, for $i = 2, 3, \ldots, \frac{n+5}{9}$. Such an argument applies to C_2 (to C_3), as well: If p_1 (q_1) is to the left, to the right, above, or below p_0 (q_0), then p_i (q_i) is to the left, to the right, above, or below p_{i-1} (q_{i-1}), respectively, for $i = 2, 3, \ldots, \frac{n+5}{9}$. Since $r(T)$ has three children, then at least one of them is above or below $r(T)$ and one of them is to the left or to the right of $r(T)$. Hence, any order-preserving SO-drawing of T has at least $\frac{n+5}{9} + 1$ height and width. □

The proved bound is tight, as shown in the following:

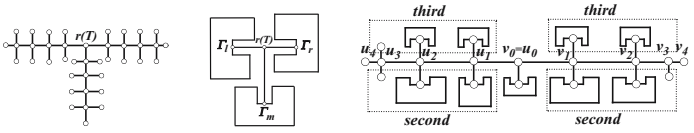
Theorem 4. *Any n-node ternary tree T admits an $O(n^2)$ area order-preserving SO-drawing.*

Proof: We show an inductive algorithm constructing an order-preserving SO-drawing Γ of T satisfying the top visibility property. If $n = 1$, then Γ is trivially constructed. Suppose $n > 1$. Let T_l, T_m, and T_r be the left, middle, and right subtree of $r(T)$. By induction, drawings Γ_l, Γ_m, and Γ_r satisfying the top visibility property can be constructed for T_l, T_m, and T_r, respectively. Draw $r(T)$ in the plane. Rotate Γ_l of $\pi/2$ in clockwise direction. Place Γ_l with the rightmost vertical line intersecting it one unit to the left of $r(T)$ and with $r(T_l)$ on the same horizontal line of $r(T)$. Rotate Γ_r of $\pi/2$ in counter-clockwise direction. Place Γ_r with the leftmost vertical line intersecting it one unit to the right of $r(T)$ and with $r(T_r)$ on the same horizontal line of $r(T)$. Place Γ_m with the highest horizontal line intersecting it one unit below the lowest horizontal line intersecting Γ_l or Γ_r and with $r(T_m)$ on the same vertical line of $r(T)$ (see Fig. 3.b). It's easy to see that Γ is an order-preserving SO-drawing satisfying the top visibility property. Since Γ has at least one node for each horizontal and vertical grid line intersecting it, then its height and its width are $O(n)$. □

For non-order-preserving drawings better bounds can be achieved:

Theorem 5. *Any n-node ternary tree T admits an $O(n^{1.631})$ area SO-drawing.*

Proof: We show an inductive algorithm that constructs an SO-drawing Γ of T satisfying the top visibility property. If $n = 1$, then Γ is trivially constructed.



(a) (b) (c)

Fig. 3. (a) Tree T requiring $\Omega(n^2)$ area in any order-preserving SO-drawing. (b) Illustration for the algorithm in Theorem 4. (c) Illustration for the algorithm in Theorem 5. Subtrees labeled by *second* or *third* are second or third heaviest subtrees, respectively.

Suppose $n > 1$. Select a double-spine $\pi = (u_k, u_{k-1}, \ldots, u_1, u_0 = r(T) = v_0, v_1, \ldots, v_m)$ in T such that: $T(v_1)$ is the heaviest subtree of $r(T)$; for $j = 2, 3, \ldots, m$, $T(v_j)$ is the heaviest subtree of v_{j-1}; $T(u_1)$ is the heaviest subtree of $r(T)$ different from $T(v_1)$; for $i = 2, 3, \ldots, k$, $T(u_i)$ is the heaviest subtree of u_{i-1}. For each node v_j in π, with $j = 0, 1, \ldots, m-1$, (for each node u_i in π, with $i = 1, 2, \ldots, k-1$), call *second heaviest subtree* $S(v_j)$ ($S(u_i)$) and *third heaviest subtree* $R(v_j)$ ($R(u_i)$), the subtrees of v_j (of u_i) different from $T(v_{j+1})$ (from $T(u_{i+1})$) with the greater and the smaller number of nodes, respectively.

Recursively construct a drawing satisfying the top visibility property of each subtree of π. Let h be an horizontal grid line. Draw $r(T)$ on h. Place the drawing of $R(v_0)$ with the highest horizontal line intersecting it one unit below h and with its root on the same vertical line of $r(T)$. For $j = 1, 2, \ldots, m-1$, rotate of π the drawing of $R(v_j)$. Place the drawing Γ_j^S of $S(v_j)$ and Γ_j^R of $R(v_j)$ so that the highest horizontal line intersecting Γ_j^S is one unit below h, the lowest horizontal line intersecting Γ_j^R is one unit above h, their roots are on the same vertical line, and the leftmost vertical line intersecting Γ_j^S or Γ_j^R is one unit to the right of the rightmost vertical line intersecting Γ_{j-1}^S, Γ_{j-1}^R, or v_{j-1}. Draw v_j on h on the same vertical line of its already drawn children (or draw v_j one unit to the right of the rightmost vertical line intersecting Γ_{j-1}^S, Γ_{j-1}^R, or v_{j-1} if no child of v_j has been drawn). Draw v_m on h one unit to the right of the rightmost vertical line intersecting Γ_{m-1}^S, Γ_{m-1}^R, or v_{m-1}. For path $(u_1, u_2, \ldots, u_k)$ and its subtrees, analogously construct a drawing in which the path lies on h, to the left of $r(T)$, and the $S(u_i)$'s and the $R(u_i)$'s are below and above h, respectively (see Fig. 3.c).

It's easy to see that Γ is an SO-drawing satisfying the top visibility property. Let's analyze the area of Γ. Since there is at least one node of T for each vertical grid line intersecting Γ, then its width is $O(n)$. Denote by $H(T)$ the height of the drawing constructed by the algorithm when its input is T. Let also $H(n) = \max\{H(T)\}$ over all ternary trees T with n nodes. Since all subtrees drawn above π (below π) are aligned on their bottom side (on their top side) and since $H(n)$ is a non-decreasing function of n, then $H(n) = H(n_a) + H(n_b) + 1$, where n_a (n_b) is the number of nodes in the heaviest subtree drawn above (below) of π. We claim (1) $n_a + n_b \leq 2n/3$. Namely, we have (2) $n_a \leq n_b$, (3) $n_b \leq n/2$, and (4) $n_a \leq n - 2n_b$. Inequality (2) holds since for each node w in π, $|R(w)| \leq |S(w)|$; inequality (3) follows from the fact that, for each node v_j and u_i in π,

$|S(v_j)| \leq |T(v_{j+1})|$ and $|S(u_i)| \leq |T(u_{i+1})|$; inequality (4) follows by considering any node v_j (u_i) in π and observing that $|S(v_j)| \leq |T(v_{j+1})|$ ($|S(u_i)| \leq |T(u_{i+1})|$) and that $|R(v_j)| + |S(v_j)| + |T(v_{j+1})| \leq n$ ($|R(u_i)| + |S(u_i)| + |T(u_{i+1})| \leq n$). By (3) we have $n_b = \frac{n}{2} - \alpha$, with $\alpha \geq 0$. If $\alpha \geq n/6$, then by (2) $n_b + n_a \leq 2(n/2-\alpha) \leq 2n/3$. If $\alpha < n/6$, by (4) we have $n_a \leq n - 2(n/2-\alpha) = 2\alpha$. Hence $n_b+n_a \leq n/2-\alpha+2\alpha = n/2+\alpha \leq 2n/3$. We claim that $n_a^{(1/\log_2 3)}+n_b^{(1/\log_2 3)} \leq n^{(1/\log_2 3)}$. Hölder's inequality states that (5) $\sum a_i b_i \leq (\sum a_i^p)^{\frac{1}{p}} (\sum b_i^q)^{\frac{1}{q}}$ for every p and q such that $1/p+1/q = 1$. Substituting into (5) the values $a_1 = n_a^{(1/\log_2 3)}$, $a_2 = n_b^{(1/\log_2 3)}$, $b_1 = b_2 = 1$, $1/p = 1/\log_2 3$, and $1/q = 1 - 1/\log_2 3$, we get $n_a^{(1/\log_2 3)} + n_b^{(1/\log_2 3)} \leq \left[\left(n_a^{(1/\log_2 3)}\right)^{\log_2 3} + \left(n_b^{(1/\log_2 3)}\right)^{\log_2 3}\right]^{(1/\log_2 3)} \cdot [1 + 1]^{(1-1/\log_2 3)} = (n_a + n_b)^{(1/\log_2 3)} \cdot 2^{(1-1/\log_2 3)} \leq (2n/3)^{(1/\log_2 3)} \cdot 2^{(1-1/\log_2 3)} = n^{(1/\log_2 3)} \left(2^{1/\log_2 3} \cdot 2 \cdot 2^{-1/\log_2 3}\right) / \left(3^{1/\log_2 3}\right) = n^{(1/\log_2 3)} \left(2/(3^{1/\log_2 3})\right) = n^{(1/\log_2 3)}$. Hence, $H(n) \leq \max_{n_1+n_2 \leq 2n/3} H(n_1)+H(n_2)+1$ by induction solves to $H(n) = c \cdot n^{(1/\log_2 3)} - 1$ for some constant c, depending only on the values of $H(n)$ with n small. It follows that $H(n) = O(n^{(1/\log_2 3)}) = O(n^{0.631})$. □

5 Straight-Line Orthogonal Drawings of Complete Ternary Trees

For complete ternary trees we present two algorithms constructing drawings with better area bounds than the ones obtained for general ternary trees. Let Γ_h be a drawing of a complete ternary tree T_h with height h. In both algorithms inductively suppose to have a drawing Γ_{h-1} of T_{h-1} satisfying the top visibility property, take three copies Γ'_{h-1}, Γ''_{h-1}, and Γ'''_{h-1} of Γ_{h-1}, rotate Γ'_{h-1} of $\pi/2$ and Γ''_{h-1} of $3\pi/2$ in clockwise direction. The algorithms differ in the geometric construction of Γ_h. In *Construction* 1 draw $r(T_h)$ on any horizontal grid line l_h. Place Γ'''_{h-1} with the highest horizontal line intersecting it one unit below l_h and with the root $r(T_{h-1})$ in Γ'''_{h-1} on the same vertical line of $r(T_h)$. Place Γ'_{h-1} with the rightmost vertical line intersecting it one unit to the left of the leftmost vertical line intersecting Γ'''_{h-1} and with the root $r(T_{h-1})$ in Γ'_{h-1} on l_h. Place Γ''_{h-1} with the leftmost vertical line intersecting it one unit to the right of the rightmost vertical line intersecting Γ'''_{h-1} and with the root $r(T_{h-1})$ in Γ''_{h-1} on l_h (see Fig. 4.a). In *Construction* 2 draw $r(T_h)$ on any horizontal grid line l_h. Place Γ'_{h-1} with the rightmost vertical line intersecting it one unit to the left of $r(T_h)$ and with the root $r(T_{h-1})$ in Γ'_{h-1} on l_h. Place Γ''_{h-1} with the leftmost vertical line intersecting it one unit to the right of $r(T_h)$ and with the root $r(T_{h-1})$ in Γ''_{h-1} on l_h. Place Γ'''_{h-1} with the highest horizontal line intersecting it one unit below the lowest horizontal line intersecting Γ''_{h-1}, and with the root $r(T_{h-1})$ in Γ'''_{h-1} on the same vertical line of $r(T_h)$ (see Fig. 4.b). We have the following:

Theorem 6. *An n-node complete ternary tree T_h admits an $O(n^{1/\log_4 3}) = O(n^{1.262})$ area SO-drawing.*

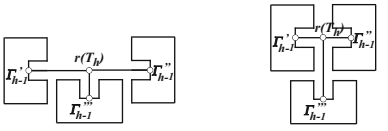


Fig. 4. Constructions 1 (a) and 2 (b) for SO-drawings of complete ternary trees

Proof: Construct a drawing Γ_h of T_h by inductively using Construction 1. Denoting by W_h and by H_h the width and the height of Γ_h, respectively, by construction we have $W_h = W_{h-1} + 2H_{h-1}$ and $H_h = \max\{W_{h-1}, H_{h-1} + (W_{h-1} + 1)/2\}$. Assume by inductive hypothesis that $W_{h-1} = 2^{h-1} - 1$ and that $H_{h-1} = 2^{h-2}$. Notice that this holds in the base case, where $W_1 = H_1 = 1$. Observe also that by inductive hypothesis $H_{h-1} + (W_{h-1} + 1)/2 = 2^{h-2} + (2^{h-1} - 1 + 1)/2 = 2^{h-1} > W_{h-1} = 2^{h-1} - 1$. Hence, $H_h = 2^{h-1}$ and $W_h = 2^{h-1} - 1 + 2 \cdot 2^{h-2} = 2^h - 1$, that proves the inductive hypothesis. The area of Γ_h is equal to $(2^h - 1) \cdot 2^{h-1} < 4^h = 4^{O(\log_3 n)} = O(n^{1/\log_4 3})$. Inductively applying Construction 2 instead of Construction 1 yields to a drawing with asymptotically the same area. □

Next, we show that n-node complete ternary trees have $\Omega(n^{0.438})$ minimum side in any SO-drawing. This result sharply contrasts with the analogous for binary trees. Namely, *any* binary tree admits an SO-drawing in which one side is $O(\log n)$. Let Γ_h be any SO-drawing of T_h. One of the children of $r(T_h)$, say v_1, is such that no other child of $r(T_h)$ is drawn on the line l through $r(T_h)$ and v_1. Moreover, for $i = 1, 2, \ldots, h-2$, node v_i has exactly one child v_{i+1} drawn on l. Hence, in any SO-drawing of T_h, there is a spine of h nodes drawn all on the same horizontal or vertical line l, such that no other child of $r(T_h)$ is on l. We call *leg* of Γ_h such a spine. Analogously, in any SO-drawing of T_h there is a double-spine of $2h - 1$ nodes that are drawn all on the same horizontal or vertical line. We call *arm* of Γ_h such a double-spine. We have:

Lemma 2. *The minimum side of any SO-drawing of an n-node complete ternary tree is $\Omega(n^{0.438})$.*

Proof: Let Γ_h be an SO-drawing of a complete ternary tree T_h in which the length of the leg is minimum. Let $l(\Gamma_h)$ be the length of the leg in Γ_h. We claim that $l(\Gamma_h) \geq l(\Gamma_{h-1}) + l(\Gamma_{h-2})$. Consider the arms of the subtrees of $r(T_h)$. Either two of such arms are vertical and one horizontal or vice versa. Assume,

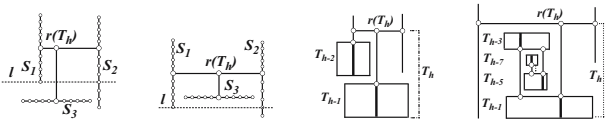


Fig. 5. Illustrations for Lemma 2. Thick lines drawn inside subtrees represent their legs.

possibly rotating Γ_h of $\pi/2$, that two of such arms, say S_1 and S_2, are vertical and one, say S_3, horizontal. Consider the possible non-crossing placements of S_1, S_2, and S_3, and consider the lowest horizontal line l intersecting both S_1 and S_2. Two are the cases; either S_3 is below l (Fig. 5.a), or not (Fig. 5.b). In the first case we trivially have $l(\Gamma_h) > l(\Gamma_{h-1}) + l(\Gamma_{h-2})$ (see Fig. 5.c) and the claim follows. In the second case we have $l(\Gamma_h) > l(\Gamma_{h-1}) + l(\Gamma_{h-3}) + l(\Gamma_{h-5}) + \ldots + l(\Gamma_{h/2-\lfloor h/2 \rfloor+3}) + l(\Gamma_{h/2-\lfloor h/2 \rfloor+1})$ (see Fig. 5.d). However, recurrence (1) $f(x) = f(x-1)+f(x-3)+f(x-5)+\ldots+f(x/2-\lfloor x/2 \rfloor+3)+f(x/2-\lfloor x/2 \rfloor+1)$ asymptotically provides for $f(x)$ the same value provided by $f(x) = f(x-1) + f(x-2)$. Namely, from (1) we get $f(x-2) = f(x-3) + f(x-5) + \ldots + f((x-2)/2 - \lfloor (x-2)/2 \rfloor + 3) + f((x-2)/2 - \lfloor (x-2)/2 \rfloor + 1)$, and since $(x-2)/2 - \lfloor (x-2)/2 \rfloor = x/2 - \lfloor x/2 \rfloor$ we get $f(x-3) + f(x-5) + \ldots + f(x/2-\lfloor x/2 \rfloor+3)+f(x/2-\lfloor x/2 \rfloor+1) = f(x-2)$, that substituted in (1) gives $f(x) = f(x-1)+f(x-2)$. Hence, $l(\Gamma_h) \geq l(\Gamma_{h-1})+l(\Gamma_{h-2})$, implying that $l(\Gamma_h)$ grows at least as the terms of the *Fibonacci series*, for which it is well know that the ratio of two consecutive terms l_{k+1} and l_k tends to the *golden ratio* ϕ. Hence $l(\Gamma_h) = \Omega(\phi^h) = \Omega(\phi^{\log_3 n}) = \Omega(n^{1/\log_\phi 3}) = \Omega(n^{0.438})$. The statement follows by observing that the minimum length of the arm of Γ_h grows asymptotically at least as the leg of Γ_h and that each side of Γ_h is at least long as the leg or as the arm of Γ_h. □

In the following we prove that, for complete ternary trees, the lower bound of Lemma 2 is tight. Again, we introduce two constructions, called Constructions $\hat{1}$ and $\hat{2}$, defined as follows: Construction $\hat{1}$ has the same geometric inductive step of Construction 1, but the side drawings are recursively constructed with Construction $\hat{2}$ and the base drawing is recursively constructed with Construction $\hat{1}$; Construction $\hat{2}$ has the same geometric inductive step of Construction 2, but the side drawings are recursively constructed with Construction $\hat{1}$ and the base drawing is recursively constructed with Construction $\hat{2}$.

Lemma 3. *The drawings built by Construction* $\hat{1}$ *have* $O(n^{0.438})$ *height.*

Proof: Denote by H_h^1 (by W_h^2) the height (the width) of the drawing of a complete ternary tree with height h built with Construction $\hat{1}$ (with Construction $\hat{2}$). By simple geometric considerations, we have (1) $H_h^1 = \max\{W_{h-1}^2, H_{h-1}^1 + (W_{h-1}^2 + 1)/2\}$ and (2) $W_h^2 = \max\{W_{h-1}^2, 2H_{h-1}^1 + 1\}$. Suppose by induction that $H_{h-1}^1 + (W_{h-1}^2+1)/2 \geq W_{h-1}^2$ and that $2H_{h-1}^1+1 \geq W_{h-1}^2$. Such inductive hypotheses are verified in the base case, where $H_1^1 = 1$ and $W_1^2 = 1$. Due to the inductive hypotheses, (1) and (2) turn in (1') $H_h^1 = H_{h-1}^1 + (W_{h-1}^2 + 1)/2$, and (2') $W_h^2 = 2H_{h-1}^1+1$, respectively. We have $H_h^1+(W_h^2+1)/2 = H_{h-1}^1+(W_{h-1}^2+1)/2+(2H_{h-1}^1+1+1)/2 = 2H_{h-1}^1 + (W_{h-1}^2)/2 + 3/2 > 2H_{h-1}^1 + 1 = W_h^2$, and that $2H_h^1+1 = 2H_{h-1}^1 + W_{h-1}^2 + 1 + 1 > 2H_{h-1}^1 + 1 = W_h^2$. Hence, the inductive hypothesis is verified and (1') and (2') hold. Substituting (2') into (1'), we get $H_h^1 = H_{h-1}^1+((2H_{h-2}^1+1)+1)/2 = H_{h-1}^1+H_{h-2}^1+1$. As in the proof of Lemma 2, H_h^1 grows as the terms of the Fibonacci series, yielding $H_h^1 = O(n^{0.438})$. □

6 Conclusions and Open Problems

In this paper we have shown some upper bounds (Theorems 2, 4, 5, and 6) and lower bounds (Theorems 1 and 3) concerning the area requirement of straight-line orthogonal drawings of binary and ternary trees. As can be noticed from Table 1 some of these bounds are asymptotically tight, whereas for others there is still space for improvements. In particular, for order-preserving SO-drawings of binary trees and for SO-drawings of ternary trees there are wide gaps between the area upper bounds we provided and the actual lower bounds. For complete ternary trees we conjecture that an algorithm combining Constructions 1 and 2 could improve the upper bound we provided here. However, the most fascinating problem in this area still remains, in our opinion, the one of determining whether binary trees admit straight-line orthogonal drawings in linear area.

References

1. Chan, T.M., Goodrich, M.T., Rao Kosaraju, S., Tamassia, R.: Optimizing area and aspect ratio in straight-line orthogonal tree drawings. Comput. Geom. 23(2), 153–162 (2002)
2. Chan, T.M.: A near-linear area bound for drawing binary trees. Algorithmica 34(1), 1–13 (2002)
3. Crescenzi, P., Di Battista, G., Piperno, A.: A note on optimal area algorithms for upward drawings of binary trees. Comput. Geom. 2, 187–200 (1992)
4. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing. Prentice Hall, Upper Saddle River, NJ (1999)
5. Dolev, D., Trickey, H.W.: On linear area embedding of planar graphs. Technical report, Stanford, USA (1981)
6. Garg, A., Goodrich, M.T., Tamassia, R.: Planar upward tree drawings with optimal area. Int. J. Comput. Geometry Appl. 6(3), 333–356 (1996)
7. Garg, A., Rusu, A.: Area-efficient order-preserving planar straight-line drawings of ordered trees. Int. J. Comput. Geometry Appl. 13(6), 487–505 (2003)
8. Garg, A., Rusu, A.: Straight-line drawings of general trees with linear area and arbitrary aspect ratio. In: ICCSA, vol. (3), pp. 876–885 (2003)
9. Kim, S.K.: Simple algorithms for orthogonal upward drawings of binary and ternary trees sung. In: CCCG, pp. 115–120 (1995)
10. Shiloach, Y.: Arrangements of Planar Graphs on the Planar Lattice. PhD thesis, Weizmann Institute for Science (1976)
11. Shin, C.S., Kim, S.K., Chwa, K.Y.: Area-efficient algorithms for straight-line tree drawings. Comput. Geom. 15(4), 175–202 (2000)
12. Valiant, L.G.: Universality considerations in VLSI circuits. IEEE Trans. Comp. 30(2), 135–140 (1981)

Polynomial Area Bounds for MST Embeddings of Trees

Michael Kaufmann

Wilhelm-Schickard-Institut für Informatik – Universität Tübingen, Germany

Abstract. In their seminal paper on geometric minimum spanning trees, Monma and Suri [6] gave a method to embed any tree of maximal degree 5 as a minimum spanning tree in the Euclidean plane. They derived area bounds of $O(2^{k^2} \times 2^{k^2})$ for trees of height k and conjectured that an improvement below $c^n \times c^n$ is not possible for some constant $c > 0$. We partially disprove this conjecture by giving polynomial area bounds for arbitrary trees of maximal degree 3 and 4.

1 Introduction

As a classical concept, a minimum spanning tree (MST) of n points in the plane is defined as being the smallest tree that spans all n points. As the distance between two non-adjacent vertices must be at least as large as any distance between adjacent vertices on the path between the two vertices, the MST reflects certain proximity relations between the points in the plane, it plays important roles in various fields. It recently gained attention in the field of sensor networks where the positioning and energy consumption of sensor nodes is an important issue. For example, in [4] the authors use the FIRST ORDER RADIO MODEL as energy model, to develop heuristics for MSTs in the plane with hop and degree constraints.

We reconsider the embedding question: Given a certain tree topology as an input, find geometric positions in the Euclidean plane for the vertices of the tree, such that the geometric MST for those points exactly corresponds to the given tree topology. We call this problem the MST embedding problem. Monma and Suri [6] gave a method to find such positions for trees of maximal degree 5, and they showed that their algorithm uses an area of $O(2^{k^2} \times 2^{k^2})$ for trees of height k and made a simple observation, that trees of degree 7 cannot be drawn as Euclidean MST. Eades and Whitesides [3] filled the gap by their result that the decision whether an MST embedding is possible for a given tree of degree 6 is NP-hard. In both papers, the open problem is given whether or not the area bounds can be improved to polynomial. Note that the algorithm of Monma and Suri does not even give a polynomial bound for complete trees of maximal degree 5, their area bound there reads $n^{O(logn)}$, which is clearly superpolynomial.

Extensions of this work to 3D have been performed by Di Battista and Liotta [2] as well as by King [5]. In the former paper, the authors proved that trees with maximal degree 9 can be embedded as MSTs in three-dimensional space; this result has been recently extended to trees of maximal degree 10.

In this paper, we concentrate on the area requirements for Euclidean MST embeddings. First, we give a simple technique for complete binary trees, which will be extended to the case of arbitrary binary trees. In the second part of the paper, we consider

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 88–100, 2007.

ternary trees, which are trees of maximum degree 4. Again, we develop a basic scheme for the complete ternary tree, and generalize it afterwards to arbitrary ternary trees.

Note that we do not strive for the best polynomial bounds but try to keep the techniques as simple as possible. Nevertheless, we achieve the first polynomial bounds drastically improving from the previous exponential ones and solving long-standing open problems.

2 Preliminaries

A minimum spanning tree MST of a set of n points in the plane is defined to be a tree with lowest total cost where the cost of each edge (u, v) is defined by the Euclidean distance between u and v. Given a tree T, the MST embedding problem now asks for a mapping of the vertices in T to points in the plane such that the MST of this point set is exactly the input tree T. This problem can be solved for trees of maximum degree 5, it cannot be solved for trees of maximum degree larger than 6, and it is NP-hard to decide whether there exists a MST embedding for a given tree of maximum degree 6. Furthermore it is known that Euclidean MSTs have the properties that all edges on the path in the tree between non-adjacent vertices v and w are not longer than the distance between v and w in the plane. We call this the MST condition. In our constructions, we have to prove that the MST condition has been observed. It is clear that the edges are non-crossing when drawn straight-line. We define the area consumption of an embedding to be the area of the rectangle enclosing the embedding.

3 The Complete Binary Case

Let T be a complete binary tree of size n and let $n = 2^k - 1$ for some integer k. It consists of a root vertex r and two subtrees T_1 and T_2 rooted at the direct children r_1 and r_2 of the root. We recursively embed the subtrees of size $n/2$ into two equal-sized cones with angle of 90 degrees where the roots are located at the top of the cones of the corresponding subtree. We place the two cones of the subtrees next to each other separated by a distance of d where d will be chosen later. Let L denote the side length of the cones of the subtrees. Then we place the root r on top of the two cones connecting the two roots by edges of length $c \cdot L$ such that those edges again form an angle of 90 degrees. Clearly, the whole construction again gives a cone of side length $(c + 1) \cdot L$ where the root r is placed at the top, cf. Figure 1.

By the construction, the distance d between the inner corners of the cones is $d = \sqrt{2}(c - 1)L$. To ensure the conditions for the MST, we choose c such that d is larger than the length cL of the two longest edges which is incident to the root. All other distances obey the MST conditions by induction. This is true if $c \geq \sqrt{2}/(\sqrt{2} - 1)$.

Hence we obtain a recursion for the sidelength $S(n) = (c+1)S(n/2) = (\frac{2\sqrt{2}-1}{\sqrt{2}-1})^{\log n}$ which is $n^{\log \frac{2\sqrt{2}-1}{\sqrt{2}-1}} \leq n^{\log 4.415} \leq n^{2.2}$. From this, we obtain an area bound of $O(n^{4.4})$.

Theorem 1. *We can embed a complete binary tree of n vertices in an area of size $O(n^{4.4})$ such that the embedding obeys the MST conditions.*

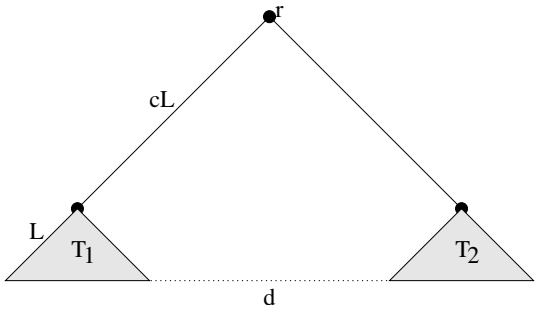


Fig. 1. The recursive construction for the complete binary case. Note that the figure should be seen as a sketch, the actual values in the figure violate the MST condition.

4 The Arbitrary Binary Case

This case is more involved. Before, our analysis relied on the fact that the depth of the tree and hence the recursion had only depth log n, which results in a polynomial bound for the area. The basic schema remains the same. We recursively embed the subtrees into cones with uniform angles of 90 degrees and try to combine two subtrees with two sibling vertices as roots by placing the common ancestor such that they form another larger cone observing the necessary distances. The two subtrees and hence their cones might differ more or less in size. If their cones differ by at most a constant factor each time, we can use an analogous construction as before to achieve a depth still logarithmic in n. If not, we have to introduce an alternative technique.

More exactly, let $P = v_1, v_2, \ldots v_k$ be the path from root v_1 to leaf v_k where v_{i+1} is defined as the root of the larger of the two subtrees of v_i for each $1 \leq i < k$. For each such v_i on this path, we consider the two subtrees T_{il} and T_{ir}, and we assume that T_{il} is at least as big as T_{ir}. If $|T_{il}|/|T_{ir}| \leq 3$ we call this a balanced step, and the step is called unbalanced otherwise. Clearly, there are at most $O(\log n)$ balanced steps. Between any two balanced steps there is a sequence $v_f, ..., v_g$ of unbalanced steps, of course there are at most $O(\log n)$ such sequences.

Realization of a balanced step at the vertex v_i with subtrees T_{il} and T_{ir}. Let L_l and L_r be the lengths of the two cones where the subtrees T_{il} and T_{ir} are embedded and let $L_l \geq L_r$.

We scale the smaller cone up by a factor of L_l/L_r such that the two cones have the same size and the same analysis as in the complete binary case can be done. Since the number of nodes in the larger subtree T_{il} is at most $3/4$ times the number of nodes in the original tree rooted at v_i this case can occur only $\log_{4/3} n$ times.

Realization of a maximal sequence of unbalanced steps. Let the sequence be $v_f, ..., v_g$ with intermediate edges (v_i, v_{i+1}) for $i = f, \ldots, g-1$. This sequence with the edges is called *c*hain. The final node v_g is either a leaf or it is the root of two balanced subtrees. We assume that the subtree rooted at v_g has been recursively embedded into

cone C_g by a balanced step. At each intermediate vertex v_i, there will be a relatively small subtree, say T_{ir} embedded recursively into cone C_i of side length L_i.

If the sequence is very short ($g - f \leq 2$), say only one or two unbalanced steps, followed by a balanced one, we will just neglect it and perform one or two ordinary balanced steps instead as described above. If $g - f > 2$ is odd, we perform the layout of the topmost two trees of the chain by balanced steps, while if $g - f > 2$ is even, we perform only one balanced step and then add the remaining unbalanced steps of the chain in one step. This is done to make sure that the last subtree of the chain with unbalanced layout lies towards the left of the chain.

It is clear, that the number of balanced recursion steps increases only by a factor of at most 3, so it is still $O(\log n)$, more exactly the number is at most $3 \log_{4/3} n$. We will use this fact again by correcting the following construction:

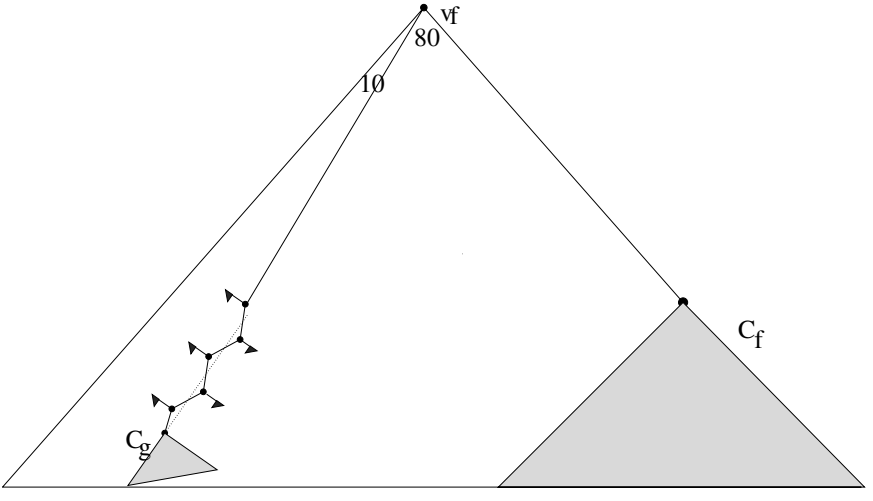


Fig. 2. The global recursive structure when adding a whole chain and the cone C_f

Let $e_i = (v_{i-1}, v_i)$ for $g \geq i \geq f$. We embed the chain of edges e_i in a zig-zag way with constant angles of 120 degrees, while the length of e_i will be chosen roughly proportional to the size of the adjacent cone. Consider an arbitrary vertex v_i. Opposite to the 120 degree angle, we have an angle of 240 degrees, which we subdivide into three disjoint cones of 90, 60, and 90 degrees. We place the cone C_i adjacent to v_i of size L_i together with its connecting segment s_i in the middle angle of 60 degrees such that the side of C_i which is closer to e_i is perpendicular to segment s_i. Furthermore we choose the length of s_i such that the small cone just fits into the angle of 60 degrees. Hence $s_i = L_i / \tan(30) \leq 1.8 L_i$.

Note that by construction the MST condition is observed for the distances between C_i and C_j for $|i - j|$ odd, namely between cones that lie on opposite sides of the chain.

We consider the horizontal distance between two adjacent cones C_{i-1} and C_{i+1}. The intermediate edges are s_{i-1}, e_i, e_{i+1} and s_{i+1}. Clearly we have distance $d_i = len(e_i) \cdot cos(30) + len(e_{i+1}) \cdot cos(30) - L_{i+1} \geq 0.8c(L_i + L_{i+1}) - L_{i+1}$, if we

choose $len(e_i) = cL_i$. Now to observe the MST condition, we have to choose c such that $d_i \geq len(e_i)$, and $d_i \geq len(e_{i+1})$ as well. Therefore, we require that $len(e_i) \geq max\{cL_i, 0.5len(e_{i+1})\}$ and $len(e_{i+1}) \geq 0.5len(e_i)$ as well for all i.

Then we can conclude that $d_i \geq 0.8 \cdot 1.5 \cdot cL_{i+1} - L_{i+1} \geq cL_{i+1}$ for $c \geq 5$. If $len(e_i) > 0.5len(e_{i+1})$ then c can be chosen even smaller. The case that $len(e_i) \geq len(e_{i+1})$ is simpler.

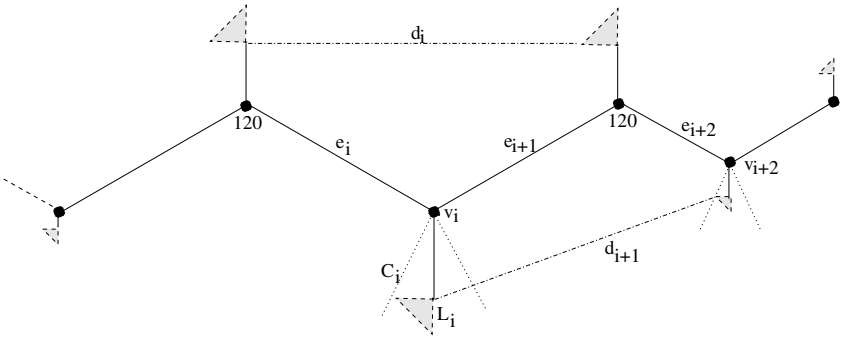


Fig. 3. Adding a chain of unbalanced vertices together with their adjacent subtrees

Note that segment s_{i+1} is smaller than $len(e_{i+1})$, and hence it is smaller than d_i, and s_{i-1} is smaller than $1.8L_{i-1}$. Furthermore $len(e_{i-1}) \geq 5L_{i-1}$, which means that $len(e_i) \geq 0.5len(e_{i-1}) \geq 2.5L_{i-1} \geq len(s_{i-1})$.

Lemma 1. *The distances between the vertices within the chain augmented by the adjacent subtrees observe the MST conditions.*

Ideally, the length of the chain should be bounded proportional to the sum of the lengths of the adjacent small cones. The requirement that the lengths of adjacent edges should differ by at most factor of 2 makes an estimation more difficult, but the following amortisation argument shows that this increases the total edge length of the chain at most by a factor of 3: e_i is chosen to be of length $max\{c \cdot L_i, c \cdot L_{i-1}/2, c \cdot L_{i+1}/2, ..., c \cdot L_{i+j}/2^j$. Hence each L_i at most contributes to the segment e_i, with half of its length to e_{i-1} and e_{i+1} and with $1/2^j$th of its length to e_{i-1} and e_{i+1} in general. Overall it is accounted at most three times. So, all L_i's together are accounted at most $3 \sum L_i$.

Lemma 2. *The total length of the chain with cones C_i of length L_i is at most* $15 \sum_{i=f+1}^{g} L_i$.

Next, we sketch the way how to complete the recursion and achieve again an appropriate cone of 90 degrees.

Completing the recursion step. The next Figure 4 gives an intuition how to rebuild the giant cone including the chain of small cones and the first cone C_f as well: We add the first cone C_f of the chain in a way similar to a balanced step. Since the construction of the chain require some width, basically, we keep the angle between the two edges

incident to the root of the two subtrees a little smaller than 90 degrees, namely 80 degrees. This leads to an increase of the lengths of the two topmost edges by a small constant factor to keep the necessary distance between the left and the right subtree. We keep the triangle between the root v_f and the two innermost points between the left and the right subtree to be equilateral. An equilateral triangle has angles of 60 degrees, hence we assign the remaining 20 degrees to the two subtrees, such that each of them lies within a cone of 10 degrees.

The edge e_f which connects v_f and v_{f+1} is routed using the same slope, as the left side of the cone C_g which is constructed recursively by a balanced step. The whole chain lies within a cone, called CC of 60 degrees whose middle axis extends the left side of C_g. The remaining 10 degrees of the final 90 degree cone are used to cover the small subtrees that stick out to the left of the line between the left side of the large cone C_g and the root v_f. We can easily verify that this small angle of 10 degrees is enough to include the whole substructure by observing the restrictions on the size of the small subtrees and the sufficient lengths of the edges from the chain to the root.

We have to embed all three parts, the cone C_f, the chain plus the cone C_g, and the small cones like C_{f+1} sticking out to the left, within cones of 10 degrees each. The final size of the giant cone depends on the actual sizes of the three components, at the end, we maximize over those three components. We give the analysis for the case that the chain dominates the final size, assuming that C_f is small and the other small cones are also small enough. The other cases are much simpler:

The length of the chain is at most $15 \sum_{i=f+1}^{g} L_i$, we call it LC. Largely overestimating, we require this value also for the cathetus opposite from the 10 degrees angle at v_f. Hence we get a bound of $LC/tan(10) \leq 5.7LC$ for the length of (v_f, v_{f+1}).

Note that this can be slightly larger, if the chain starts at the upper end of its upward directed cone CC, but not in the center. Furthermore we neglected the fact, that due to

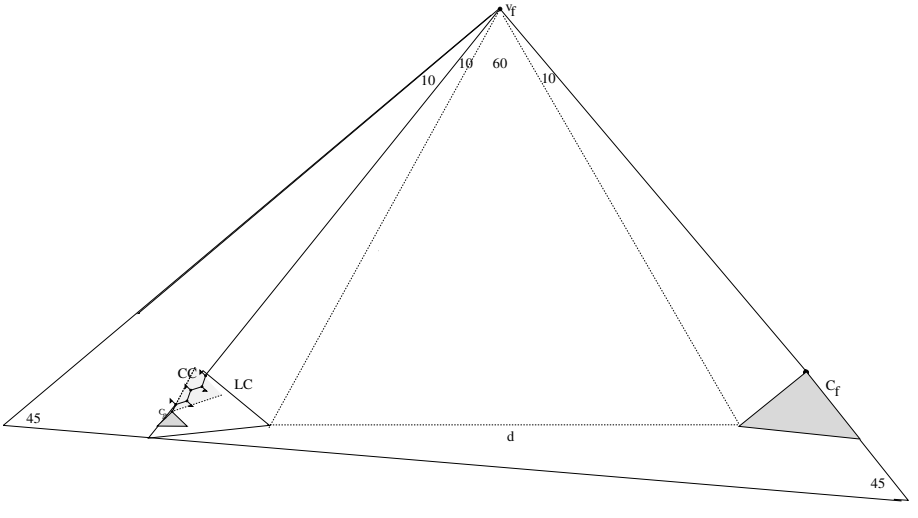


Fig. 4. The giant cone which is rotated such that the three parts C_f, the chain and the leftmost small cones are enclosed into cones of angles of 10 degrees, and the equilateral triangle keeps the distance

the separating equilateral triangle and the additional angle of 10 degrees on the left hand side, the sidelength of the final giant cone is larger than $LC + 5.7LC$ by an additional small factor. This factor can be bounded using more elementary trigonometry by 1.16.

Hence the new giant cone has a side length of $1.16 \cdot (5.7 + 1)15 \sum_{i=f}^{g} L_i \leq 117 \sum_{i=f}^{g} L_i$. Summarizing, we have

Lemma 3. *Adding a chain causes an increase of the side length, which is proportional to the sum of the lengths of its subtrees, with 117 as the multiplicative constant.*

We conclude the discussion by the recursion on the side length $S(n)$: We had at most $\log_{4/3} n$ balanced steps, and before each balanced step, there might be the insertion of a maximal chain consisting of unbalanced steps, hence only $\log_{4/3} n$ subsequent insertions of chains. In total, this means a maximal recursion depth of $2 \log_{4/3} n$, we get $S(n) \leq 117^{2 \log_{4/3} n} \leq O(n^{32.4})$.

Theorem 2. *We can embed an arbitrary binary tree of n vertices in an area of size polynomial in n such that the embedding obeys the MST conditions. The degree of the polynomial is at most 65.*

Note that this analysis above is a rough estimation. E.g., in balanced steps we have a constant factor, much smaller than 117, and in unbalanced steps and long chains, the larger subtrees are much smaller than $3n/4$. We leave this for further improvement.

5 The Complete Ternary Case

Instead of maintaining cones for the subtrees, we maintain diamonds. Consider proper subtree T_r rooted at vertex r. Vertex r is placed in the centre of its corresponding diamond of side length L. The diamond for T_r is partitioned into an upper and a lower

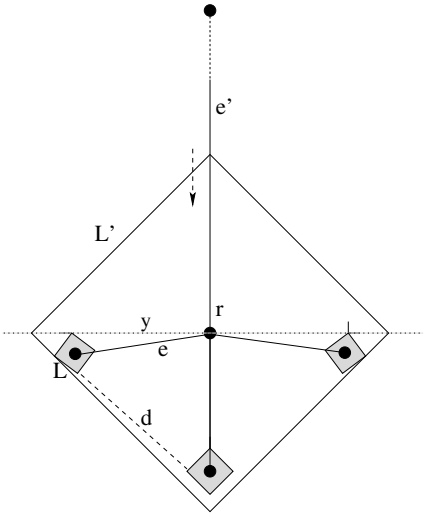


Fig. 5. The recursive scheme for complete ternary trees

half. The lower half contains the diamonds for the three subtrees, such that the edges from r to the diamonds maximize the angles between two adjacent edges.

This is a simple recursive scheme and we only have to choose the lengths of the edges appropriately such that the MST property still holds. Furthermore, we have to consider the ratio between the long edge e' from r to its parent and the shorter edges e to its children. It will then provide the area bound.

Now let the side length of the small diamonds be L. The length of edge e is $len(e)$ while the distance from vertex v to the upper / lower corners of the left and right diamonds is denoted by y. Note that y is larger than $len(e)$.

Clearly, the lower corner of the say left diamond and the left corner of the middle diamond should have a distance d of at least $len(e)$. Since $y \geq len(e)$ it is sufficient to ensure that $d = y$. The key observation is that under this condition the vertex r together with the lower corner of the left diamond and the left corner of the middle diamond forms a equal-sided triangle. All angles in this triangle have 60 degrees. Hence, the three small angles between the edges from r and the lines from r to the corners of the two diamonds complete the right angle, hence they have 10 degrees. So, we can use more trigonometry: $tan(10) = (L/\sqrt{2})/len(e)$. Hence $len(e) = (L/\sqrt{2})/tan(10) < 4.1L$.

Furthermore,the distance from the parent of r to the upper corners of the diamonds on the side is larger than the length of the edge $e' = (p, r)$. Next, we compute the side length L' of the large diamond which encloses r and the three small diamonds. More exactly, we will compute the length of the horizontal axis through vertex r. We know from the construction before that the segment which extends the edge e from the centre of the diamond to the left end has length $4.1L + L/\sqrt{2}$. The angle between this segment and the horizontal line has 10 degrees. Hence the cathetus opposite of this angle has length $sin(10) \cdot (4.1L + L/\sqrt{2}) < 0.84L$, we call it z. The length of the horizontal axis from r to the left corner is $z + z/tan(10) \leq 5.6L$, since it consists of two subsequent parts.

From this consideration, we know the ratio between subsequent diamonds, and hence we can conclude the needed side length $S(n) \leq 5.6S(n/3) = 5.6^{\log_3 n} \leq n^{3.94}$ and hence the area is polynomial in n.

Theorem 3. *We can embed a complete ternary tree of n vertices in an area of size $O(n^{7.88})$ such that the embedding obeys the MST conditions.*

6 The Case of Arbitrary Ternary Trees

As in the binary case, we have to consider the case that the subtrees of the vertices might have different sizes, leading to a much larger depth than in the complete ternary case. Note that the case of binary trees might have been omitted after all, since it is just a subcase of the ternary case. The techniques used are similar to those for the general binary case, but they are even more involved. Therefore we omit the calculation of concrete exponents of the polynomials.

Consider a vertex v root of a subtree T of size n' divided into three subtrees T_1, T_2 and T_3 of sizes n_1, n_2 and n_3 respectively, such that $n_1 \geq n_2 \geq n_3 \geq 0$. T_i is called 'large' if $n_i \geq n'/4$ and 'small' otherwise, for $i = 1, 2, 3$.

We consider 2 cases: If we have at least 2 large subtrees, we proceed similar to the complete ternary case. Observe that if we would be in this case each time, then the maximal size of a subtree would shrink in each iteration at least by a constant fraction $3/4$ or so. area if we only follow the scheme as in the complete ternary case. This corresponds to the balanced recursion step in the binary case. We take the size of the three diamonds identical by scaling up the sizes of the edge lengths in the smaller subtrees. Since the recursion depth remains $O(\log n)$, the area size will still be polynomial.

The second case is much more interesting: We assume to have 2 small subtrees. We walk down into the large subtree and check, if there are at least two large subtrees starting from there. If this is true for either this step or the next step, such that there are again at least 2 large subtrees, we can neglect the area loss of the 2 small subtrees by scaling them up as before and make a balanced recursion step. Hence the number of large recursion steps at most triples by this effect but it still remains logarithmic.

Constructing a chain of small diamond twins. If not, we construct a maximal chain of vertices $v_1, ..., v_k$ with small subtrees represented by small diamonds. The indexing of the chain starts at the lowest vertex v_1, which actually has one large subtree and two small subtrees. In general, v_i has two small and one large subtree with root v_{i+1}, which again has one large subtree and two small subtrees. The vertex v_k is either the root, or its parent has two large subtrees, such that we can apply a balanced recursion step.

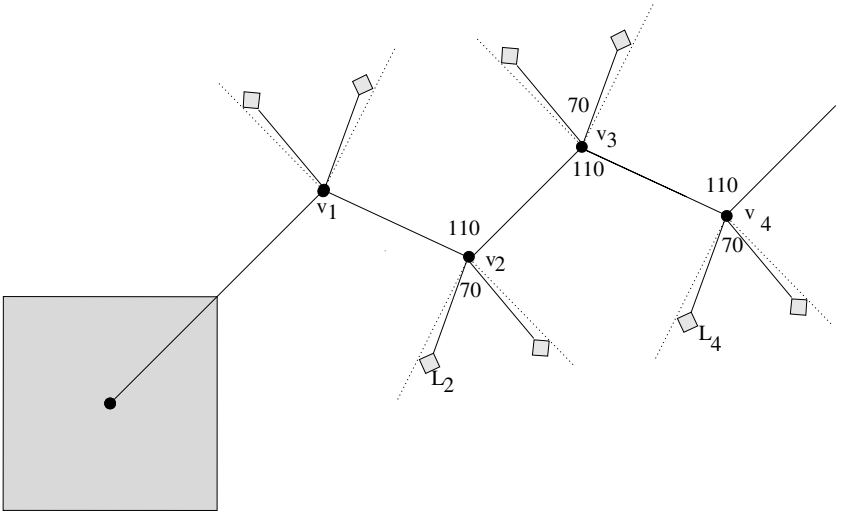


Fig. 6. Embedding a chain of small diamonds

To embed the chain with the twin diamonds appended at each vertex we take an appproach similar to the general binary case: We embed the chain in a zig-zag way with angles of 110 degrees and the lengths are suitably chosen roughly proportional to the size of the adjacent diamonds. We will determine the exact edge lengths lateron.

Consider vertex v_i. Opposite of the 110 degree angle, we install a cone of 70 degrees where we place the two small diamonds adjacent to v_i. First of all, we scale the smaller

diamond such that both diamonds have the same size. The diamonds are connected with edges g_i and g_i' and they are placed tangent to the two sides of the cone. The lengths of the edges have to be chosen such that the distance d_i between the two diamonds is at least as large as the lengths of g_i and g_i'. We ensure this by enlarging the lengths such that together with v_i the two corners of the diamonds opposite to each other form an equilateral triangle. Hence the two diamonds have to be placed within the cones of degree 5 on the sides of the equilateral triangle. Solving $tan(2.5) = L_i/(\sqrt{2}len(g_i))$ where L_i is the width of the largest of the two diamonds, and $len(g_i)$ gives the lengths of the two connecting edges. Hence $len(g_i) = L_i/(\sqrt{2}tan(2.5)) \approx 16.2 \cdot L_i$.

Note that by the choice of the cone for the two diamonds, they are farer away from the position of v_{i-1} resp. v_{i+1} independent of the lengths of e_{i-1} and e_i in the chain.

The case of uniform sizes. To actually choose the length of the edges e_i, assume for a moment, that we have uniform side lengths L_i of the diamonds. Then we can choose the lengths of the g_i's and those of the e_i's as being uniform. To keep the distance between the diamonds at v_i and v_{i+2} large enough, we consider the triangle with cathetus e_{i+1} and angle of 20 degrees at vertex v_{i+1}, and call the other cathetus D. If $D \geq len(g_{i+2}) + \sqrt{2}L_{i+2} \geq 17.5L_{i+2}$, the distance between the two diamonds is large enough. So, $D = tan(20) \cdot len(e_{i+1}) \geq 0.37len(e_{i+1})$, and furthermore $len(e_{i+1}) = 17.5 \cdot L_{i+2}/0.37 \geq 47.3L_{i+2}$.

The general case. Unfortunately, this is not sufficient since we want to have the total length of the chain to be proportional to the sum of the L_i's. So we have to observe the different sizes of the diamonds. We introduce another restrictive piece of construction.

The following calculation shows how much we can reduce the edge length within one step. Consider vertices v_i, v_{i+1} and v_{i+2} with connecting edges e_i and e_{i+1}. We choose $len(e_i) = 47.3L_i$, and want to reduce the length $len(e_{i+1})$ of the next edge to $3/4 \cdot len(e_i)$. Let D be the cathetus opposite of the 20 degrees angle at v_{i+1}, where the small diamonds at v_{i+2} should be placed. Then $len(D) = len(e_{i+1} \cdot tan(20) \geq 0.36len(e_{i+1}) = 0.36 \cdot 3/4 \cdot len(e_i) \geq len(e_i)/4$. By this 20 degree angle, we introduced a line r_{i+1} parallel to the right border of the cone at v_i at the distance of $len(e_{i+1})$, keeping the diamonds incident to v_i and v_{i+2} respectively sufficiently apart from each other. Hence it is safe to embed at vertex v_{i+2} diamonds of size at most $L_{i+2} = L_i/4$, and the connecting edges having length $len(g_{i+2}) = 17.5L_{i+2}$ while reducing the length of e_{i+1} by factor $3/4$.

In general, we introduce lines r_i and p_i at each vertex v_i parallel to the cone boundaries at the neighboring vertices v_{i-1} and v_{i+1}, extending the two lines of the cone at v_i to the other side of the chain. We require that the small diamonds adjacent to v_i and more specifically the length of g_i are restricted by the lines r_{i-1} and p_{i+1}, which is critical especially if the lengths of the edges e_i change. The above calculation showed to which extend edge e_{i+1} can be shortened if $L_{i+2} \leq L_i/4$. But note that the line p_{i+2} comes then closer to the edge e_{i+1} such that it might intersect the diamonds at v_{i+1}. This problem can be resolved by requiring that the edge e_{i+1} can only be shorted if the diamonds at both v_{i+1} and v_{i+2} as well are smaller by at least a factor of 4 compared to the diamonds at v_i.

We conclude the analysis and and summarize the arguments for the correctness.

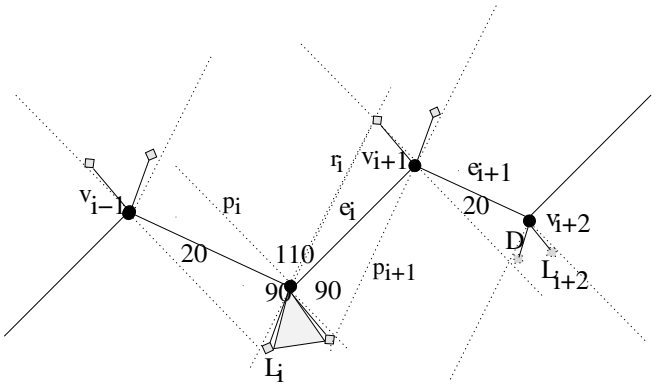


Fig. 7. A detailed look to the chain of small diamonds when the size of the diamonds and the length of the edges shrink. Notice the lines p_i and r_i parallel to the sides of the cones.

Lemma 4. *Proceeding from v_i to v_{i+2}, we can shrink the length of the second edge e_{i+1} by a factor of $4/3$ compared to e_i, if the widths of the diamonds at v_{i+1} and v_{i+2} shrink by at least a factor of 4.*

1. The Scaling up procedures do not violate any MST condition.
2. Cones at adjac. vertices do not conflict.They are strictly separated by construction.
3. Diamonds within the same cone do not interfere since they are separated by an equilateral triangle.
4. Diamonds adjacent to v_i do not interfere with those adjacent to v_{i+2}; here we can argue using the separating lines parallel to the sides of the cone.

The final length of the chain. Note that we did not fix the length of e_i, its lower bound was $47.3L_i$, according to the lemma, the length depends on the size of the diamonds in the close or further vicinity. So we do the same with v_{i-1} and v_{i+1} and then maximize.

More precisely, for each v_i, we receive a lower bound of $47.3L_i$ for the two incident edges e_{i-1} and e_i respectively. Furthermore, we have the dependency between neighboring edges e_i and e_{i+1}, namely that the ratio of the lengths can be chosen to be either $3/4, 1$ or $4/3$. Remark that we could also choose some intermediate value, but for the clarity of exposition we take only those discrete values.

Finally, we argue that the sum over the lengths of the edges e_i on the chain is polynomial in the number of vertices involved (more exactly in the sum over the lengths of all small diamonds). We have to cover the diamonds involved with a sequence of larger diamonds of a certain size such that adjacent groups might either be identical or differ by a ratio of $1/4$ or 4. In these cases, the edge lengths stay either identical, shrink or enlarge by a factor of $4/3$ respectively.

By an amortization argument which is not much more complicated than the one above, we can show that the sum of the edge lengths is still $O(L)$, where $L = \sum_{i=1}^{k} L_i$, hence linear in the sum of the side lengths of the diamonds involved.

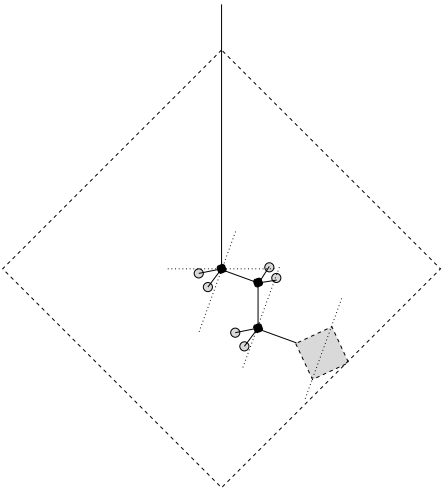


Fig. 8. Placing the chain of small diamonds plus the large diamond within the giant diamond

The recursive construction and its complexity. Finally, we use the construction from Figure 8 to combine the last large diamond of size n_0 with the chain of diamonds of sizes $n_1, \ldots, n_k$ and get the following recursion for the resulting side length:

$S(n) = c_0 S(n_0) + c_1 \cdot \sum_{i=1}^{k} S(n_i)$ for appropriate constants c_0 and c_1. Since we have at most $O(\log n)$ levels of recursion while $n = \sum_{i=0}^{k} n_i$ and $n_i <= 3/4n$ for any $i = 0, \ldots, k$ we conclude in the same way as in the general binary case:

Theorem 4. *Let T be any tree with n vertices of maximal degree 4. We can find an MST-embedding for T in a grid of size $O(n^d)$ for a suitable large constant d.*

7 Discussion and Conclusion

We have shown some techniques for constructing and analysing area-efficient MST-embeddings of trees. Our goal was to prove polynomial area bounds improving the previously known exponential bounds. A more detailed analysis of course could provide exact constants for the exponents. We started with binary trees and some basic analysis. The complexity of the analysis increased when proceeding to ternary trees where the vertices have maximum degree 4. It is known that trees with maximum degree 5 also have MST-embeddings. For this case we expect an exponential area lower bound.

Acknowledgement. This work has been initiated at the Workshop of Graph Drawing and Computational Geometry in Bertinoro, Italy. Thanks to the organizers and participants for the inspiring atmosphere. The initial part about complete binary trees was done together with Roberto Tamassia. Special thanks also go to Fabrizio Frati, Markus Geyer and Barbara Pampel for many critical remarks und suggestions.

References

1. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, Englewood Cliffs (1999)
2. Di Battista, G., Liotta, G.: Computing proximity drawings of trees in the 3-dimensioanl space. In: Sack, J.-R., Akl, S.G., Dehne, F., Santoro, N. (eds.) WADS 1995. LNCS, vol. 955, pp. 239–250. Springer, Heidelberg (1995)
3. Eades, P., Whitesides, S.: The Realization Problem for Euclidean Minimum Spanning Trees in NP-Hard. Algorithmica 16(1), 60–82, 1–15 (1996)
4. Cheng, H., Liu, Q., Jia, X.: Heuristic algorithms for real-time data aggregation in wireless sensor networks. In: Proceedings of the 2006 International Conference on Communications and Mobile Computing (2006)
5. King, J.: Realization of Degree 10 Minimum Spanning Trees in 3-Space. In: CCCG 2006. Proceedings of the 18th Canadian Conference on Computational Geometry, pp. 39–42 (2006)
6. Clyde, L., Monma, C.L., Suri, S.: Transitions in Geometric Minimum Spanning Trees. Discrete & Computational Geometry 8, 265–293 (1992)

Moving Vertices to Make Drawings Plane*

Xavier Goaoc[1], Jan Kratochvíl[2], Yoshio Okamoto[3,**], Chan-Su Shin[4,***], and Alexander Wolff[5]

[1] LORIA – INRIA Lorraine, Nancy, France
goaoc@loria.fr
[2] Dept. Applied Math. and Inst. Theoret. Comp. Science, Charles Univ., Czech Rep.
honza@kam.mff.cuni.cz
[3] Dept. Information and Computer Sciences, Toyohashi Univ. of Technology, Japan
okamotoy@ics.tut.ac.jp
[4] School of Digital Inform. Eng., Hankuk Univ. of Foreign Studies, Yongin, Korea
cssin@hufs.ac.kr
[5] Faculteit Wiskunde en Informatica, TU Eindhoven, The Netherlands
http://www.win.tue.nl/~awolff

Abstract. In John Tantalo's on-line game *Planarity* the player is given a non-plane straight-line drawing of a planar graph. The aim is to make the drawing plane as quickly as possible by moving vertices. In this paper we investigate the related problem MinMovedVertices which asks for the minimum number of vertex moves. First, we show that MinMovedVertices is NP-hard and hard to approximate. Second, we establish a connection to the graph-drawing problem 1BendPointSetEmbeddability, which yields similar results for that problem. Third, we give bounds for the behavior of MinMovedVertices on trees and general planar graphs.

1 Introduction

It is somewhat surprising that many people still draw graphs by hand, usually not on a piece of paper but on a computer display. Modern technology gives us the means to edit a drawing by dragging vertices. Even when we use an automatic graph-drawing tool, we often do some manual polishing to obtain nicer drawings.

In this paper, we consider the problem of editing a given drawing to obtain another drawing that fulfills a certain criterion. We restrict ourselves to straight-line drawings of planar graphs. Our edit operation is "moving vertices." When

* This work was started on the 9th "Korean" Workshop on Computational Geometry and Geometric Networks organized by A. Wolff and X. Goaoc, July 30–August 4, 2006 in Schloß Dagstuhl, Germany. Further contributions were made at the 2nd Workshop on Graph Drawing and Computational Geometry organized by W. Didimo and G. Liotta, March 11–16, 2007 in Bertinoro, Italy.

** Partially supported by Grant-in-Aid for Scientific Research from Ministry of Education, Science and Culture, Japan, and Japan Society for the Promotion of Science.

*** Supported by Research Grant 2007 of the Hankuk University of Foreign Studies.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 101–112, 2007.

we move a vertex v to a new position, the incident edges are redrawn so that v is again connected to its adjacent vertices by straight-line segments. Our criterion is planarity. According to the famous theorem of Wagner [15], Fáry [2], and Stein [12] every planar graph has a plane straight-line drawing. We want to obtain such an embedding from a given (usually non-plane) straight-line drawing. Our goal is to minimize the number of vertices to move. This is a natural question because the less vertices we move the better the *mental map* [8] of an observer is preserved when making a given drawing plane, e.g., in a step-by-step fashion. Note that for a given straight-line drawing the minimum number of moves can also be seen as the edit distance from the closest plane drawing.

At the 5th Czech-Slovak Symposium on Combinatorics in Prague in 1998, Mamoru Watanabe asked the following question, which concerns a special case of our problem: Is it true that every polygon P with n vertices can be untangled, i.e., turned into a non-crossing polygon, by moving at most εn of its vertices for some absolute constant $\varepsilon < 1$? Pach and Tardos [9] have answered this question in the negative by showing that there must be polygons where at most $O((n \log n)^{2/3})$ of the vertices can be kept fixed. They also gave a simple algorithm (which can be implemented in $O(n \log n)$ time) that always keeps more than $\sqrt{n}$ vertices. In a longer version of this paper [3] we show that their algorithm is not optimal. Pach and Tardos [9] in turn asked the following question: can any straight-line drawing of any planar graph with n vertices be made plane by vertex moves while keeping $\Omega(n^\gamma)$ vertices fixed for some absolute constant $\gamma > 0$? We still do not know the answer to this question, but we report some progress.

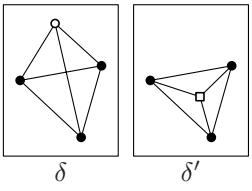


Fig. 1. Two drawings of K_4: δ is not plane, δ' is plane; $d(\delta, \delta') = 1$

There is a popular on-line game that is related to the problem of Pach and Tardos. In John Tantalo's game *Planarity* [13] the player is given a non-plane straight-line drawing of a planar graph. The player can move vertices, which always keep straight-line connections to their neighbors. The aim is to make the drawing plane as quickly as possible. We study the game from three view points: (a) algorithms, (b) mathematics (upper-bound constructions), and (c) complexity. Our complexity results (detailed below) made us understand why it is so hard to play the game well.

Formalization. In this paper, a *drawing* of a graph $G = (V, E)$ will always mean a straight-line embedding of G in the plane $\mathbb{R}^2$. Since such an embedding is completely defined by the position of the vertices, it corresponds to an injective map $\delta\colon V \to \mathbb{R}^2$. A drawing is *plane* if no two edges cross, i.e., they are only allowed to intersect in a common endpoint. A graph is planar if it admits a plane drawing; trivially not every drawing of a planar graph is plane.

The *vertex-moving distance* d between two drawings δ and δ' of a graph G is defined as the number of vertices of G whose images under δ and δ' differ:

$$d(\delta, \delta') = \big|\{v \in V \mid \delta(v) \neq \delta'(v)\}\big|.$$

This distance can easily be computed. Given our edit operation, d represents the edit distance for straight-line drawings of graphs. Figure 1 shows an example. Using d we can express the central question of this paper as follows.

> How close is a given drawing of a planar graph to being plane with respect to the vertex-moving distance d?

For a drawing δ of a planar graph G, denote by $\mathrm{MMV}(G, \delta)$ the minimum number of vertices that need to be moved in order to make δ plane. MMV measures distance from planarity: $\mathrm{MMV}(G, \delta) = \min_{\delta'} d(\delta, \delta')$, where δ' ranges over all plane drawings of G. This gives rise to the following computational problem.

> MinMovedVertices(G, δ): Given a drawing δ of a planar graph G, find a plane drawing δ' of G with $d(\delta, \delta') = \mathrm{MMV}(G, \delta)$.

Sometimes this question is better studied from the symmetric point of view. Given a drawing δ of a graph G, we denote by $\mathrm{MKV}(G, \delta)$ the maximum number of vertices that remain fixed when making δ plane. We refer to such vertices as *fixed* vertices. Obviously it holds that $\mathrm{MKV}(G, \delta) = n - \mathrm{MMV}(G, \delta)$, where n is the number of vertices of G. MKV measures similarity with the closest plane drawing. The corresponding problem is defined as follows.

> MaxKeptVertices(G, δ): Given a drawing δ of a planar graph G, find a plane drawing δ' of G with $\mathrm{MKV}(G, \delta)$ fixed vertices.

Let $\mathrm{MKV}(G) = \min_{\delta \text{ drawing of } G} \mathrm{MKV}(G, \delta)$ denote the maximum number of vertices of G that can be kept fixed when starting with the worst-possible drawing of G.

Our results. First, we prove that the decision versions of MaxKeptVertices and equivalently MinMovedVertices are NP-hard, see Section 2. We also prove that MinMovedVertices is hard to approximate. Namely, for any $\varepsilon \in (0, 1]$ there is no polynomial-time $n^{1-\varepsilon}$-approximation algorithm for MinMovedVertices unless $\mathcal{P} = \mathcal{NP}$.

Second, we establish a connection to a well-known graph-drawing problem, namely 1BendPointSetEmbeddability. Given a planar graph $G = (V, E)$ with n vertices we say that a graph is *k-bend (point-set) embeddable* if for any set S of n points in the plane there is a one-to-one correspondence between V and S such that G can be *k-bend (point-set) embedded on S*, i.e., the edges of G can be drawn as non-crossing simple polygonal chains with at most k bends. Kaufmann and Wiese [5] showed that (a) every 4-connected planar graph is 1-bend embeddable, (b) every planar graph is 2-bend embeddable, and (c) given a planar graph $G = (V, E)$ and set S of n points on a line, it is NP-complete to decide whether there is a correspondence between V and S that makes it possible to 1-bend embed G on S. We strengthen their result by showing that the problem remains hard even if the correspondence is given. We also show that an optimization version of the problem is hard to approximate.

Third, we give bounds on $\mathrm{MKV}(G)$ for trees and general planar graphs, see Sections 3 and 4, respectively. Table 1 summarizes the best known bounds. A lower bound of k means: we can make any drawing of any graph G in the given

Table 1. Best known bounds for MKV(G), where n is the number of vertices of G

graph class	where	lower bound	upper bound
cycles	Pach & Tardos [9]	$\lceil\sqrt{n}\,\rceil$	$O((n\log n)^{2/3})$
trees	Section 3 | 3	$\lceil\sqrt{n/3}\,\rceil$	$\lceil n/3\rceil + 4$
outerplanar graphs	Spillner & Wolff [11]	$\sqrt{n-1}/3$	$2\sqrt{n-1}+1$
planar graphs	Section 4	3	$\lceil\sqrt{n-2}\,\rceil+1$
	Spillner & Wolff [11]	$\frac{1}{3}\sqrt{\frac{2(\log n)-2}{\log\log n}-1}$	

graph class plane while keeping at least k vertices fixed. An upper bound of k means: there is an arbitrarily large graph G in the given graph class and a drawing δ of G such that at most k vertices can stay fixed when making δ plane.

Independent recent results. In May 2007, Verbitsky [14] considered the function $\mathrm{MMV}(G) = \max_{\delta \text{ plane drawing of } G} \mathrm{MMV}(G,\delta)$, to which he refers as the *shift complexity* of a graph. He, too, observed that $\mathrm{MMV}(G) \le n-3$ (i.e., $\mathrm{MKV}(G) \ge 3$) for any planar graph G with $n \ge 3$ vertices. Further he gave two linear lower bounds on $\mathrm{MMV}(G)$ depending on the connectivity of G. By reduction from independent set in line-segment intersection graphs he showed that computing $\mathrm{MMV}(G,\delta)$ is NP-hard.

In June 2007, Kang et al. [4] investigated the problem of straightening the edges of a given plane drawing (with curved edges) through vertex moves. They showed that for arbitrary large n, there exist an n-vertex graph G_n and a *plane* (curved-edge) drawing δ_n of G_n with $\mathrm{MKV}(G_n,\delta_n) = O(n^{2/3})$. Our upper bound of $O(\sqrt{n}\,)$ (see Theorem 7) is stronger, but our initial drawings are not plane.

In September 2007, Spillner and Wolff [11] showed that $\mathrm{MKV}(G)$ actually grows with the size of G and gave asymptotically tight bounds for outerplanar graphs, see Table 1.

2 Complexity

In this section, we investigate the complexity of MinMoveVertices and of 1BendPointSetEmbeddability with given vertex–point correspondence.

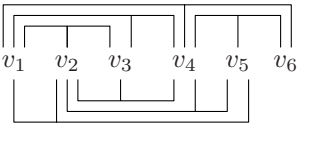


Fig. 2. Embedding of a planar 3-SAT formula

Theorem 1. *Given a planar graph G, a drawing δ of G, and an integer $K > 0$, it is NP-hard to decide whether* $\mathrm{MMV}(G,\delta) \le K$.

Proof. Our proof is by reduction from Planar3SAT, which is known to be NP-hard [7]. An instance of Planar3SAT is a 3SAT formula φ whose variable-clause graph is planar. Note that that graph can be laid out (in polynomial time) such that variables correspond to points on the x-axis and clauses correspond

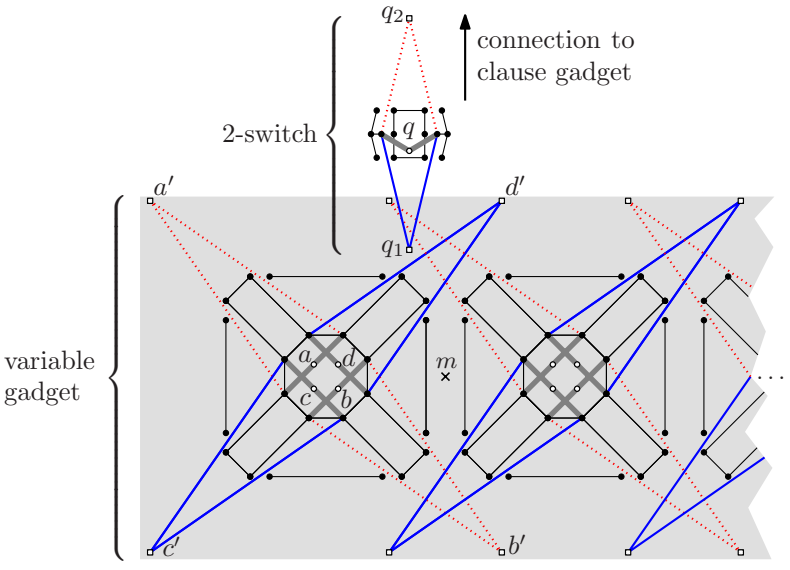


Fig. 3. Edge positions in variable gadget: immobile (thin solid black) and mobile (very thick solid gray). The predestined positions of mobile edges either correspond to *true* (thick solid blue) or to *false* (thick dotted red).

to non-crossing three-legged "combs" above or below the x-axis [6], see Fig. 2. Let v and c be the numbers of variables and clauses of φ, respectively. We now construct a graph G_φ with a straight-line drawing δ_φ such that the following holds: δ_φ can be made plane by moving at most K vertices if and only if φ is satisfiable. We fix K later.

Our graph G_φ consists of two types of substructures (or *gadgets*), modeling the variables and clauses of φ. In our gadgets, see Figs. 3 and 4, there are two types of vertices and edges; those that *may* move and those that are *meant* not to move. We refer to the two types as *mobile* and *immobile*. If φ has a satisfying truth assignment, *all* immobile (and a few mobile) vertices are fixed, otherwise at least one immobile vertex must move. In the figures, immobile vertices are marked by black disks, mobile vertices by circles, and their predestined positions by little squares. Immobile edges are drawn as thin solid black line segments, mobile edges as very thick solid gray line segments, and their predestined positions are drawn as thick colored line segments.

Now consider the gadget for some variable x in φ, see the shaded area in Fig. 3. The gadget consists of a horizontal chain of a certain number of roughly square *blocks*. Each block consists of 28 vertices and 32 edges. Each block has four mobile vertices, each incident to two very thick gray edges. In Fig. 3 the four mobile vertices of the leftmost block are labeled in clockwise order a, d, b, and c. Note that the gray edges incident to a and b intersect those incident to c and d. Thus either both a and b or both c and d must move to make the block plane. Each mobile vertex $w \in \{a, b, c, d\}$ can move into exactly one position

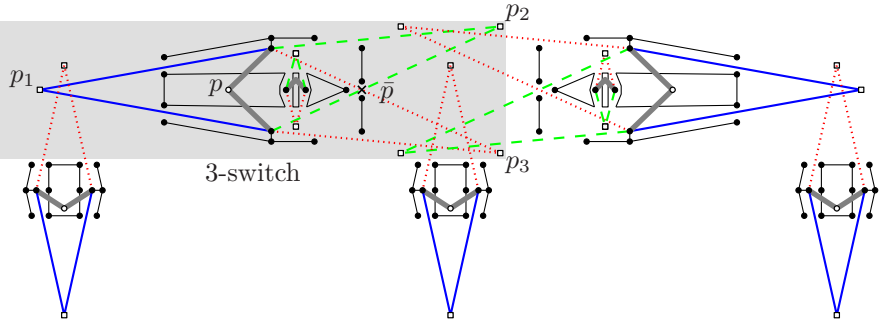


Fig. 4. A clause gadget consists of three big 2-switches and two 3-switches. Each 3-switch contains another small 2-switch. Note that not all immobile vertices are marked.

w' (up to wiggling). The resulting incident edges are drawn by thick dotted red and thick solid blue line segments, respectively. Note that neighboring blocks in the chain are placed such that the only way to make them plane simultaneously is to move *corresponding* pairs of vertices and edges. Thus either all blocks of a variable gadget use the blue line segments or all use the red line segments. These two ways to make a variable gadget plane correspond to the values *true* and *false* of the variable, respectively.

For each of the $3c$ literals in φ we connect the gadget of the corresponding variable to the gadget of the clause that contains the literal. Each block of each variable gadget is connected to a specific clause gadget above or below the variable gadget, thus there are $3c$ blocks in total. Each connection is realized by a part of G_φ that we call a *2-switch*. A 2-switch consists of 15 vertices and 14 edges. The mobile vertex q of the 2-switch in Fig. 3 is incident to two very thick gray edges that intersect two immobile edges of the 2-switch. Thus q must move. There are (up to wiggling) two possible positions, namely q_1 and q_2, see Fig. 3.

The 2-switch in Fig. 3 corresponds to a positive literal. For negated literals the switch must be mirrored either at the vertical or at the horizontal line that runs through the point m. Note that a switch can be stretched vertically in order to reach the right clause gadget. Further note that if a literal is *false*, the mobile vertex of the corresponding 2-switch must move away from the variable gadget and towards the clause gadget to which the 2-switch belongs. In that case we say that the 2-switch *transmits pressure*.

A clause gadget consists of three vertical 2-switches and two horizontal *3-switches*. A 3-switch consists of 23 vertices and 18 edges plus a small "inner" 2-switch, see the shaded area in Fig. 4. Independently from the other, each of the two 3-switches can be stretched horizontally in order to reach vertically above the variable gadget to which it connects via a 2-switch. The mobile vertex p of the left 3-switch in Fig. 4 is incident to two very thick gray edges that intersect two immobile edges of the 3-switch. Thus p must move. There are (again up to wiggling) three possible positions, namely p_1, p_2, and p_3. Note that we need the inner 2-switch, otherwise there would be a forth undesired position for moving

p, namely the one labeled $\bar{p}$ in Fig. 4. By construction a clause gadget can be made plane by only moving the mobile vertices of all switches if and only if at most two of the three big 2-switches transmit pressure, i.e., if at least one of the literals in the clause is *true*.

The graph G_φ that we have now constructed has $O(c)$ vertices, $O(c)$ edges, and $X = 26c$ crossings; $4{\cdot}3c$ in blocks and $2{\cdot}7c$ in switches. By moving any mobile vertex to any of its predestined positions, a pair of original crossings disappears. If φ is satisfiable, G_φ can be made plane by moving $K = X/2$ mobile vertices since no new crossings are introduced. If φ is not satisfiable, there is at least one pair of crossings that cannot be eliminated by moving the corresponding mobile vertex alone since all its predestined positions are blocked. Thus at least *two* vertices must be moved to eliminate that pair of crossings—and still all the other $K-1$ pairs of crossings must be eliminated by moving at least one vertex per pair, totaling in at least $K+1$ moves. Thus φ is satisfiable if and only if G_φ can be made plane by moving exactly K (mobile) vertices.

Since there is enough slack in our construction, it is possible to place vertices at integer coordinates whose total length is polynomial in the length L of a binary encoding of φ. This and the linear size of G_φ yield that our reduction is polynomial in L. □

We now consider the approximability of MinMovedVertices. Since $\mathrm{MMV}(G,\delta) = 0$ for plane drawings, we cannot use the usual definition of an approximation factor unless we slightly modify our objective function. Let $\mathrm{MMV}'(G,\delta) = \mathrm{MMV}(G,\delta)+1$ and call the resulting decision problem MinMovedVertices′. Now we can modify the above reduction to get a non-approximability result.

Theorem 2. *For any fixed real $\varepsilon \in (0,1]$ there is no polynomial-time $n^{1-\varepsilon}$-approximation algorithm for* MinMovedVertices′ *unless $\mathcal{P} = \mathcal{NP}$.*

Proof. Let n_φ be the number of vertices of the graph G_φ with drawing δ_φ that we constructed above. We go through all immobile vertices v of G_φ. Let N_v be the neighborhood of v. We replace v by a star with central vertex v adjacent to the vertices in N_v and $n_\varphi^{(3-\varepsilon)/\varepsilon}$ additional new vertices infinitesimally close to v. Let G be the resulting graph, δ its drawing, and $n \le \left(n_\varphi^{(3-\varepsilon)/\varepsilon}+1\right)\cdot n_\varphi$ the number of vertices of G. Note that φ is satisfiable if and only if $\mathrm{MMV}'(G,\delta) = \mathrm{MMV}'(G_\varphi,\delta_\varphi) = K{+}1$. Otherwise, additionally at least one complete star has to be moved, i.e., $\mathrm{MMV}'(G,\delta) \ge K + n_\varphi^{(3-\varepsilon)/\varepsilon} + 2$. Note that G can be constructed in polynomial time since ε is fixed.

Now suppose there was a polynomial-time $n^{1-\varepsilon}$-approximation algorithm $\mathcal{A}$ for MinMovedVertices′. We can bound its approximation factor by $n^{1-\varepsilon} \le \left((n_\varphi^{(3-\varepsilon)/\varepsilon}+1)\cdot n_\varphi\right)^{1-\varepsilon} \le \left(2n_\varphi^{(3-\varepsilon)/\varepsilon}\cdot n_\varphi\right)^{1-\varepsilon} = 2^{1-\varepsilon} n_\varphi^{(3-3\varepsilon)/\varepsilon} \le 2n_\varphi^{(3-3\varepsilon)/\varepsilon}$. Now let M be the number of moves that $\mathcal{A}$ needs to make δ plane. If φ is satisfiable, then $M \le \mathrm{MMV}'(G,\delta)\cdot n^{1-\varepsilon} = (K+1)\cdot n^{1-\varepsilon} \le (n_\varphi+1)\cdot 2n_\varphi^{(3-3\varepsilon)/\varepsilon} = 2n_\varphi^{(3-2\varepsilon)/\varepsilon} + O\left(n_\varphi^{(3-3\varepsilon)/\varepsilon}\right)$. On the other hand, if φ is unsatisfiable, then $M \ge \mathrm{MMV}'(G,\delta) \ge n_\varphi^{(3-\varepsilon)/\varepsilon}$. Since we can assume that n_φ is sufficiently large, the result of algorithm $\mathcal{A}$ (i.e., the number M) tells us whether φ is satisfiable. So

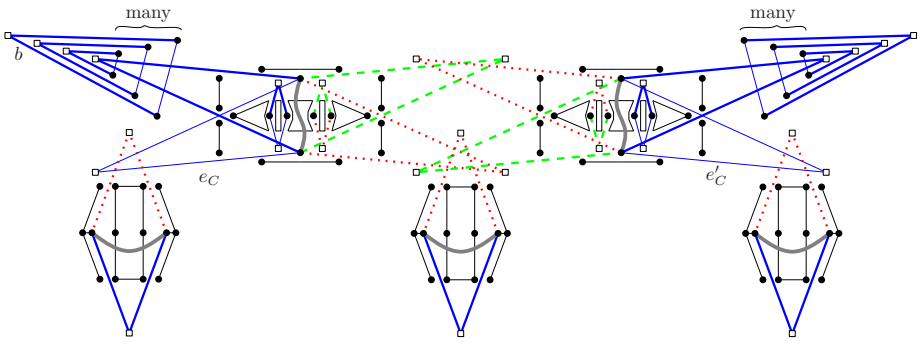


Fig. 5. Gadget of clause C for the non-approximability proof concerning the number of edges with one bend. The edges e_C and e'_C can now be drawn in *four* combinatorially different ways (thin solid blue vs. thick solid blue vs. dotted red vs. dashed green). This makes sure that there always is a drawing with at most one bend per edge. However, if the given planar 3SAT formula φ has no satisfying truth assignment, then for every truth assignment there is a clause that evaluates to *false*, and in the corresponding gadget a large number of edges of type b needs a bend.

either our assumption concerning the existence of $\mathcal{A}$ is wrong, or we have shown the NP-hard problem PLANAR3SAT to lie in $\mathcal{P}$, which in turn would mean that $\mathcal{P} = \mathcal{NP}$. □

We now state a hardness result that establishes a connection between MINMOVE VERTICES and the well-known graph-drawing problem 1BENDPOINTSETEMBEDDABILITY. The proof uses nearly the same gadgets as in the proof of Theorem 1: Set G'_φ to a copy of G_φ where each length-2 path (u, v, w) containing a mobile vertex v is replaced by the edge $\{u, w\}$. The vertices of G'_φ are mapped to the corresponding vertices in δ_φ. Then it is not hard to see that G'_φ has a 1-bend drawing iff the given planar-3SAT formula φ is satisfiable.

Theorem 3. *Given a planar graph $G = (V, E)$ with $V \subset \mathbb{R}^2$, it is NP-hard to decide whether G has a plane drawing with at most one bend per edge.*

Now suppose we already know that G has a plane drawing with at most one bend per edge. Then it is natural to ask for a drawing with as few bends as possible. Let $\beta(G)$ be 1 plus the minimum number of bends over all plane one-bend drawings of G. Then we can show the following hardness-of-approximation result concerning bend minimization.

Corollary 1. *Given a fixed $\varepsilon \in (0, 1]$ and a graph $G = (V, E)$ with $V \subset \mathbb{R}^2$ that has a plane one-bend drawing, it is NP-hard to approximate $\beta(G)$ within a factor of $n^{1-\varepsilon}$.*

For the proof we slightly change the clause gadget in the proof of Theorem 1, see Figure 5. For the calculations, see the proof of Theorem 2.

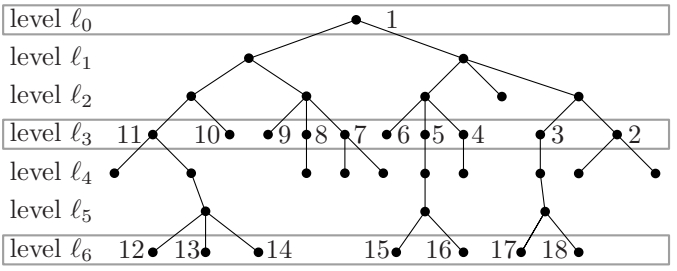


Fig. 6. The ordering of vertices in L_0

3 Trees

In this section we give a lower bound on MKV for trees. We use the following well-known theorem.

Theorem 4 (Erdős and Szekeres [1]). *Let $A = (a_1, \ldots, a_n)$ be a sequence of n different real numbers. If $n \geq sr + 1$ then A has an increasing subsequence of $s + 1$ terms or a decreasing subsequence of $r + 1$ terms.*

In particular, this theorem implies that a sequence of n distinct integers always contains a monotone subsequence of length at least $\sqrt{n-1} + 1 \geq \lceil \sqrt{n} \rceil$.

Theorem 5. $\mathrm{MKV}(T) \geq \lceil \sqrt{n/3} \rceil$ *for any n-vertex tree T.*

Proof. Let δ be an arbitrary drawing of T. We pick an arbitrary root r of T. Let $h \geq 0$ be the height of T with respect to r. For $i = 0, \ldots, h$ let *level* ℓ_i be the set of vertices of T that are at tree distance i from r. For $j \in \{0, 1, 2\}$ let L_j be the union of all ℓ_i with $i \equiv j \bmod 3$. According to the pigeon-hole principle at least one of the three sets, say L_0, contains at least $n/3$ vertices. We label the vertices of L_0 with the integers from 1 to $|L_0|$ such that (i) all vertices in the same level are consecutive in alternating directions, i.e., from left to right for every even-numbered level in L_0 and from right to left for every odd-numbered level in L_0, and (ii) a level closer to the root gets smaller labels, see Fig. 6.

Let ℓ be a line, say the x-axis, such that the projection π orthogonal to ℓ does not map any two vertices of the drawing δ to the same point. The image of π induces an ordering of the vertices in L_0. By Theorem 4, this ordering contains a monotone subsequence $F_0 \subset L_0$ of at least $\lceil \sqrt{n/3} \rceil$ vertices.

We fix the vertices in F_0. First we move the vertices in $L_0 \setminus F_0$. Let B be an axis-parallel rectangle such that $F_0 \subset B$ and no point of F_0 lies on the boundary of B. If $1 \notin F_0$ ($|L_0| \notin F_0$), move it to any point on the left (right) edge of B. For any two vertices j, k that are consecutive in $F_0 \cup \{1, |L_0|\}$, move vertices $j + 1, \ldots, k - 1$ in an equidistant manner on the line segment $\overline{jk}$.

We draw nested $\sqcap$-shaped corridors between vertices in ℓ_i and their respective children in ℓ_{i+3} if i is even. If i is odd, we use $\sqcup$-shaped corridors, see Fig. 7. Due to our labeling scheme no two such corridors intersect. Finally, the vertices

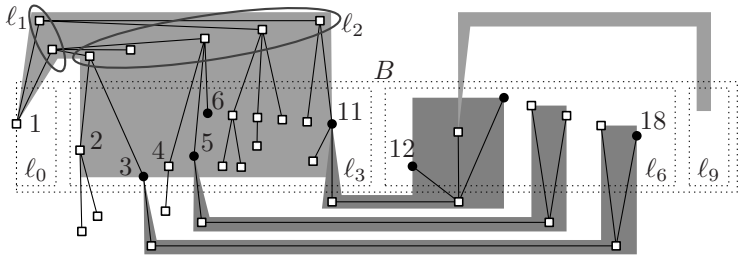


Fig. 7. Corridors connect the vertices in L_0. Vertices in F_0 are marked by black disks.

in $L_1 \cup L_2$ go to positions near the bends of the corridors (see levels ℓ_1 and ℓ_2 in Fig. 7), which allows us to connect vertices in ℓ_i and ℓ_{i+3} by two-bend edges. □

Remark 1. The proof of Theorem 5 yields an $O(n \log n)$-time algorithm for making drawings of trees plane. It uses that the longest monotone subsequence of a sequence of n integers can be found in $O(n \log n)$ time [10].

4 Planar Graphs

We now give bounds for the case of general planar graphs. We start with a rather trivial lower bound.

Theorem 6. $\mathrm{MKV}(G) \geq 3$ *for any planar graph* G *with* $n \geq 3$ *vertices.*

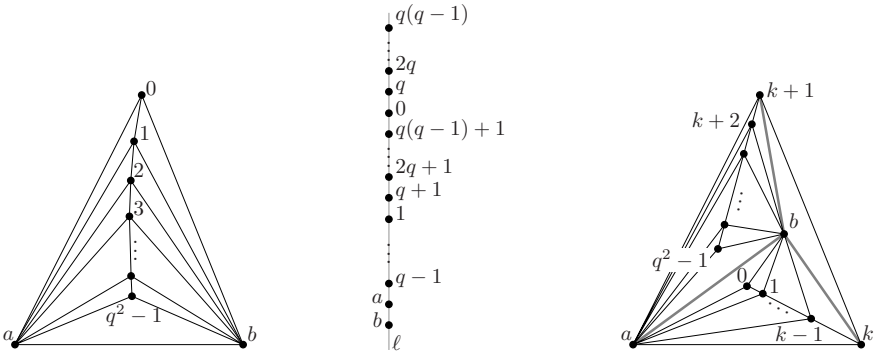
Proof. Let δ be any drawing of G. Any planar graph admits a plane drawing δ_1 in which no three points are collinear and a plane drawing δ_2 in which some triplet of points is collinear. If there are three vertices v_1, v_2, and v_3 whose images under δ are not collinear, we can find an affine transform L that maps $\delta_1(v_i)$ to $\delta(v_i)$. Since $L \circ \delta_1$ is a plane drawing of G that agrees with δ on $\{v_1, v_2, v_3\}$ it follows that $\mathrm{MKV}(G, \delta) \geq 3$. If the images of all vertices are aligned under δ, we apply the same argument with δ_2. □

We now give an upper bound for general planar graphs that is better than the upper bound $O((n \log n)^{2/3})$ of Pach and Tardos [9] for cycles. Our construction uses the sequence $\sigma_q =$

$$(q-1)q, (q-2)q, \ldots, 2q, q, \underline{0}, 1+(q-1)q, \ldots, 1+q, \underline{1}, \ldots, q^2-1, \ldots, (q-1)+q, \underline{q-1}.$$

Note that σ_q can be written as $(\sigma_q^0, \sigma_q^1, \ldots \sigma_q^{q-1})$, where $\sigma_q^i = ((q-1)q+i, (q-2)q+i, \ldots, 2q+i, q+i, i)$ is a subsequence of length q. Thus σ_q consists of q^2 distinct numbers. Note that the longest monotone subsequence of σ_q has length q.

Theorem 7. *For any integer* n_0 *there exists a planar graph* G *with* $n \geq n_0$ *vertices and* $\mathrm{MKV}(G) \leq \lceil \sqrt{n-2} \rceil + 1$.



(a) plane drawing of case 1 **(b)** drawing δ (w/o edges) **(c)** plane drawing of case 2

Fig. 8. Drawings of graph G_q (proof of Theorem 7)

Proof. For $q \geq 1$ we define the graph G_q as a chain of q^2 vertices all connected to the two endpoints of an edge $\{a, b\}$, see Fig. 8a. Let δ be the drawing of G_q where the vertices forming the chain are placed on a vertical line ℓ in the order given by σ_q. We place the vertices a and b below the others on ℓ, see Fig. 8b. Let δ' be a plane drawing of G_q with $\mathrm{MMV}(G_q, \delta) = d(\delta, \delta')$. Since all faces of G_q are 3-cycles, the outer face in δ' is a triangle. All faces of G_q contain a or b. This has two consequences. First, a and b must move to new positions in δ', otherwise all other vertices would have to move. Second, at least one of them, say a, appears on the outer face.

Case 1: Vertex b also lies on the outer face.
Then there are just two possibilities for the embedding of G_q: as in Fig. 8a or with the indices of all vertices reversed, i.e., vertex i becomes $q^2 - i - 1$. Now let $0 \leq i < j < k \leq q^2-1$ be three fixed vertices. By symmetry we can assume that j lies in $\Delta(a, b, i)$. Then k also lies in $\Delta(a, b, i)$ since the chain connecting j to k does not intersect the sides of this triangle. Note that k cannot lie between i and j on ℓ as otherwise one of the edges $\{a, k\}$ and $\{b, k\}$ would intersect the polygonal chain connecting i to j. Thus, each triplet of fixed vertices forms a monotone sequence along ℓ. This in turn yields that *all* fixed vertices in $\{0, \ldots, q^2-1\}$ form a monotone sequence along ℓ. Due to the construction of σ_q such a sequence has length at most $q = \lceil \sqrt{n-2} \rceil$.

Case 2: Vertex b does not lie on the outer face.
Then the outer face is of the form $\Delta(a, k, k+1)$ with $0 \leq k \leq q^2 - 2$. The three edges $\{b, a\}$, $\{b, k\}$, and $\{b, k+1\}$ incident to b split $\Delta(a, k, k+1)$ into the three triangles $\Delta(a, k, b)$, $\Delta(a, b, k+1)$, and $\Delta(b, k, k+1)$, see Fig. 8c. Every vertex of δ' lies in one of them. Since δ' is plane, vertex $k-1$ must belong to $\Delta(a, k, b)$ and, by induction, so do all vertices $i \leq k$; similarly, all vertices $i \geq k+1$ lie in $\Delta(a, b, k+1)$. We can thus apply the argument of case 1 to each of the two subgraphs contained in $\Delta(a, b, k)$ and $\Delta(a, b, k+1)$. This yields two non-overlapping monotone sequences of length q each. Note, however, that both

most be increasing or both decreasing, since one type forces a to the left and b to the right of ℓ and the other does the opposite. Now Observation 2 of [11] yields that at most $q+1$ vertices can remain fixed. □

Remark 2. The drawing δ in the proof of Theorem 7 can be slightly perturbed so that no three vertices are aligned.

5 Conclusion

Inspired by John Tantalo's on-line game *Planarity* we have introduced a new and apparently simple graph-drawing problem, which turned out to be rather difficult. There are many open questions. On the computational side, we showed inapproximability for MinMovedVertices. However, this does not imply anything for the approximability of MaxKeptVertices, which remains open.

Are the problems in $\mathcal{NP}$? Are they hard for cycles? What about parameterized complexity? On the combinatorial side, there are large gaps to be filled and other classes of planar graphs to be studied.

References

1. Erdős, P., Szekeres, G.: A combinatorial problem in geometry. Compos. Math. 2, 463–470 (1935)
2. Fáry, I.: On straight-line representation of planar graphs. Acta Sci. Math. (Szeged) 11, 229–233 (1948)
3. Goaoc, X., Kratochvíl, J., Okamoto, Y., Shin, C.-S., Wolff, A.: Moving vertices to make drawings plane (June 2007), http://arxiv.org/abs/0706.1002
4. Kang, M., Schacht, M., Verbitsky, O.: How much work does it take to straighten a plane graph out? (June 2007), http://arxiv.org/abs/0707.3373
5. Kaufmann, M., Wiese, R.: Embedding vertices at points: Few bends suffice for planar graphs. J. Graph Algorithms Appl. 6(1), 115–129 (2002)
6. Knuth, D.E., Raghunathan, A.: The problem of compatible representatives. SIAM J. Discr. Math. 5(3), 422–427 (1992)
7. Lichtenstein, D.: Planar formulae and their uses. SIAM J. Comput. 11(2), 329–343 (1982)
8. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. J. Visual Languages and Computing 6(2), 183–210 (1995)
9. Pach, J., Tardos, G.: Untangling a polygon. Discrete Comput. Geom. 28(4), 585–592 (2002)
10. Schensted, C.: Longest increasing and decreasing subsequences. Canadian Journal of Mathematics 13, 179–191 (1961)
11. Spillner, A., Wolff, A.: Untangling a planar graph (September 2007), http://arxiv.org/abs/0709.0170
12. Stein, S.K.: Convex maps. Proc. Amer. Math. Soc. 2, 464–466 (1951)
13. Tantalo, J.: Planarity (2007), http://planarity.net/
14. Verbitsky, O.: On the obfuscation complexity of planar graphs (May & June 2007), http://arxiv.org/abs/0705.3748
15. Wagner, K.: Bemerkungen zum Vierfarbenproblem. Jahresbericht Deutsch. Math.-Verein. 46, 26–32 (1936)

Point-Set Embedding of Trees with Edge Constraints*

(Extended Abstract)

Emilio Di Giacomo[1], Walter Didimo[1], Giuseppe Liotta[1], Henk Meijer[2], and Stephen Wismath[3]

[1] Dip. di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia
{digiacomo, didimo, liotta}@diei.unipg.it
[2] Roosevelt Academy, The Netherlands
h.meijer@roac.nl
[3] Department of Mathematics and Computer Science, University of Lethbridge
wismath@cs.uleth.ca

Abstract. Given a graph G with n vertices and a set S of n points in the plane, a *point-set embedding* of G on S is a planar drawing such that each vertex of G is mapped to a distinct point of S. A *geometric point-set embedding* is a point-set embedding with no edge bends. This paper studies the following problem: The input is a set S of n points, a planar graph G with n vertices, and a geometric point-set embedding of a subgraph $G' \subset G$ on a subset of S. The desired output is a point-set embedding of G on S that includes the given partial drawing of G'. We concentrate on trees and show how to compute the output in $O(n^2 \log n)$ time and with at most $1 + 2\lceil k/2 \rceil$ bends per edge, where k is the number of vertices of the given subdrawing. We also prove that there are instances of the problem which require at least $k - 3$ bends for some of the edges.

1 Introduction

Let G be a planar graph with n vertices and let S be a set of n points in the plane. A *point-set embedding of G on S* is a crossing-free drawing of G such that each vertex is represented as a distinct point of S and the edges are polygonal chains. The problem of computing a point-set embedding of a graph, also known as the *point-set embeddability problem*, has been extensively studied both when the mapping of the vertices to the points is chosen by the drawing algorithm and when it is partially or completely given as part of the input. A limited list of papers about different versions of the point-set embeddability problem includes, for example, [1,2,3,5,6,7,8,12,15].

This paper studies a natural extension of the point-set embeddability problem. It is assumed to have a mapping of some edges of G to segments defined on S

* Research partially supported by the MIUR Project "MAINSTREAM: Algorithms for massive information structures and data streams" and by NSERC.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 113–124, 2007.

and the goal is to compute a point-set embedding of G that includes the given segments. More precisely, we focus on trees and study the following question: The input is a set S of n points, a tree T with n vertices, and a point-set embedding of a subtree $T' \subset T$ on a subset of S such that all edges of this partial drawing are straight-line segments. The desired output is a *constrained point-set embedding* of T on S, i.e. a point-set embedding of T on S that includes the given partial drawing of T'.

From the application point of view, the point-set embeddability problem is relevant in those contexts where the display of the vertices is constrained to use a set of prescribed locations. Our variant adds the constraint that a portion of the graph is already drawn; this can be important for example to preserve the user's mental map when a certain subgraph of an evolving network does not change over time. Again, representing certain edges as straight-line segments and placing their end-vertices at specific locations can be used to emphasize the importance of these objects with respect to other objects of the graph.

We recall that a recent paper on extending a partial straight-line drawing is [16]. Given a planar graph G and a planar straight-line drawing Γ of a subgraph of G, the author of [16] shows that it is NP-hard deciding whether G admits a planar straight-line drawing including Γ. The main difference between the problem studied in [16] and the one investigated in this paper is that, when extending the partial straight-line drawing, we have fixed locations for the vertices and we allow bends along the edges.

The main contribution of this paper is to provide lower and upper bounds to the maximum number of bends per edge in a constrained point-set embedding of a tree. An outline of the results is as follows.

- We prove that a constrained point-set embedding of a tree on a set of points can require one edge bend even if the partial drawing consists of just a single edge. We recall that every tree with n vertices admits a straight-line point-set embedding onto any set of n points in general position [3,8].
- We extend the above result by showing a lower bound that depends on the number of vertices of the given subdrawing of the tree. Namely we prove there exist trees with $n > 7$ vertices and partial drawings with $k < n$ vertices such that any constrained point-set embedding has at least $n-k$ edges, each having at least $k-3$ bends.
- We describe a drawing algorithm that computes a constrained point-set embedding of a tree in $O(n^2 \log n)$ time and with at most $1 + 2\lceil k/2 \rceil$ bends per edge, where n is the number of vertices of the tree and k is the number of vertices of the given subdrawing. We remark that the difference between such an upper bound and the lower bound mentioned above is at most 5.

The proof of the upper bound is based on the partial solution of a computational geometry problem that in our opinion is of independent interest. Kaneko and Kano [9,10] studied the problem of computing a point-set embedding with straight-line edges of a forest F of rooted trees such that the location of the root of each tree of F is part of the input. Kaneko and Kano show that the drawing

can always be computed for special types of forests (rooted star forests or forests of trees where the sizes of any two trees differ by at most one) but the problem is still open in the general case.

One of the basic ingredients of our upper bound technique sheds more light on the problem described above. Namely, let $T_0, \ldots, T_{h-1}$ be a forest of trees with n vertices in total. Let $S = \{p_0, \ldots, p_{n-1}\}$ be a set of n points in general position such that $p_0, \ldots, p_{h-1}$ are points of the convex hull of S. We describe an $O(n^2 \log n)$ time procedure to compute a straight-line point-set embedding of the forest such that the root of T_i is on p_i $(i = 1, \ldots, h-1)$.

The remainder of this paper is organized as follows. Preliminary definitions are in Section 2. The study of the constrained point-set embeddability problem for trees is in Section 3. Lower bounds are provided in Subsection 3.1 and an upper bound is given in Subsection 3.2. Conclusions and open problems are in Section 4.

2 Preliminaries

We assume familiarity with basic notions of graph drawing and of computational geometry (see, e.g., [4,11,13,17]).

Let $G = (V, E)$ be a planar graph with n vertices and let S be a set of n points in the plane. A *point-set embedding of G on S*, denoted as $\Gamma(G, S)$, is a planar drawing of G such that each vertex is mapped to a distinct point of S. $\Gamma(G, S)$ is called a *geometric point-set embedding* if each edge is drawn as a straight-line segment.

Let $D(S)$ be a straight-line drawing whose vertices are points of a subset of S. We say that $D(S)$ is a *partial drawing* of G on S if it represents a graph isomorphic to a subgraph of G. A *constrained point-set embedding* $\Gamma(G, D(S))$ is a point-set embedding of G on S such that $D(S)$ is a subdrawing of $\Gamma(G, D(S))$.

For example, Figure 1 shows a graph G, a partial drawing $D(S)$ of G on a set S of points, and a constrained point-set embedding $\Gamma(G, D(S))$.

In the remainder of the paper, we say that the points of S are in *general position* if no three points of S lie on the same line. A corner v of a polygon

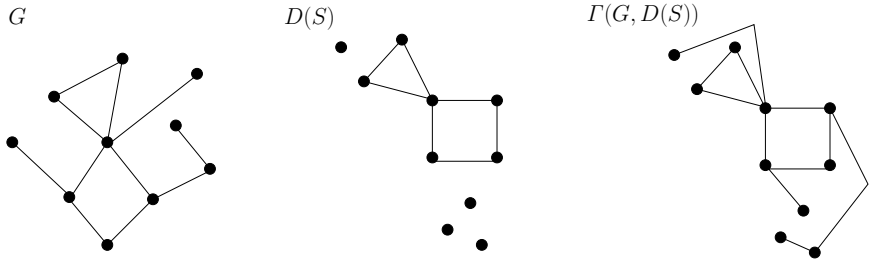


Fig. 1. A planar graph G. A set S of points and a partial drawing $D(S)$ of G. A constrained point-set embedding $\Gamma(G, D(S))$ with at most one bend per edge.

in the plane is said to be a *reflex corner* if the angle at v inside the polygon is greater than 180 degrees.

3 Constrained Point-Set Embeddings of Trees

In this section we investigate the constrained point-set embeddability problem for a tree T on a set S of points. We present lower and upper bounds to the maximum number of bends per edge in a constrained point-set embedding $\Gamma(T, D(S))$. These bounds depend on the number of vertices of the partial drawing $D(S)$.

3.1 Lower Bounds

We first show that there exist a tree and a set of points such that, even for a partial drawing consisting of a single edge, a constrained point-set embedding requires at least one edge bend. A more general lower bound is then provided.

Lemma 1. *There exist a tree T of n vertices, a set S of n points, and a partial drawing $D(S)$ of T on S consisting of a single edge, such that every constrained point-set embedding $\Gamma(T, D(S))$ has an edge with at least 1 bend.*

Sketch of Proof: Consider the tree T and the drawing $D(S)$ in Figure 2. Let s denote the single edge of $D(S)$. Let v_0, v_1, v_2, and v_3 denote four vertices of T as illustrated, i.e., v_0, v_1, v_2, and v_3 form a path from the root of T to a leaf. Assume that we have a drawing $\Gamma(T, D(S))$ without bends. Notice that no point above s can be connected to a point below s without a bend. Because of symmetry we only need consider three cases: either (v_0, v_1), (v_1, v_2), or (v_2, v_3) is mapped to s.

(v_0, v_1) **is mapped to** s**:** Removal of v_0 and v_1 splits T into three sub-trees, one with 3 vertices and two with 4 vertices each. Each sub-tree either has to be drawn above or below s. This cannot be done since there are 6 points above and 5 points below s and no combination of $\{3, 4, 4\}$ adds up to 5.

(v_1, v_2) **is mapped to** s**:** Removal of v_1 and v_2 splits T into three sub-trees, two with 1 and one with 9 vertices. No combination of $\{1, 1, 9\}$ adds up to 5.

(v_2, v_3) **is mapped to** s**:** Removal of v_2 and v_3 splits T into two sub-trees, one with 1 and one with 10 vertices. No combination of $\{1, 10\}$ adds up to 5. □

Lemma 2. *There exist a tree T with $n > 7$ vertices, a set S of n points, and a partial drawing $D(S)$ of a tree with $7 \leq k < n$ vertices, such that every constrained point-set embedding $\Gamma(T, D(S))$ has $n - k$ edges each having at least $k - 3$ bends.*

Sketch of Proof: Consider a tree T consisting of a path $v_0, v_1, \ldots, v_{k-3}$ of $k - 2$ vertices, a vertex u adjacent to v_1, a vertex w adjacent to v_2 and $n - k$ vertices adjacent to v_{k-3} (see Figure 3 for an illustration with $k = 9$ and $n = 14$). Let T' be the subgraph of T containing all vertices of T except the $n - k$ vertices adjacent to v_{k-3}. There is exactly one subgraph in T isomorphic to T', and the

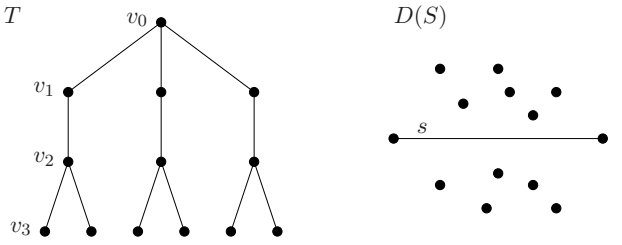


Fig. 2. A tree T, a set S of points, and a partial drawing $D(S)$ with a single edge s

remaining $n-k$ vertices of T are adjacent to v_{k-3} which is the only leaf node of T' with a degree 2 neighbor. Notice that for such a leaf to exist we require $k \geq 7$. Let $D(S)$ be a partial drawing of T on S constructed as shown in Figure 3; the edges of $D(S)$ (the solid edges in the figure) form a tree isomorphic to T'. Since, as already observed, there is only one subgraph in T isomorphic to T', the edges that we must add to $D(S)$ to get a drawing $\Gamma(T, D(S))$ are those adjacent to v_{k-3} (see, e.g., edge e in Figure 3). As also shown in the figure, it is not hard to see that each of these edges requires at least $k-3$ bends. □

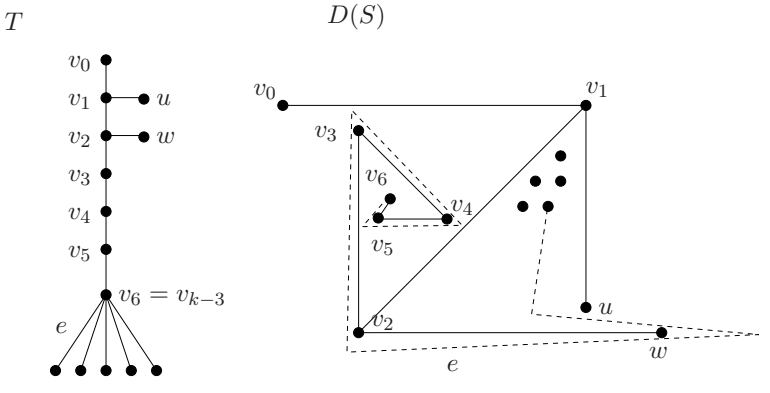


Fig. 3. Illustration of the proof of Lemma 2. A tree T with $n = 14$ vertices, a set S of n points, and a partial drawing $D(S)$ (with solid edges) with $k = 9$ vertices. Every $\Gamma(T, D(S))$ requires $n - k = 5$ edges each having at least $k - 3 = 6$ bends (see for example the dashed edge).

3.2 Upper Bound

Let T be a tree with n vertices and let S be a set of n points. In this section we show that if $D(S)$ is a partial drawing of T on S such that $D(S)$ represents a tree with k vertices, then we can always construct a constrained point-set embedding $\Gamma(T, D(S))$ with at most $1 + 2\lceil k/2 \rceil$ bends per edge. This means that each edge of T that we add to complete $D(S)$ is drawable with a number of bends that is

linear in the number of vertices of $D(S)$ and that does not depend on the size of T. Notice that the bound $1 + 2\lceil k/2 \rceil$ is equal either to $k+1$ (if k is even) or to $k+2$ (if k is odd). This implies that the difference between this upper bound and the lower bound given in Lemma 2 is at most 5.

We start by providing two lemmas that are the technical foundation of our drawing technique. The first lemma sheds more light on a point-set embeddability problem studied by Kaneko and Kano [9,10].

Lemma 3. *Let G consist of a forest of trees $T_0, T_1, \ldots, T_{h-1}$. Let $T_i = (V_i, E_i)$ for all $0 \leq i < h$. Let $S = \{p_0, p_1, \ldots, p_{n-1}\}$ be a set of points in general position such that $p_0, p_1, \ldots, p_{h-1}$ are points of the convex hull of S. There exists an $O(n^2 \log n)$-time algorithm that computes a geometric point-set embedding $\Gamma(G, S)$ such that the root of T_i is on p_i $(0 \leq i < h)$.*

Sketch of Proof: Let $CH(S)$ be the convex hull of S. Without loss of generality, assume that $p_0, p_1, \ldots, p_{h-1}$ occur in this order on the boundary of $CH(S)$ in clockwise order (if this is not the case, we can simply reorder them).

We first show that we can find a line a with the following properties: (i) a does not intersect any point p_i of S and there are points from S on both sides of a; (ii) denoted by $I \subset \{0, 1, \ldots, h-1\}$ the set of indices for which all convex hull points p_j, with $j \in I$, lie on one side of a, we have that the total number of points on that side is equal to $\sum_{j \in I} |V_j|$.

We call such a line a *dividing line*. An example of dividing line is shown in Figure 4. We can use a ham-sandwich type argument to prove that a dividing line exists. We say that a side of a is too light if we have convex hull points p_j with $j \in I \subset \{0, 1, \ldots, h-1\}$ to that side of a and the total number of points to that side of a is smaller than $\sum_{j \in I} |V_j|$. If one side of a is too light, the other side is said to be too heavy.

Consider points p_0 and p_1 on $CH(S)$. Let a_0 and a_1 be lines through p_0 and p_1 and such that any other point of a_0 and a_1 is outside the polygon defined by $CH(S)$ (refer to Figure 4 for an illustration). Let p be the intersection point of a_0 and a_1 (the proof will still work if p is a point at infinity). We start with line $a = a_0$ and rotate a around p in the counterclockwise direction until $a = a_1$. We can always slightly perturb a_0 and a_1 (and hence p) in such a way that a never intersects two points of S at the same time. Without loss of generality assume that the range of motions for a does not include a horizontal line and that when $a = a_0$, all remaining points of S lie to the right of a when moving along a toward p. If we rotate a slightly away from a_0, only p_0 lies to the left of a. If T_0 consists only of its root, a is a dividing line and we are done; otherwise the left side of a is too light. If we place a such that only p_1 is on its right, then either T_1 consists only of a root and we are done, or the left side of a is too heavy. If the left side of a is too light and during the rotation of a from a_0 to a_1 it passes a point p_j with $0 \leq j < h$, the left side of a remains too light. Since during rotation at any time at most one point moves from the right to the left side of a, and since in the beginning the left side of a is too light and at the end the left side of a is too heavy, it follows that at some moment a is a dividing line.

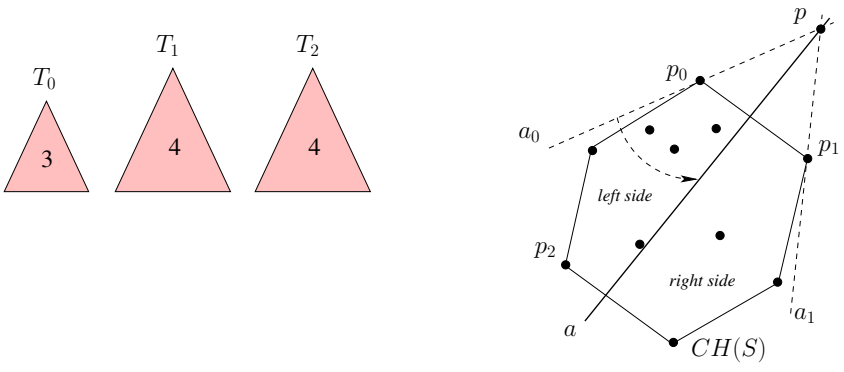


Fig. 4. An example of dividing line a. G consists of three trees T_0, T_1, and T_2, where T_0 has 3 vertices and T_1, T_2 have 4 vertices each. On one side of a there are the root points p_0 and p_2 for trees T_0 and T_2, and a total number of points equal to the number of vertices of T_0 and T_2; on the other side of a there are the points for drawing T_1.

The complexity of finding a dividing line is $O(n \log n)$, because we can first radially sort the points of S around p and then execute a scan-line algorithm from a_0 to a_1 to find a dividing line. Also, if p must be perturbed by an $\epsilon > 0$ to avoid that it is collinear with any two points of S before starting the search of a dividing line, such an ϵ can be determined in $O(n)$ time, using the radial sorting of the points around p itself.

Once we have found a dividing line, the polygon whose boundary is $CH(S)$ is divided into two subregions. By recursively applying the same procedure on each of the two subregions we can find dividing lines that split $CH(S)$ into convex subregions $P_0, P_1, \ldots, P_{h-1}$ such that each P_i contains $|V_i|$ vertices. Therefore we find the required drawing by executing the following algorithm:

Step A. Divide $CH(S)$ into convex subregions $P_0, P_1, \ldots, P_{h-1}$ such that each P_i contains $|V_i|$ vertices.
Step B. Draw each T_i inside P_i with the technique of Bose et al. [3].

Since all h dividing lines can be found in $O(h \cdot n \log n)$ time, where $h \leq n$, and the algorithm of Bose et al. [3] runs in $O(n \log n)$ time, it follows that the given algorithm runs in $O(n^2 \log n)$. □

The next lemma extends the previous result to the case where the roots of the trees are placed on the boundary of a non-convex polygon. In this case, the number of bends along the edges depend on the number of reflex corners of the polygon.

Lemma 4. *Let G consist of a forest of trees $T_0, T_1, \ldots, T_{h-1}$. Let $T_i = (V_i, E_i)$ for all $0 \leq i < h$. Let $S = \{p_0, p_1, \ldots, p_{n-1}\}$ be a set of points in general position such that $p_0, p_1, \ldots, p_{h-1}$ are points along the boundary of a polygon P and the remaining points of S are inside P. Also, let k be the number of reflex*

corners of P. There exists an $O(n^2 \log n)$-time algorithm that computes a point-set embedding $\Gamma(G, S)$ inside P such that the root of T_i is on p_i $(0 \leq i < h)$ and each edge of $\Gamma(G, S)$ has at most $2\lceil k/2 \rceil$ bends.

Sketch of Proof: For an illustration of this proof, refer to Figure 5. In the figure the forest to be drawn consists of two trees, T_0 and T_1, and the polygon P has three reflex corners. We prove the lemma by construction:

Step 1. We partition P into $k+1$ convex polygons, for example by iteratively drawing a bisector from each reflex vertex until this bisector hits another line segment. We perturb the subdivision in such a way that no point from S lies on any of the added subdivision edges and so that none of the convex polygons has an angle equal to π. We call the added subdivision edges *dummy edges* (the dashed edges in the figure).

Step 2. Consider the dual graph of this subdivision, find a spanning tree T of the dual graph and select a node r of T with the property that the number of edges in T from r to any leaf node of T is at most $\lceil k/2 \rceil$. Make r the root of T. In the following, for any node v of T, P_v will denote the convex polygon corresponding to v (P_r is the convex polygon corresponding to the root). In the figure, the nodes of the dual graph of the subdivision are represented by big squares and the edges of the selected spanning tree are in bold.

Step 3. The objective of this step is to add extra points on the boundary of P_r so that each of these points corresponds to a distinct point of S that does not lie in P_r. Let $S' = S$. We add *dummy points* to S' by executing a post-order traversal of T. For each visited node v of T distinct from r we do the following. Let e be the dummy edge that separates P_v from the polygon corresponding to its parent node in T. Recall that no points from S lie on e, except possibly at its end-points. Let S_v be the set of points in $S' \cap P_v$ except possibly those at some end-point of e (we include every other point of S that is on the boundary of P_v). Place $|S_v|$ dummy points on e in such a way that none of the points on e lies on a line through two points from S_v. Construct a straight-line perfect planar matching from the $|S_v|$ points in P_v to the $|S_v|$ points on e. Add to S' the dummy points placed on e. In the figure, the dummy points are represented by empty circles.

Step 4. After the execution of Step 3 all nodes of T have been visited except the root r of T. Notice that there are n points in $S' \cap P_r$. In order to guarantee that no three points of S' are collinear, we slightly modify the boundary of P_r, by replacing each dummy edge of P_r with a "slightly convex" polygonal chain. More precisely, if e is a dummy edge of P_r such that n_e dummy points are placed on e, we replace e with a convex polygonal chain C_e such that C_e has n_e vertices and it does not change the inside/outside relations of the points of S with respect to P_r. Then we move each of the n_e dummy points on a distinct vertex of C_e, in such a way that the linear ordering of these points along C_e is the same they had along e.

Step 5. Compute a straight-line drawing of $\Gamma(G, S' \cap P_r)$ inside P_r by using Lemma 3; the root of each T_i $(i \in \{0, \ldots, h-1\})$ is placed either on p_i (if

p_i belongs to the boundary of P_r) or on the dummy point of the boundary of P_r that corresponds to p_i. Finally, we replace all edges of the drawing connected to a dummy point by narrow tunnels and we use these tunnels to planarly draw the edges of the tree; the number of dummy nodes traversed by an edge of the tree corresponds to the number of bends of that edge in the final drawing.

In Step 5, any edge of G is drawn from a point p of S via dummy points until it reaches P_r. Every time an edge passes through a point on a dummy edge, a bend is added. Since the longest path in the spanning tree T of the subdivision passes through $\lceil k/2 \rceil$ dummy edges and an edge of the drawing of G may connect two points that lie in two (possibly coincident) polygons whose corresponding nodes are at distance $\lceil k/2 \rceil$ from the root r of T, the number of bends per edge is at most $2\lceil k/2 \rceil$.

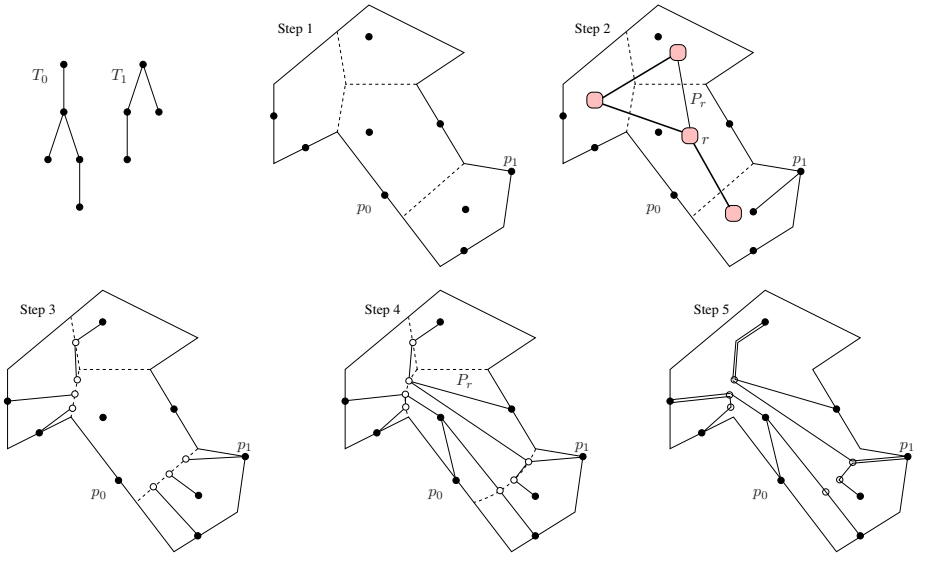


Fig. 5. Steps of the constructive proof of Lemma 4

We now briefly discuss the time complexity of the drawing algorithm described above. Step 1 can be performed in $O(n^2)$ time by using standard partitioning techniques of a polygon into convex regions (see, e.g., [14]). Step 2 is executed in $O(k)$ time, because T has $k+1$ vertices. Regarding Step 3, for each polygon $P_v \neq P_r$ we need to project the points of S_v on a dummy edge e of the boundary of P_v, so that the straight-line edges used in the projection do not intersect. To accomplish this we can for example radially sort the points of S_v by rotating counterclockwise the line containing e around the middle point q of e, and then project all points of S_v with slope less than $\pi/2$ onto points of e to the right of q, and points of S_v with slope greater than $\pi/2$ onto points of e to the left of q (see

Figure 6 for an example). Since each S_v contains at most n points, this step can be executed in $O(n^2 \log n)$ time. Step 4 can be executed in $O(n)$ time, and the complexity of Step 5 is dominated by the complexity of the drawing algorithm of Lemma 3, which is $O(n^2 \log n)$. □

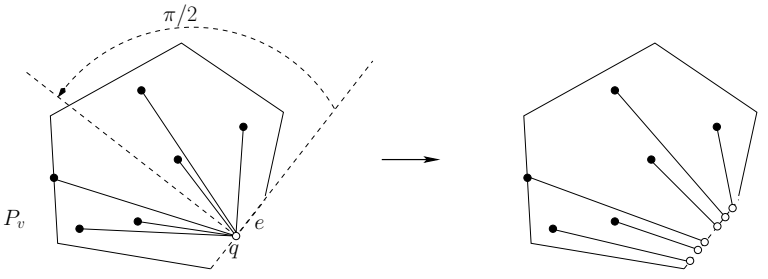


Fig. 6. Illustration of the technique used in Step 3 in the proof of Lemma 4

We can now prove the following theorem.

Theorem 1. *Let T be a tree with n vertices and let S be a set of n points in general position. Let $D(S)$ be a partial drawing of T representing a subtree with k vertices. There exists a constrained point-set embedding $\Gamma(T, D(S))$ with at most $1 + 2\lceil k/2 \rceil$ bends per edge, which can be computed in $O(n^2 \log n)$ time.*

Sketch of Proof: Since $D(S)$ is a partial drawing of T on S, it is a straight-line drawing of a subtree T' of T. We first construct a polygon P that follows the boundary of T' and that leaves T' outside (refer to Figure 7 for an illustration). More precisely, draw a convex polygon P that properly contains S and then modify it as follows: Find a location p on a side of P from which we can draw a straight-line segment to a location q on an edge of $D(S)$. Cut P at p and draw a line segment from p in the direction of q until it almost reaches q. We then continue to draw line segments that trace around the edges of $D(S)$. Once we have gone around the tree T' and are almost back at q, we draw a line segment back to the original boundary of P, close to p; in other words we have cut a tracing of T' out of P, while keeping inside the polygon all other points of S. For each vertex v of T', polygon P has $deg(v)$ corners close to v, where $deg(v)$ denotes the degree of v in T'. At the corner of P closest to a leaf of T' there is an angle almost equal to 2π (i.e., this vertex is a reflex corner of P). At the corners of P close to an internal vertex of T' there is at most one angle larger than π. Near the points p and q the polygon P has angles less than π. Therefore, P has at most one reflex corner for each vertex of T', and hence it has at most k reflex corners.

We now place k dummy points on the boundary of P close to the k vertices of T'. Namely, for each vertex v of T' we place a dummy point p_v close to v on the boundary of P. For a vertex v of T', let T_v denote the subtree of T rooted at v and consisting only of the edges of T that do not belong to T'. Using Lemma 4,

we construct each subtree T_v so that its root is placed on p_v instead of v. From Lemma 4 we know that this drawing has at most $2\lceil k/2 \rceil$ bends per edge, and can be computed in $O(n^2 \log n)$ time. Then, for each v we connect p_v to v with a straight-line segment and remove the dummy point p_v, thus creating one more bend per edge. The theorem follows. □

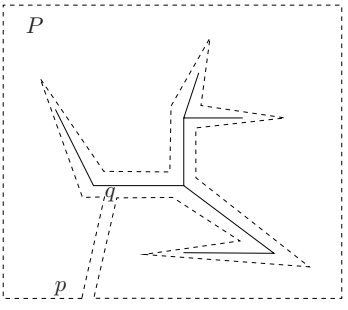


Fig. 7. Cutting a tree out of a polygon P. The tree is represented by solid edges while P has dashed segments.

The next result is a consequence of the proof of Theorem 1. Indeed, in that proof T' is an arbitrarily chosen subtree of T among those isomorphic to $D(S)$.

Corollary 1. *Let T be a tree with n vertices, S a set of n points in general position, and T' any subtree of T with k vertices. If $\Gamma(T', S)$ is a geometric point-set embedding of T' on a subset of S, then T has a point-set embedding $\Gamma(T, S)$ on S such that $\Gamma(T', S) \subset \Gamma(T, S)$ and every edge that does not belong to T' has at most $1+2\lceil k/2 \rceil$ bends. Also, $\Gamma(T, S)$ can be computed in $O(n^2 \log n)$ time.*

4 Conclusions and Open Problems

This paper introduced the problem of computing a point-set embedding of a graph G on a set S of points, with the constraint that a partial straight-line planar drawing of G on a subset of S is given. We concentrated on trees, and presented lower and upper bounds to the maximum number of bends per edge. We showed a lower bound equal to $k-3$ and an upper bound equal to $1+2\lceil k/2 \rceil$, where k is the number of vertices of the partial drawing. The upper bound is proved by means of an $O(n^2 \log n)$-time drawing algorithm. The drawing technique exploits a partial solution of a well-investigated and still unsolved computational geometry problem.

We mention in the following three open problems related to the results of this paper and that could be the subject of further investigation: (*i*) Extend the study to families of graphs other than trees. (*ii*) Compute constrained point-set embeddings with the minimum number of bends. (*iii*) Study the constrained

point-set embeddability problem in the case that the partial drawing to be extended contains some bends along its edges.

References

1. Badent, M., Di Giacomo, E., Liotta, G.: Drawing colored graphs on colored points. In: WADS 2007. LNCS, Springer, Heidelberg (2007)
2. Bose, P.: On embedding an outer-planar graph on a point set. Computational Geometry: Theory and Applications 23, 303–312 (2002)
3. Bose, P., McAllister, M., Snoeyink, J.: Optimal algorithms to embed trees in a point set. Journal of Graph Algorithms and Applications 2(1), 1–15 (1997)
4. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing. Prentice-Hall, Upper Saddle River (1999)
5. Di Giacomo, E., Didimo, W., Liotta, G., Meijer, H., Trotta, F., Wismath, S.K.: k-colored point-set embeddability of outerplanar graphs. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 318–329. Springer, Heidelberg (2007)
6. Di Giacomo, E., Liotta, G., Trotta, F.: On embedding a graph on two sets of points. International Journal of Foundations of Computer Science, Special Issue on Graph Drawing 17(5), 1071–1094 (2006)
7. Halton, J.H.: On the thickness of graphs of given degree. Information Sciences 54, 219–238 (1991)
8. Ikebe, Y., Perles, M., Tamura, A., Tokunaga, S.: The rooted tree embedding problem into points in the plane. Discrete Comput. Geometry 11, 51–63 (1994)
9. Kaneko, A., Kano, M.: Straight line embeddings of rooted star forests in the plane. Discrete Appl. Mathematics 101, 167–175 (2000)
10. Kaneko, A., Kano, M.: Semi-balanced partitions of two sets of points and embeddings of rooted forests. Int. J. Comput. Geom. Appl. 15(3), 229–238 (2005)
11. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
12. Kaufmann, M., Wiese, R.: Embedding vertices at points: Few bends suffice for planar graphs. Journal of Graph Algorithms and Applications 6(1), 115–129 (2002)
13. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. Lecture Notes Series on Computing, vol. 12. World Scientific, Singapore (2004)
14. O'Rourke, J.: Art Gallery Theorems and Algorithms. Oxford Univ. Press, Oxford (1987)
15. Pach, J., Wenger, R.: Embedding planar graphs at fixed vertex locations. Graphs and Combinatorics 17, 717–728 (2001)
16. Patrignani, M.: On extending a partial straight-line drawing. International Journal of Foundations of Computer Science, Special Issue on Graph Drawing 17(5), 1061–1069 (2006)
17. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction, 3rd edn. Springer, Heidelberg (1990)

Representation of Planar Hypergraphs by Contacts of Triangles

Hubert de Fraysseix, Patrice Ossona de Mendez, and Pierre Rosenstiehl

CAMS UMR 8557, CNRS/EHESS, Paris, France
{hf,pom,pr}@ehess.fr

Abstract. Many representation theorems extend from planar graphs to planar hypergraphs. The authors proved in [10] that every planar graph has a representation by contact of triangles. We prove here that this representation result extend to planar linear hypergraphs. Although the graph proof was simple and led to a linear time drawing algorithm, the extension for hypergraphs needs more work. The proof we give here relies on a combinatorial characterization of those hypergraphs which are representable by contact of segments in the plane, We propose some possible generalization directions and open problems, related to the order dimension of the incidence posets of hypergraphs.

1 Introduction

1.1 Hypergraphs

A *hypergraph* $\mathcal{H}$ is an ordered pair $(X, \mathcal{E})$ where X is a finite set whose elements are called *vertices* and $\mathcal{E}$ is a collection of nonempty subsets E of X called *edges* whose union is X. Two vertices $x, y \in X$ (resp. two edges $E, E' \in \mathcal{E}$) are *adjacent* if $\{x, y\}$ is included in some edge of $\mathcal{H}$ (resp. if $E \cap E'$ is not empty). A hypergraph is *linear* if any two edges have at most one common element: $\forall E, E' \in \mathcal{E} : |E \cap E'| \leq 1$. A *loop* of a hypergraph is an edge of cardinality 1. A loopless linear hypergraph is said to be *simple*. We note $n(\mathcal{H}) = |X|$ the *order* of $\mathcal{H}$ and $m(\mathcal{H}) = |\mathcal{E}|$ its *size*. The *sub-hypergraph* of $\mathcal{H}$ induced by a subset $A \subseteq X$ is the hypergraph $\mathcal{H}_A = (A, \mathcal{E}_A)$, where $\mathcal{E}_A = \{E \cap A : \quad E \in \mathcal{E}; E \cap A \neq \emptyset\}$. The *representative graph* $\mathrm{B}(\mathcal{H})$ of $\mathcal{H}$ is the black and white colored bipartite graph whose set of white vertices is X, whose set of black vertices is $\mathcal{E}$ and whose edges are those pairs $\{x, E\}$ such that $x \in E \in \mathcal{E}$. For terms in Hypergraph Theory, not specifically defined here, we refer the reader to [2].

Different generalizations of the concept of graph planarity to hypergraphs have been considered (See [13], for instance). Zykov proposed to represent the edges of a hypergraph by a subset of the faces of a planar map [28]. Walsh has shown that Zykov's definition (as well as another definition by Cori [4]) is equivalent to the following: A hypergraph is *planar* if and only if its representative graph $\mathrm{B}(\mathcal{H})$ is planar [26] (See also [5,14,27]). In the figures, hypergraphs are displayed by mean of their representative graph.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 125–136, 2007.

1.2 Geometrical Representations

Geometrical representations of planar graphs gained much attention these last three decades, particularly since planar graphs have been proved to have a straight line representation on a linear size grid [11,12,23]. Other geometrical representations have been proposed, like rectilinear representations (a.k.a. visibility representations) [19,24], representations by contacts of convex sets (for instance, circles [1,15] or triangles [10]) and representations by contacts or intersections of Jordan arcs or of straight line segments [7,9,8,21].

Contacts systems of segments are studied in [9,8]. For instance, it is proved there that a graph is the contact graph[1] of a family of segments if and only if any subgraph induced by a subset of $p \geq 2$ vertices has at most $2p-3$ edges. In particular any triangle-free planar graph is the contact graph of a family of segments (and this result extends to "triangle-free" planar hypergraphs). Not every planar graph (thus not every planar linear hypergraph) is representable by contacts of segments. The characterization of those planar linear hypergraphs which are contact hypergraphs of segments and points is recalled and discussed in Section 2.

A possible relaxation stands in allowing intersections. In [20], Scheinerman asked whether any planar graph could be represented as the intersection graph of a family of segments. Chalopin, Gonçalves and Ochem recently claimed to have proved this conjecture. The authors have proposed the following strengthening of Scheinermann's conjecture.

Conjecture 1. Any planar linear hypergraph is the intersection hypergraph of a family of segments.

Another possible relaxation stands in considering triangles instead of straight line segments. Actually, it is well known that any planar graph is the contact graph of a family of convex sets in the plane. Different kinds of convex sets have been considered, like circular disks [1,15]. However, as no three circular disks can be in contact at a common point, this representation fails to extend to hypergraphs. In [10], it is proved that any planar graph can be represented as the contact graph of a family of triangles in the plane. The aim of this paper is to extend this later result to planar linear hypergraphs.

1.3 Further Definitions on Graphs

For a subset X of vertices of a plane graph G, we denote $\mathcal{N}(X)$ the union of X and the set of the neighbors of the vertices in X in G, by $G[X]$ the subgraph of G induced by X. The set of edges incident to a vertex in X and a vertex out of X is denoted by $\omega(X)$. When G is directed, $\omega^-(X)$ denotes the set of the edges of $\omega(X)$ which are oriented toward X.

We denote by Extr(X) the vertex set of the outer face of $G[X]$. The *Closure* of X is the union of all the subsets Y such that $\mathrm{Extr}(Y) = \mathrm{Extr}(X)$. Notice that

[1] To be precise, we only allow one-sided contacts, see Section 2 for more details.

$\mathrm{Clos}(X) = \mathrm{Clos}(Y)$ if and only if $\mathrm{Extr}(X) \subseteq Y \subseteq \mathrm{Clos}(X)$. The subset X is *closed* if $X = \mathrm{Clos}(X)$. The subset X of vertices is a *disk* of G if X is closed, has cardinality at least 3, and the outer face of $G[X]$ is a cycle.

2 Contacts of Segments or Pseudo-segments

A finite set of Jordan arcs is called a family of *pseudo-segments* if every pair of arcs in the set intersects in at most one point. In particular, any family of segments is a family of pseudo-segments. A *one-sided contact family of pseudo-segments and points* is a couple $(\mathcal{A}, P)$, where $\mathcal{A}$ is a family of pseudo-segments that may touch (only on one side at each contact point) but may not cross and whose union is connected, and where P is a set of points in the union of the pseudo-segments including all the extremities of the pseudo-segments. For the sake of simplicity we shall use the term of *contact system* instead of "one-sided contact family" in the remaining of the paper.

Each contact system $(\mathcal{A}, P)$ defines a connected bipartite plane graph $G = (V_\circ, V_\bullet, E)$, its *incidence graph*, where $V_\circ$ corresponds to the pseudo-segment set, $V_\bullet$ corresponds to the point set and E corresponds to the set of incidences between points and pseudo-segments.

The graph G has no cycle of length 4 (as two pseudo-segments share at most one point) thus has girth at least 6. The planar linear hypergraph $\mathcal{H}$ whose representative graph is G is the *contact hypergraph* of $(\mathcal{A}, P)$. When no three pseudo-segments of $\mathcal{A}$ touch at a single point, the contact hypergraph actually is a graph, which we shall call the *contact graph* of $(\mathcal{A}, P)$. Contact hypergraphs of pseudo-segments have been characterized in [7] and contact hypergraphs of segments have been characterized in [8] and [9].

Moreover, the contact system also defines an orientation of G: if $x \in V_\bullet$ corresponds to a point p on a pseudo-segment S corresponding to $y \in V_\circ$, the edge $\{x, y\}$ is oriented from x to y if p is an extremity of S and from y to x, otherwise. This orientation is such that the indegree of a vertex in $V_\circ$ is exactly 2 and the indegree of a vertex in $V_\bullet$ is at most 1. We call such an orientation a $(2, \leq 1)$-*orientation*. The converse is quite simple to prove (see [7]):

Theorem 1. *A directed bipartite plane graph $G = (V_\circ, V_\bullet, E)$ is the incidence graph of a contact system of pseudo-segments and points with the embedding and the orientation induced by the contact system if and only if G has girth at least 6, and the orientation of G is a $(2, \leq 1)$-orientation.* □

The characterization of incidence graphs of segments and points given in [8] (Theorem 38) can be stated in a similar way:

Theorem 2. *A directed bipartite plane graph $G = (V_\circ, V_\bullet, E)$ is the incidence graph of a contact system of segments and points with the embedding and the orientation induced by the contact system if and only if G has girth at least 6, the orientation of G is a $(2, \leq 1)$-orientation and each subset $A \subseteq V_\circ$ which has cardinality at least two is such that $G[\mathcal{N}(A)]$ has at least three sources on its outer face.* □

To consider disks instead of neighborhoods in the last condition, we introduce a new function. For a subset X of vertices, $\sigma(X)$ denotes be the sum of $\omega^-(\mathrm{Clos}(X))$ and the number of sources of G in $\mathrm{Extr}(X)$.

Lemma 1. *Let $G = (V_\circ, V_\bullet, E)$ be a directed bipartite plane graph with girth at least 6, whose orientation is a $(2, \leq 1)$-orientation.*

Then G is the incidence graph of a contact system of segments and points with the embedding and the orientation induced by the contact system if and only if $\sigma(D) \geq 3$ for every disk D of G.

Proof. Assume G is the incidence graph of a contact system of segments and points with the embedding and the orientation induced by the contact system and let D be a disk of G. It is easily checked that $\sigma(D)$ is the number of sources of $G[\mathcal{N}(D \cap V_\circ)]$ lying on its outer face. Thus $\sigma(D) \geq 3$ according to Theorem 2.

Conversely, assume $\sigma(D) \geq 3$ for every disk D of G and let $A \subseteq V_\circ$ be a subset of a least two white vertices of G. It is easily checked that the number of sources of $G[\mathcal{N}(A)]$ lying on its outer face is equal to $\sigma(\mathcal{N}(A))$. As $\sigma(X) = \sigma(\mathrm{Clos}(X))$ and as G has girth at least 6, the inequality $\sigma(\mathcal{N}(A)) \geq 3$ will follow from the statement that $\sigma(X) \geq 3$ for every closed subset X such that $\left|\mathrm{Extr}(X) \cap V_\circ\right| \geq 2$, that we shall prove by induction on $\left|\mathrm{Extr}(X) \cap V_\circ\right|$.

If $\left|\mathrm{Extr}(X) \cap V_\circ\right| = 2$ then $G[X]$ includes no cycle, thus $\left|\mathrm{Extr}(X) \cap V_\circ\right| = 2$ and $\sigma(X) \geq 3$, by an easy case analysis. Assume $\sigma(X) \geq 3$ for every closed subset X such that $2 \leq \left|\mathrm{Extr}(X) \cap V_\circ\right| \leq k$ and let X be a closed subset with $(k+1) \geq 3$ white vertices. If $G[X]$ is disconnected, then either one connected component $G[X']$ of $G[X]$ has at least two white vertices on its outer face and the result follows from $\sigma(X) \geq \sigma(X')$ and the induction, or G has at least one connected component $G[X']$ with exactly one white vertex on its outer face, and the result follows from $\sigma(X) \geq \sigma(X \setminus X')$ and the induction. If $G[X]$ has a vertex v of degree 1 on its outer face, it is easily checked that $\sigma(X) \geq \sigma(X - v)$ thus $\sigma(X) \geq 3$ according to the induction. Otherwise, if the outer face of $G[X]$ is a cycle, then X is a disk and $\sigma(X) \geq 3$ by assumption. Otherwise, there exists closed subsets X_1, X_2 such that $|X_1 \cap X_2| = 1$ and $X = X_1 \cup X_2$ and each of $G[X_1]$ and $G[X_2]$ includes at least a cycle (thus includes at least two white vertices on its outer face). By induction, $\sigma(X_1) \geq 3$ and $\sigma(X_2) \geq 3$. As it is easily check that $\sigma(X) \geq \sigma(X_1) + \sigma(X_2) - 2$, we get $\sigma(X) \geq 4$.

3 Contacts of Triangles and Segments

Main Theorem. *Any planar linear hypergraph is the contact hypergraph of a family of triangles and segments.*

We shall first state some preliminary lemmas and explicit a few transformations on bipartite plane graphs. First recall the following orientation lemma, whose proof has been included for the sake of completeness.

Lemma 2 ([16]). *Let G be a multigraph, let λ be a mapping from $V(G)$ to $\mathbb{N}$. Then there exists an orientation of G such that each vertex $v \in V(G)$ has indegree bounded by $\lambda(v)$ if and only if*

$$\forall A \subseteq V(G): \quad \left| E(G[A]) \right| \leq \sum_{v \in A} \lambda(v). \tag{1}$$

Moreover, this orientation is such that each vertex v has indegree $\lambda(v)$ if and only if we also have the global condition $\left| E(G) \right| = \sum_{v \in V(G)} \lambda(v)$.

Proof. If every vertex v has indegree bounded by $\lambda(v)$ then the number of edges of $G[A]$ is bounded by $\sum_{v \in A} \lambda(v)$ for any subset A of vertices.

Conversely, assume (1) holds. To any orientation $\mathcal{O}$ associates the non negative integer value $f(\mathcal{O}) = \sum_{v:\ \mathrm{d}^-(v) > \lambda(v)} (\mathrm{d}^-(v) - \lambda(v))$. Let $\mathcal{O}$ be an orientation of G such that $f(\mathcal{O})$ is minimal. If $f(\mathcal{O}) = 0$ then any vertex v has indegree bounded by $\lambda(v)$. Assume $f(\mathcal{O}) > 0$. Let x_0 be a vertex such that $\mathrm{d}^-(x_0) > \lambda(x_0)$ and let $I(x_0)$ be the set of the vertices $v \in V(G)$ such that there exists a directed path (with respect to $\mathcal{O}$) from v to x_0. As no vertex in $I(x_0)$ has an arc coming from the outside of $I(x_0)$, we get $\left| E(G[I(x_0)]) \right| = \sum_{v \in I(x_0)} \mathrm{d}^-(v)$. As $\left| E(G[I(x_0)]) \right| \leq \sum_{v \in I(x_0)} \lambda(v)$ (according to (1)) and as $I(x_0)$ includes at least a vertex v such that $\mathrm{d}^-(v) > \lambda(v)$ (namely x_0) we deduce that $I(x_0)$ also includes a vertex y such that $\mathrm{d}^-(y) < \lambda(y)$. By construction there exists a directed path from y to x_0. By reversing the orientation of the edges of this directed path we get a new orientation $\mathcal{O}'$ for which all the indegrees but those of y and x_0 remain the same, the indegree of y increases by one and the indegree of x_0 decreases by one. Thus $f(\mathcal{O}') < f(\mathcal{O})$, a contradiction. □

Given a 2-connected bipartite plane graph $G = (V_\circ, V_\bullet, E(G))$ with girth 6 whose faces have length 6, we define:

- the bipartite plane graph $G^+ = (V_\circ \cup \{r\}, V_\bullet, E(G^+))$ obtained from G by adding a vertex r in the outer face linked to the black vertices of this face,
- the bipartite plane multigraph $G^+_{\|} = (V_\circ \cup \{r\}, V_\bullet, E(G^+_{\|}))$ obtained from G^+ by doubling every edge.

Lemma 3. *The graph $G^+_{\|}$ has an orientation $\mathcal{O}^+_{\|}$ such that every vertex in $V_\circ \cup V_\bullet$ has indegree 3 and r is a source.*

Proof. For every $A \subseteq V_\circ \cup V_\bullet$ we have $2\left| E(G^+)[A] \right| \leq 3|A| - 6$, according to Euler's formula. Hence $\left| E(G^+_{\|})[A] \right| \leq 3|A| - 6$. Let λ is the mapping from $V(G^+_{\|})$ to $\mathbb{N}$ defined by $\lambda(v) = 0$ if $v = r$, and $\lambda(v) = 3$ otherwise. Then, for any $A \subseteq V(G^+_{\|})$, we have $\left| E(G^+_{\|})[A] \right| \leq \sum_{v \in A} \lambda(v)$. As $\left| E(G^+_{\|}) \right| = \sum_{v \in V(G^+_{\|})} \lambda(v)$, it follows from Lemma 2 that $G^+_{\|}$ has an orientation $\mathcal{O}^+_{\|}$ such that $d^-(v) = \lambda(v)$ for every vertex. □

Given such an orientation $\mathcal{O}^+_{\|}$ of $G^+_{\|}$, we define:

- $\Upsilon(\mathcal{O}^+_{\|})$ is the set of the edges $\{x, y\}$ of G^+ such that both (x, y) and (y, x) are arcs of $G^+_{\|}$,

- $\Omega(\mathcal{O}_{\|}^{+})$ is the orientation of G^{+} such that an edge $\{x, y\}$ is oriented from x to y if either both arcs linking x and y in $G_{\|}^{+}$ are oriented from x to y with respect to $\mathcal{O}_{\|}^{+}$, or $\{x, y\} \in \Upsilon(\mathcal{O}_{\|}^{+})$ and y is white.
- the *type* of a vertex v as 1 if either v has two incoming edges coming from one of its neighbors, or as 2 if the three incoming edges of v come from different neighbors of v.

Fact 1. *Assume every vertex in $V_{\circ} \cup V_{\bullet}$ is of type 1 in $G_{\|}^{+}$ with respect to $\mathcal{O}_{\|}^{+}$. Then:*

- *The vertex r is a source of G^{+} with respect to $\Omega(\mathcal{O}_{\|}^{+})$,*
- *the set $\Upsilon(\mathcal{O}_{\|}^{+})$ is a perfect matching of G,*
- *each vertex $v \in V_{\bullet}$ has exactly one incoming edge in G^{+}, and this edge does not belong to M,*
- *each vertex $v \in V_{\circ}$ has exactly two incoming edge in G^{+}, exactly one of which belongs to M.*

Proof. The vertex r is a source of $\Omega(\mathcal{O}_{\|}^{+})$ by construction, as it is a source of $\mathcal{O}_{\|}^{+}$. Also, no edge incident to r in G^{+} belongs to $\Upsilon(\mathcal{O}_{\|}^{+})$. Every vertex different from r has indegree 1 in $G^{+} \setminus \Upsilon(\mathcal{O}_{\|}^{+})$ with respect to $\Omega(\mathcal{O}_{\|}^{+})$ as it is of type 1 in $G_{\|}^{+}$. The set $\Upsilon(\mathcal{O}_{\|}^{+})$ is a perfect matching of G as it is obviously a 1-factor (each vertex of $G = G^{+} - r$ has degree 1 in $\Upsilon(\mathcal{O}_{\|}^{+})$). The last two items also follows directly from the definition of $\Omega(\mathcal{O}_{\|}^{+})$. □

Lemma 4. *Let $\mathcal{O}_{\|}^{+}$ be an orientation of $G_{\|}^{+}$ such that r is a source and every vertex in $V_{\circ} \cup V_{\bullet}$ has indegree 3 and type 1. Consider the orientation $\Omega(\mathcal{O}_{\|}^{+})$ of G^{+}. Each disk D of G is such that $\big|\,\omega^{-}(D)\,\big| \geq 3$.*

Proof. Let $M = \Upsilon(\mathcal{O}_{\|}^{+})$, let γ be the outer face of $G^{+}[D]$ and $2l$ its length, let $\omega_{\circ}$ (resp. $\omega_{\bullet}$) be the subset of $\omega(D)$ formed by the edges having an endpoint in $D \cap V_{\circ}$ (resp. $D \cap V_{\bullet}$).

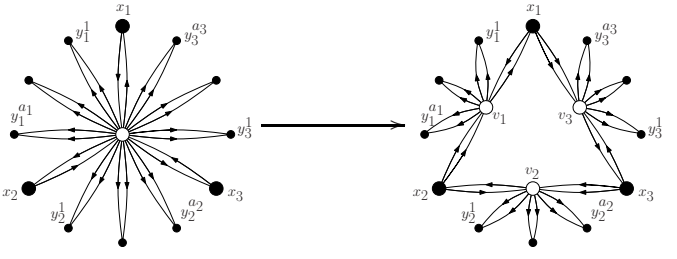
The summation of the indegrees of the vertices in the subgraph $G_{\|}^{+}[D]$ gives: $2\big|\,E(G^{+}[D])\,\big| = 3\big|\,D\,\big| - 2\big|\,\omega^{-}(D) \setminus M\,\big| - \big|\,\omega(D) \cap M\,\big|$. The value $2\big|\,E(G^{+}[D])\,\big|$ is also the sum of the length of the faces of H thus, as all the interior faces have length 6 and as the outer face as length $2l$, we get $2\big|E(G^{+}[D])\big| = 6(\big|\,E(G^{+}[D])\,\big| - \big|\,D\,\big| + 1) + 2l$, that is: $2\big|\,E(G^{+}[D])\,\big| = 3\big|\,D\,\big| - l - 3$. Hence $3\big|\,D\,\big| - 2\big|\,\omega^{-}(D) \setminus M\,\big| - \big|\,\omega(D) \cap M\,\big| = 3\big|\,A\,\big| - l - 3$. As $\big|\,\omega^{-}(D)\,\big| = \big|\,\omega^{-}(D) \setminus M\,\big| + \big|\,\omega_{\circ} \cap M\,\big|$, we deduce

$$\big|\,\omega^{-}(D)\,\big| = l + 3 - \big(\big|\,\omega^{-}(D) \setminus M\,\big| + \big|\,\omega_{\bullet} \cap M\,\big|\big) \tag{2}$$

Let t (resp. z) denotes the number of white vertices of γ which are matched in γ and is a sink of γ (resp. not a sink of γ). Obviously $t + z = |\gamma \cap M|$. Every sink of γ has indegree 2 in γ hence is a white vertex of γ and is matched in γ. It follows that t is the number of sinks of γ hence also the number of sources of

γ. Let $e \in \omega^-(M)$. Then either e is incident to a source of γ, or it is incident to a white vertex which is matched in γ and which is not a sink of γ. It follows that $\left|\omega^-(D) \setminus M\right| \leq t + z = |\gamma \cap M|$. Also, $\left|\gamma \cap M\right| + \left|\omega_\bullet \cap M\right| \leq l$ as this is the number of black vertices of γ matched in $\omega_\bullet \cup \gamma$. Altogether, we get $\left|\omega^-(D) \cap M\right| + \left|\omega_\bullet \cap M\right| \leq \left|\gamma \cap M\right| + \left|\omega_\bullet \cap M\right| \leq l$. Thus $\left|\omega^-(D)\right| \geq 3$, according to (2). □

Definition 1. *Le v be a vertex of type 2, and let $x_1, y_1^1, \ldots, y_1^{a_1}$, $x_2, y_2^1, \ldots, y_2^{a_2}$, $x_3, y_3^1, \ldots, y_3^{a_3}$ be the neighbors of v in circular order, where x_1, x_2, x_3 are the three neighbors of v incident to an arc oriented to v. The* splitting *of v is obtained by replacing v by three vertices v_1, v_2, v_3 and dispatching the arcs incident to v to v_1, v_2, v_3: for $i \in \{1, 2, 3\}$, the arcs incident to v_i are: one arc to x_i, one arc from x_i, for each $j = 1, \ldots, a_1$ two arcs to y_i^j, and two arcs coming from x_{i+1} (if $i < 3$, or x_1 if $i = 3$).*



Remark that the splitting of a vertex v of type 2 into v_1, v_2, v_3 preserves the length of the faces and the indegrees. After the splitting, each of v_1, v_2, v_3 is of type 1.

Lemma 5. *Let $\mathcal{O}_\parallel^+$ be an orientation of $G_\parallel^+$ such that r is a source and every vertex in $V_\circ \cup V_\bullet$ has indegree 3 and such that every vertex in $V_\circ$ has type 1. Consider the orientation $\Omega(\mathcal{O}_\parallel^+)$ of G^+. Each disk D of G is such that $\sigma(D) \geq 3$.*

Proof. We proceed by induction on the number of vertices in $V_\bullet$ which have type 2. If no vertex in $V_\bullet$ has type 2, the result follows from Lemma 4. Assume that the result holds if $V_\bullet$ includes at most k vertices of type 2 ($k \geq 0$) and assume $V_\bullet$ includes $k+1$ vertices of type 2. Let v be one of them. Let G' be the multigraph obtained from $G_\parallel^+$ by splitting v into three vertices v_1, v_2, v_3. Let H be the unique plane graph such that $H_\parallel^+ = G'$. Let F be the circuit of length 6 including v_1, v_2, v_3 arising in H^+ from the splitting of v. Let D be a disk of G. If $v \notin \mathrm{Extr}(D)$ then the result follows from the induction applied to the graph $H_\parallel^+$ obtained from $G_\parallel^+$ by splitting v (the σ-value of D in G^+ will be the same has the σ-value of $\mathrm{Clos}(\mathrm{Extr}(D))$ in H^+). Otherwise, let D' be the disk of H^+ obtained from $D - v$ by adding those of v_1, v_2, v_3 having at least two neighbors in $D - v$. Then the set of edges $\omega^-(D')$ (in H^+) is the union of the set $\omega^-(D)$ (computed in G^+) and of the set of edges in $\omega^-(D') \cap F$. As the outer face of $H^+[D']$ meets F on an interval (possibly reduced to a vertex) and as F is a circuit, $|\omega^-(D') \cap F| \leq 1$. As v was a source of G^+ and none of v_1, v_2, v_3 are, we

get that the σ-value of D in G^+ is at least equal to the σ-value of D' in H^+, which in turn is at least 3 by induction. Hence $\sigma(D) \geq 3$, what completes the proof of the induction. □

Lemma 6. *Let $\mathcal{O}^+_{\|}$ be an orientation of $G^+_{\|}$ such that r is a source and every vertex in $V_\circ \cup V_\bullet$ has indegree 3 and such that every vertex in $V_\circ$ has type 1. Consider the orientation $\Omega(\mathcal{O}^+_{\|})$ of G^+ and its restriction to G. Then G is the incidence graph of a contact system of segments and points with the embedding and the orientation induced by the contact system.*

Proof. This is a direct consequence of Lemma 5 and Lemma 1. □

Consider a contact family $\mathcal{F}$ of triangles and segments. Each triangle is formed by three segments. The contact family thus defines a bipartite incidence graph of segments and points, in which the segments of $\mathcal{F}$ appear as vertices, and the triangles of $\mathcal{F}$ appear as faces of length 6.

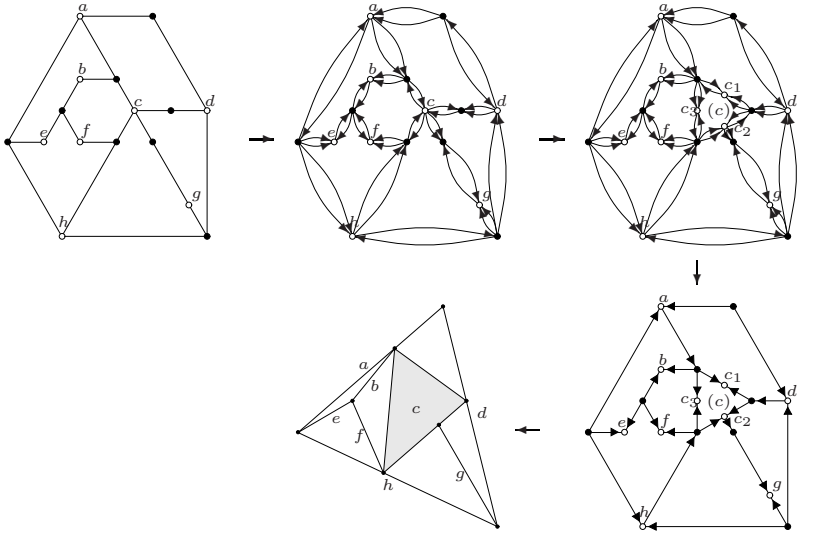


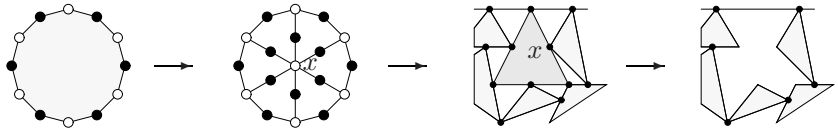
Fig. 1. Representation of a planar linear hypergraph by contacts of triangles and segments. First figure is the representative graph B of the hypergraph, second figure is the orientation of $B^+_{\|} - r$, third figure is the orientation of $\Gamma^+_{\|} - r$ obtained from $B^+_{\|} - r$ by splitting of vertices of type 2 (here c is split into c_1, c_2, c_3), fourth figure is the associated orientation of Γ. The last figure is the deduced representation of the hypergraph.

Conversely, to represent planar linear hypergraph as the incidence graph of a family of triangles and segments in contacts, we will derive from the representative plane graph B of the hypergraph a bipartite plane graph Γ, so that each vertex of B will correspond either to a vertex or a face of length 6 of Γ,

and deduce the representation of B from a representation of Γ as the incidence graph of a family of segments and points (see Fig. 1 and 2).

Although the exact details of the construction of an actual family of segments and points whose incidence graph is Γ can be found in [8], we shall give some intuition of how it works: By using augmentations one reduces to the case where Γ have exactly 3 sources lying on the outer face. Then these vertices are embedded into 3 points of the plane in general position forming a triangle T_0. The orientation and plane embedding of the bipartite graph are translated into a linear system which resolution gives the embedding of the remaining black vertices in the interior of T_0. In this embedding, the white vertices correspond to the straight line segments which endpoints will be the points corresponding to the 2 incoming black vertices. This construction is described in [8]. A hint of these two steps is given in the last two drawings of Fig. 1. According to the construction of Γ, a representation of the hypergraph $\mathcal{H}$ by contacts of triangles and segments follows.

Proof of the Main Theorem. Let $\mathcal{H}$ be a planar linear hypergraph. We may assume without loss of generality that $\mathcal{H}$ has order and size at least 3. We may also assume that the representative plane graph $B = (V_\circ, V_\bullet, E(B))$ of $\mathcal{H}$ is 2-connected and that every face of B have length 6, as this may be achieved by adding some dummy vertices to $\mathcal{H}$ without creating any additional adjacencies between the original vertices of $\mathcal{H}$. A representation of $\mathcal{H}$ by contact of triangles is then deduced from a representation of the augmented hypergraph by deleting the triangles corresponding to the dummy vertices:



According to Lemma 3, $B^+_{\parallel}$ has an orientation such that every vertex in $V_\circ \cup V_\bullet$ has indegree 3 and r is a source. By splitting all the vertices in $V_\circ$ having type 2, we get a graph $\Gamma^+_{\parallel}$ (associated with a bipartite plane graph Γ) and an associated orientation of Γ^+, so that each vertex of B corresponds either to a vertex or a face of length 6 of Γ whose edges are oriented from the black vertices. According to Lemma 6, the graph Γ is the incidence graph of a contact system of segments and points with the embedding and the orientation induced by the contact system. As the faces of length 6 coming from splittings are then empty triangles, we deduce a representation of $\mathcal{H}$. □

4 Extensions and Open Problems

The *incidence poset* of a graph (or more generally of a hypergraph) is the poset where the only covers are defined by $x < e$ if x is a vertex, e is an edge and e is incident to x. The *dimension* $\dim \mathbf{P}$ of $\mathbf{P} = (X, P)$ is the least positive integer

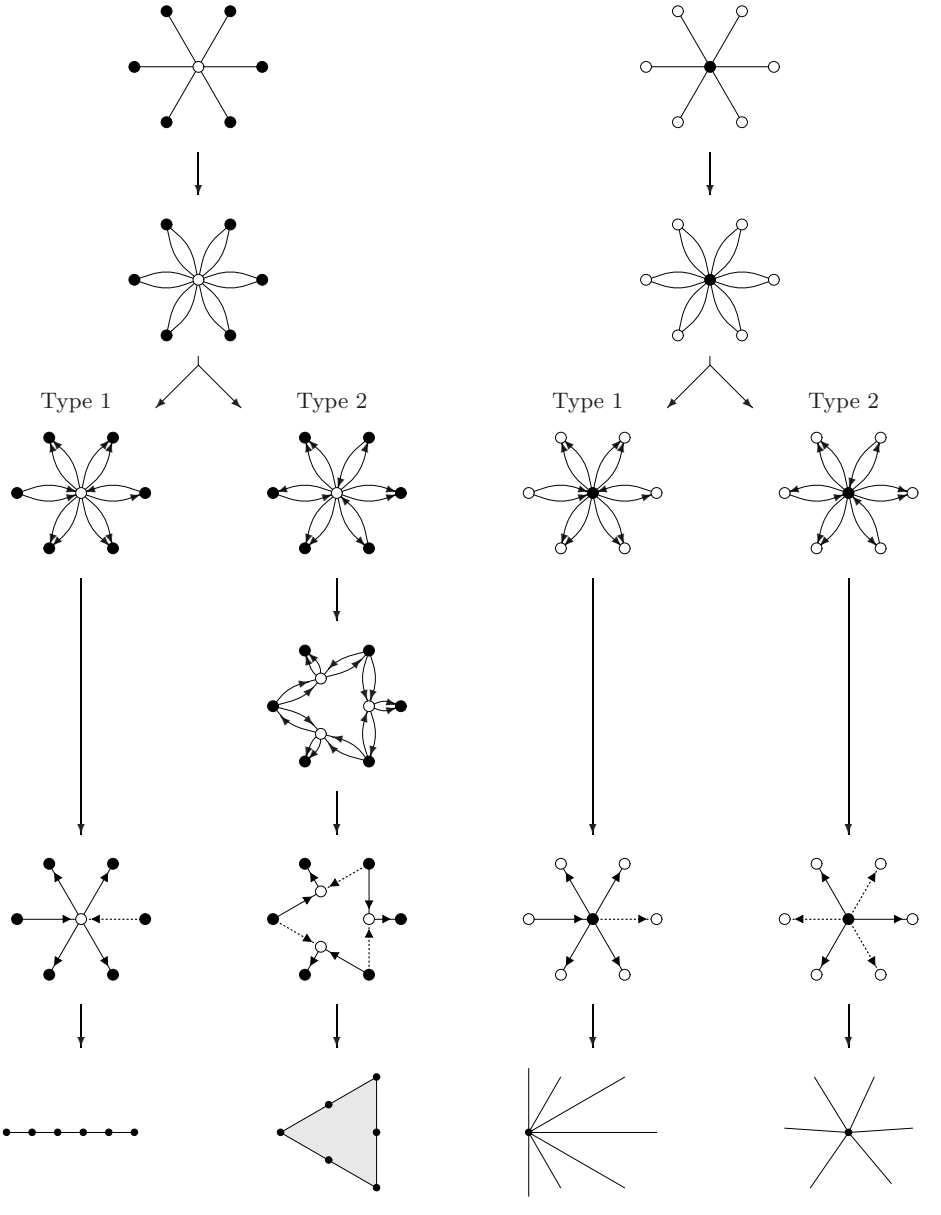


Fig. 2. From the representative graph to the contact representation. Neighborhoods of vertices are shown in B (first row), in $B_{\|}^{+}$ (second row), in $B_{\|}^{+}$ with orientation $\mathcal{O}_{\|}^{+}$ (third row), in $\Gamma_{\|}^{+}$ with orientation $\mathcal{O}_{\|}^{+}$ (fourth row), in Γ^{+} with orientation $\mathcal{O}^{+}$ where dashed edges representing M (fifth row), in the contact representation (sixth row).

t for which there exists a family $\mathcal{R} = (<_1, <_2, \ldots, <_t)$ of linear extensions of P so that $P = \bigcap \mathcal{R} = \bigcap_{i=1}^{t} <_i$. This concept has been introduced by Dushnik and Miller in [6]. A family $\mathcal{R} = (<_1, <_2, \ldots, <_t)$ of linear orders on X is called a *realizer* of P on X if $P = \bigcap \mathcal{R}$. For an extended study of partially ordered sets, we refer the reader to [25].

A celebrated theorem of Schnyder states that a graph is planar if and only if its incidence poset has dimension at most 3 [22]. Although the incidence poset of a simple planar hypergraph $\mathcal{H}$ has dimension at most 3 (what follows from [3]), the converse is false: The linear hypergraph $\mathcal{H}$ with vertices $1, \ldots, 5$ and edge set $\{\{1,2\},\{2,3\},\{3,4\},\{1,4\},\{1,3,5\},\{2,4,5\}\}$. is not planar ($\mathrm{B}(\mathcal{H})$ is a subdivision of $K_{3,3}$) though its incidence poset has a realizer[2] of size 3.

Schnyder's theorem is generalized in [17] to a sufficient condition for the geometric realizability of an abstract simplicial complex and in [18] to a general representation theorem for posets. From this later theorem, it is easily deduced that the vertices of simple hypergraphs with incidence posets of dimensions d can be represented by convex sets of the Euclidean space of dimension $d-1$, in such a way that the edges of the hypergraph are exactly the maximal subsets of vertices, such that the corresponding subset of convexes has a non-empty intersection. Thus, in some way, hypergraphs with incidence poset of dimension 3, although not necessarily planar, still have a strong relation with the plane. This encourages the following conjecture:

Conjecture 2. Any linear hypergraph with incidence poset of dimension at most 3 is the intersection hypergraph of a family of triangles and segments in the plane.

References

1. Andreev, E.M.: On convex polyhedra in Lobačevskiĭ spaces. Matematicheskii Sbornik 81, 445–478 (1970)
2. Berge, C.: Graphes et hypergraphes, 2nd edn. Dunod, Paris (1973)
3. Brightwell, G., Trotter, W.T.: The order dimension of planar maps. SIAM journal on Discrete Mathematics 10(4), 515–528 (1997)
4. Cori, R.: Un code pour les graphes planaires et ses applications, Société Mathématique de France, Paris, vol. 27 (1975)
5. Cori, R., Machì, A.: Maps, hypermaps and their automorphisms. Expo. Math. 10, 403–467 (1992)
6. Dushnik, B., Miller, E.W.: Partially ordered sets. Amer. J. Math. 63, 600–610 (1941)
7. de Fraysseix, H., Ossona de Mendez, P.: Intersection Graphs of Jordan Arcs, Contemporary Trends in Discrete Mathematics. In: DIMATIA-DIMACS. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Štiřin 1997 Proc., pp. 11–28 (1999)
8. de Fraysseix, H., Ossona de Mendez, P.: Barycentric systems and stretchability. Discrete Applied Mathematics 155(9), 1079–1095 (2007)

[2] Take
$<_1$: $2 < 1 < \{1,2\} < 3 < \{2,3\} < 5 < \{1,3,5\} < 4 < \{2,4,5\} < \{1,4\} < \{3,4\}$
$<_2$: $5 < 4 < 2 < \{2,4,5\} < 3 < \{3,4\} < \{2,3\} < 1 < \{1,3,5\} < \{1,2\} < \{1,4\}$.
$<_3$: $1 < 4 < \{1,4\} < 3 < \{3,4\} < 5 < \{1,3,5\} < 2 < \{2,4,5\} < \{1,2\} < \{2,3\}$

9. de Fraysseix, H., Ossona de Mendez, P.: On representations by contact and intersection of segments. Algorithmica 47(4), 453–463 (2007)
10. de Fraysseix, H., Ossona de Mendez, P., Rosenstiehl, P.: On triangle contact graphs. Combinatorics, Probability and Computing 3, 233–246 (1994)
11. de Fraysseix, H., Pach, J., Pollack, R.: Small sets supporting Fary embeddings of planar graphs. In: 20th Annual ACM Symposium on Theory of Computing, pp. 426–433 (1988)
12. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10, 41–51 (1990)
13. Johnson, D.S., Pollak, H.O.: Hypergraph planarity and the complexity of drawing Venn diagrams. Journal of Graph Theory 11(3), 309–325 (1987)
14. Jones, R.P.: Colourings of hypergraphs, Ph.D. thesis, Royal Holloway College, Egham, p. 209 (1976)
15. Koebe, P.: Kontaktprobleme der konformen Abbildung. Ber. Verh. Schs. Akad. Wiss. Leipzig, Math.-Phys. Kl. 88, 141–164 (1936)
16. Ossona de Mendez, P.: Orientations bipolaires, Ph.D. thesis, Ecole des Hautes Etudes en Sciences Sociales, Paris (1994)
17. Ossona de Mendez, P.: Geometric Realization of Simplicial Complexes, Graph Drawing. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 323–332. Springer, Heidelberg (1999)
18. Ossona de Mendez, P.: Realization of posets. Journal of Graph Algorithms and Applications 6(1), 149–153 (2002)
19. Rosenstiehl, P., Tarjan, R.E.: Rectilinear planar layout and bipolar orientation of planar graphs. Discrete and Computational Geometry 1, 343–353 (1986)
20. Scheinerman, E.R.: Intersection classes and multiple intersection parameters of graphs, Ph.D. thesis, Princeton University (1984)
21. Scheinerman, E.R., West, D.B.: The interval number of a planar graph: Three intervals suffice. Journal of Combinatorial Theory, Series B 35, 224–239 (1983)
22. Schnyder, W.: Planar graphs and poset dimension. Order 5, 323–343 (1989)
23. Schnyder, W.: Embedding planar graphs in the grid. In: First ACM-SIAM Symposium on Discrete Algorithms, pp. 138–147 (1990)
24. Tamassia, R., Tollis, I.G.: Tessalation representation of planar graphs. In: Proc. Twenty-Seventh Annual Allerton Conference on Communication, Control, and Computing, pp. 48–57 (1989)
25. Trotter, W.T.: Combinatorics and partially ordered sets: Dimension theory. John Hopkins series in the mathematical sciences. Johns Hopkins University Press, London (1992)
26. Walsh, T.R.S.: Hypermaps versus bipartite maps. J. Combinatorial Theory 18(B), 155–163 (1975)
27. White, A.T.: Graphs, Groups and Surfaces, revised edn. Mathematics Studies, vol. 8. North-Holland, Amsterdam (1984)
28. Zykov, A.A.: Hypergraphs. Uspeki Mat. Nauk 6, 89–154 (1974)

The Complexity of Several Realizability Problems for Abstract Topological Graphs
(Extended Abstract)

Jan Kynčl

Department of Applied Mathematics and Institute for Theoretical Computer Science, Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, 118 00 Prague, Czech Republic
kyncl@kam.mff.cuni.cz

Abstract. An *abstract topological graph* (briefly an *AT-graph*) is a pair $A = (G, R)$ where $G = (V, E)$ is a graph and $R \subseteq \binom{E}{2}$ is a set of pairs of its edges. An AT-graph A is *simply realizable* if G can be drawn in the plane in such a way that each pair of edges from R crosses exactly once and no other pair crosses. We present a polynomial algorithm which decides whether a given complete AT-graph is simply realizable. On the other hand, we show that other similar realizability problems for (complete) AT-graphs are NP-hard.

1 Introduction

A *topological graph* $T = (V(T), E(T))$ is a drawing of an (abstract) graph G in the plane with the following properties. The vertices of G are represented by a set $V(T)$ of distinct points in the plane and the edges of G are represented by a set $E(T)$ of simple curves connecting the corresponding pairs of points. We call the elements of $V(T)$ and $E(T)$ *vertices* and *edges* of T. The edges cannot pass through any vertices except their end-points. Any intersection point of two edges is either a common end-point or a *crossing*, a point where the two edges properly cross ("touching" of the edges is not allowed). We also require that any two edges have only finitely many intersection points and that no three edges pass through a single crossing. A topological graph is *simple* if every two edges have at most one common point (which is either a common end-point or a crossing). A topological graph is *complete* if it is a drawing of a complete graph.

An *abstract topological graph* (briefly an *AT-graph*), a notion established in [7], is a pair (G, R) where G is a graph and $R \subseteq \binom{E(G)}{2}$ is a set of pairs of its edges. For a topological graph T which is a drawing of G we define R_T as a set of pairs of edges having at least one common crossing and we say that (G, R_T) is an AT-graph of T. A topological graph T is called a *realization* of (G, R) if $R_T = R$. If $R_T \subseteq R$, then T is called a *weak realization* (or also a *feasible drawing*) of (G, R). If (G, R) has a (weak) realization, we say that (G, R) is *(weakly) realizable*. We say that (G, R) is *simply (weakly) realizable* if (G, R) has a simple (weak) realization, that is, a drawing which is a simple topological graph. We

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 137–158, 2007.

say that (G, R) is *weakly rectilinearly realizable* if it has a weak realization T with edges drawn as straight-line segments (such drawing T is called a *weak rectilinear realization* of (G, R)).

Complete topological graphs are one of the most studied classes of topological graphs [5,11,12,13,15], especially in connection to the crossing number problems [1,4,16,19,20].

We study the complexity of various realizability problems for AT-graphs and also for complete AT-graphs. For example, the *realizability* problem is defined as follows: the instance is an AT-graph A and the question is whether A is realizable. Similarly the *weak realizability*, the *simple realizability*, the *simple weak realizability* and the *weak rectilinear realizability* problems are defined.

Kratochvíl [9] proved that the realizability and the weak realizability are NP-hard problems (for the class of all AT-graphs). For a long time, the decidability of these problems was an open question. Pach and Tóth [14] and Schaefer and Štefankovič [18] independently found a first recursive algorithm for the recognition of string graphs, which is polynomially equivalent to the realizability [9] and the weak realizability [6]. Later Schaefer, Sedgwick and Štefankovič [17] showed that the recognition of string graphs and the weak realizability are in NP, which implies the following corollary.

Theorem 1. [9,17] *The weak realizability and the realizability of AT-graphs are NP-complete problems.*

We extend these results by finding the complexities of the other mentioned problems, for the class of all AT-graphs and also for the class of complete AT-graphs. All these results are summarized in the following table.

Theorem 2

	AT-graphs	complete AT-graphs
realizability	NP-complete [9,17]	NP-complete
weak realizability	NP-complete [9,17]	NP-complete
simple realizability	NP-complete	polynomial
simple weak realizability	NP-complete	NP-complete
weak rectilinear realizability	NP-hard	NP-hard

The weak realizability of AT-graphs is polynomially equivalent to the *simultaneous drawing* problem [3]. The instance of this problem is a graph G given as a union of planar graphs $G_1, G_2, \ldots, G_k$ sharing some common edges. The question is whether G can be drawn in the plane so that each of the subgraphs G_i is drawn without crossings. The simultaneous drawing of three planar graphs is known to be NP-complete [3]; this gives an alternative proof of the NP-completeness of the weak realizability. The complexity of simultaneous drawing of two planar graphs remains open.

The rest of this paper is devoted to the proof of Theorem 2.

2 Additional Definitions

A *face* of a topological graph T is a connected component of the set $\mathbb{R}^2 \setminus E(T)$. A *rotation* of a vertex $v \in V(T)$ is the clockwise cyclic order in which the edges incident with v leave the vertex v. A *rotation system* of the topological graph T is the set of rotations of all its vertices. Similarly we define a *rotation* of a crossing c as the clockwise order in which the four portions of the two edges crossing at c leave the point c (note that each crossing has exactly two possible rotations). An *extended rotation system* of a topological graph is the set of rotations of all its vertices and crossings.

Assuming that T and T' are drawings of the same abstract graph, we say that their (extended) rotation systems are *inverse* if for each vertex $v \in V(T)$ (and each crossing c in T) the rotation of v and the rotation of the corresponding vertex $v' \in V(T')$ are inverse cyclic permutations (and so are the rotations of c and the corresponding crossing c' in T'). For example, if T' is a mirror image of T, then T and T' have inverse (extended) rotation systems.

Topological graphs G and H are *weakly isomorphic* if there exists an incidence preserving one-to-one correspondence between $V(G), E(G)$ and $V(H), E(H)$ such that two edges of G cross if and only if the corresponding two edges of H do. In other words, two topological graphs are weakly isomorphic if and only if they are realizations of the same abstract topological graph.

Topological graphs G and H are *isomorphic* if (1) G and H are weakly isomorphic, (2) for each edge e of G the order of crossings with the other edges of G is the same as the order of crossings on the corresponding edge e' in H, and (3) the extended rotation systems of G and H are the same or inverse. This induces a one-to-one correspondence between the faces of G and H such that the crossings and the vertices incident with a face f of G appear along the boundary of f in the same (or inverse) cyclic order as the corresponding crossings and vertices in H appear along the boundary of the face f' corresponding to f.

Assuming that the topological graphs G and H are drawn on the sphere, it follows from Jordan-Schönflies theorem that G and H are isomorphic if and only if there exists a homeomorphism of the sphere which transforms G into H.

Unlike the isomorphism, the weak isomorphism can change the faces of the involved topological graphs, as well as the order in which one edge crosses other edges.

3 The NP-Hard Problems

In this extended abstract, we give only a sketch of the reduction for the NP-hard problems, the details are postponed to the Appendix.

Our proof is based on the Kratochvíl's [9] reduction from planar 3-connected 3-SAT (P3C3-SAT), which is known to be an NP-complete problem [10]. The question is the satisfiability of a CNF formula ϕ with a set of clauses C and a set of variables X, such that each clause consists of exactly 3 distinct variables and the bipartite graph $G_\phi = (C \cup X, \{cx; x \in X, c \in C, x \in c\})$ is planar and 3-connected.

The main idea is essentially the same as in Kratochvíl's proof [9]—given the formula ϕ, we construct an AT-graph A_ϕ, which consists of vertex and clause gadgets connected by *joining edges*. The only variation is that we use different clause and vertex gadgets for different problems.

The evaluation of each vertex gadget is encoded by one of the two possible orders of *joining vertices* (two for each neighbor in G_ϕ). These orders are translated by the pairs of joining edges onto the orders of joining vertices of clause gadgets. For each clause gadget there are, theoretically, eight possible orders of the joining vertices, but only those seven corresponding to the satisfying evaluation can occur in the drawing. An example of variable and clause gadgets for the simple realizability problem is in the Figure 1. The set R of pairs of edges in the corresponding AT-graph is precisely the set of crossing pairs of edges in the drawing.

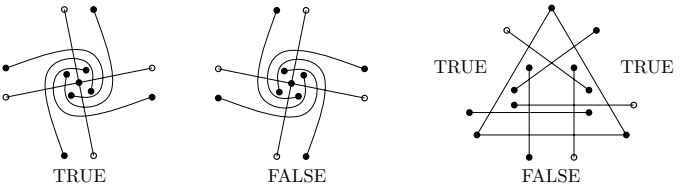


Fig. 1. Variable and clause gadgets for the simple realizability problem

4 Recognition of Simply Realizable Complete AT-Graphs

In this section we present a polynomial algorithm which decides whether a given complete AT-graph A is simply realizable. In the affirmative case, it also provides a description of the isomorphism class of one simple realization of A. For the sake of simplicity, we do not try to optimize the order of the polynomial bounding the computing time of the algorithm.

We need the following key observation.

Proposition 3. (1) *If two simple complete topological graphs are weakly isomorphic, then their extended rotation systems are either the same or inverse.*
(2) *For each edge e of a simple complete topological graph G and for each pair of edges $f, f' \in E(G)$ which have a common end-point and cross e, the AT-graph of G determines the order of crossings of e with the edges f, f'.*

The proof is postponed to the Appendix.

We will denote the rotation system of a topological graph G as $\mathcal{R}(G)$ and we will represent it as a sequence of rotations of its vertices. The rotation $\mathcal{R}(v)$ of a vertex v will be represented by a cyclic sequence of the labels of the remaining vertices.

Now we introduce a *star-cut representation* of the graph G. Choose an arbitrary vertex v and denote by $w_1, w_2, \ldots, w_{n-1}$ the remaining vertices of G so

that $\mathcal{R}(v) = (w_1, w_2, \ldots, w_{n-1})$. Let $S(v)$ denote the union of all the edges vw_i of G ($S(v)$ is a "topological star" with the central vertex v). If we consider G drawn on the sphere S^2, the set $S^2 \setminus S(v)$ is mapped by a homeomorphism Φ onto an open regular $2(n-1)$-gon D in the plane. We can visualize this by cutting the sphere along the edges of the star $S(v)$ and then unpacking the resulting surface in the plane. The map Φ^{-1} can be continuously extended to the closure of D, giving a natural correspondence between the vertices and edges of D and the vertices and edges in $S(v)$: each vertex w_i corresponds to one vertex w'_i of D and the vertex v of G corresponds to $n-1$ vertices $v'_1, v'_2, \ldots, v'_{n-1}$ of D. If Φ preserves the orientation, the counter-clockwise order of the vertices of D is $v'_1, w'_1, v'_2, w'_2, \ldots, v'_{n-1}, w'_{n-1}$. Each edge $vw_i \in E(G)$ splits into two adjacent edges $v'_i w'_i$ and $v'_{i+1} w'_i$; see Figure 2. During the cutting operation every edge e_k of G not incident with v can be cut into several pieces. Since e_k crosses each edge of $S(v)$ at most once, it is cut into at most n pieces $e_{k,j}$. Every crossing of the edge e_k with an edge vw_i corresponds to two end-points of two different pieces $e_{k,j}$,$e_{k,j'}$ lying on the edges $v'_i w'_i$ and $v'_{i+1} w'_i$.

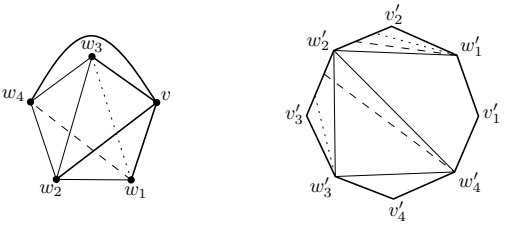


Fig. 2. A simple drawing of K_5 and its star-cut representation

The Algorithm

Suppose that we are given a complete AT-graph A with the vertex set $\{1, 2, \ldots, n\}$. The first step of the algorithm is the computation of the (abstract) rotation system $\mathcal{R}(A)$, i.e., the rotation system of a simple realization of A, if it exists:

- In order to $\mathcal{R}(A)$ being determined uniquely, we assume that $\mathcal{R}(1)$, the (abstract) rotation of the vertex 1, contains a subsequence $(2, 3, 4)$.
- Order the quintuples of the vertices of A lexicographically and denote them by $Q_1, Q_2, \ldots, Q_{\binom{n}{5}}$.
- For every induced subgraph $B_k = A[Q_k]$, $k = 1, 2, \ldots, \binom{n}{5}$, check if it is one of the five simply realizable 5-vertex AT-graphs (their drawings are in the Figure 10). If not, the algorithm terminates and answers "NO", i.e., that A is not simply realizable. Otherwise we determine the rotation system $\mathcal{R}(B_k)$: we choose one of the two possible mutually inverse rotation systems, which is compatible with the rotation systems $\mathcal{R}(B_1)$,$\mathcal{R}(B_2)$,...,$\mathcal{R}(B_{k-1})$. (By the choice of the ordering of the quintuples Q_i, there exists $k' < k$ such that $|Q_k \cap Q_{k'}| = 4$. If u, v, w, z are the vertices of the intersection $Q_k \cap Q_{k'}$, then $\mathcal{R}(B_k)$ determines the order of the elements v, w, z in the rotation of u in $B_{k'}$, which then determines $\mathcal{R}(B_{k'})$.)

– For each vertex $v \in V(A)$, compute the rotation $\mathcal{R}(v)$ from the rotation systems $\mathcal{R}(B_k)$, such that $v \in Q_k$: we choose a "reference vertex" $u \neq v$ and consider all subsequences of elements u, w, z ($w, z \in V(A) \setminus \{u, v\}$, $w \neq z$) in the rotations of v in the rotation systems $\mathcal{R}(B_k)$. These ordered triples determine a complete oriented graph $G_{u,v}$ on the set $V(A) \setminus \{u, v\}$. The rotation of v is then determined by the topological order of the vertices of $G_{u,v}$, which can be found in linear time. If $G_{u,v}$ has an oriented cycle, the algorithm terminates and answers "NO".

At this stage we know that if A is simply realizable, then it has a simple realization with the computed rotation system $\mathcal{R}(A)$. But it may still happen that $\mathcal{R}(A)$ is not realizable as a rotation system of a simple complete topological graph. To decide this, we try to find an isomorphism class of some simple realization of A by constructing its star-cut representation.

By Proposition 3, we can determine the order of crossings of each edge with an arbitrary star $S(v)$, and also the rotation of all crossings on the edges of $S(v)$.

– Fix an arbitrary vertex $v \in V(A)$ and denote the other vertices of A by $w_1, w_2, \ldots, w_{n-1}$, such that $\mathcal{R}(v) = (w_1, w_2, \ldots, w_{n-1})$.
– Fix an orientation for each edge $w_i w_j$, $i < j$, by choosing w_i as an initial vertex.
– For every edge $e = w_i w_{i'}$ and every two edges $vw_j, vw_{j'}$ which cross e, determine the order $O_e(j, j')$ of crossings of e with vw_j and $vw_{j'}$ from the AT-graph $A[\{v, w_i, w_{i'}, w_j, w_{j'}\}]$.
– For every edge $e = w_i w_{i'}$, the orders $O_e(j, j')$ define a complete oriented graph on the $s_v(e)$ edges incident with v and crossing e. If this graph has an oriented cycle, terminate and answer "NO", otherwise construct a topological order O_e of its vertices (i.e., the order in which e crosses the edges incident with v). If e crosses one (or no) edge incident with v, then O_e is a one-element (or an empty) sequence.
– For every crossing c_e^j of the edges $e = w_i w_i'$ and vw_j determine its rotation $\mathcal{R}(c_e^j)$, from the rotation system $\mathcal{R}(A[w_i, w_{i'}, w_j, v])$.

Now we are ready to start a construction of a star-cut representation of a possible simple realization of A, which would be obtained by cutting the sphere along the edges of the star $S(v)$.

– Draw a convex $2(n-1)$-gon D and denote its boundary cycle as C. Denote the vertices of C counter-clockwise by $v_1, w_1, v_2, w_2, \ldots, v_{n-1}, w_{n-1}$. For $i = 1, 2, \ldots, n-1$, denote by f_{2i-1} the open edge $v_i w_i$, and by f_{2i} the open edge $w_i v_{i+1}$ (where $v_n = v_1$).
– Denote the edges of A not incident with v by $e_1, e_2, \ldots, e_{\binom{n-1}{2}}$. For each edge e_i define $s_v(e_i)+1$ pseudochords $e_{i,1}, e_{i,2}, \ldots, e_{i,s_v(e_i)+1}$. We interpret $e_{i,j}$ as a portion of the edge e_i between the $(j-1)$-th and the j-th crossing of e_i with some edge incident with v (where the 0-th and $(s_v(e_i)+1)$-th crossing is the initial and the terminal vertex of e_i), and we consider $e_{i,j}$ oriented consistently with e_i. Denote the initial vertex of $e_{i,j}$ by $a_{i,j}$ and the terminal

vertex by $b_{i,j}$. Note that $a_{i,j+1}$ and $b_{i,j}$ correspond to the same crossing (the j-th crossing of the edge e_i with some edge incident with v), which we denote by $c_{i,j}$.

- From the orders O_{e_i} and from the rotations of the crossings $c_{i,j}$ determine, for each $k = 1, 2, \ldots, 2(n-1)$, the set of the end-points $a_{i,j}$, $b_{i,j}$ lying on the edge f_k.
- For each $k = 1, 2, \ldots, n-1$, construct a sequence O_{w_k} of the one-element sets $\{a_{i,1}\}$, $\{b_{i,s_v(e_i)+1}\}$ containing the end-points lying at w_k, such that their order in O_{w_k} is the same as the clockwise order of the corresponding pseudochords incident with w_k, which is determined by the rotation $\mathcal{R}(w_k)$. Note that we consider the end-points of the distinct pseudochords as distinct objects, even if they are all identical with w_k.
- Construct a cyclic sequence O_C, as a concatenation of the sequences $\{f_1\}$, O_{w_1}, $\{f_2, f_3\}$, O_{w_2}, $\{f_2, f_3\}, \ldots, O_{w_{n-1}}, \{f_{2(n-1)}\}$.
- For every pseudochord $e_{i,j}$, construct its *type* $t(e_{i,j})$ which is defined as a pair (X, X') such that the sets X, X' are elements of O_C and $a_{i,j} \in X$, $b_{i,j} \in X'$. Note that if (X, X') is a type of some pseudochord $e_{i,j}$, then $X \neq X'$.

We claim that the knowledge of the types $t(e_{i,j})$ now suffices to determine the realizability of the AT-graph A (in a polynomial time).

We say that the types (X, X') and (Y, Y') are

interlacing if all the sets X, X', Y, Y' are distinct and if one of the cyclic sequences (X, Y, X', Y'), (X, Y', X', Y) is a subsequence of O_C,
avoiding if they are not interlacing and all the sets X, X', Y, Y' are distinct,
parallel if $(X, X') = (Y, Y')$ or $(X, X') = (Y', Y)$, and
adjacent otherwise, i.e., if exactly one of the following equalities holds: $X = Y$, $X = Y'$, $X' = Y$ or $X' = Y'$.

See Figure 3 for examples.

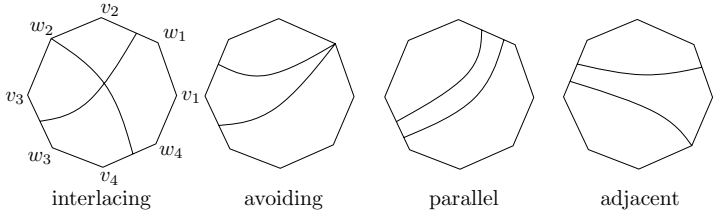


Fig. 3. Pairs of pseudochords with four different pairs of types

Clearly, if the types of two pseudochords $e_{i,j}$, $e_{i',j'}$ are interlacing, then $e_{i,j}$ and $e_{i',j'}$ are forced to cross (if drawn inside D), and if the types $t(e_{i,j})$, $t(e_{i',j'})$ are avoiding, then the pseudochords $e_{i,j}$ and $e_{i',j'}$ have no common crossing. The crossing status of two pseudochords with parallel or adjacent types is not uniquely determined, it depends on the order of their end-points on the edge(s)

f_k, containing an end-point of both pseudochords. However, we can deduce some information about these pseudochords if we group them into larger structures.

Let e_i,e'_i be two fixed edges. We define a *positive* (i,i')*-ladder* as an inclusion-maximal sequence $((e_{i,j},e_{i',j'}),(e_{i,j+1},e_{i',j'+1}),\ldots,(e_{i,j+k},e_{i',j'+k}))$, such that $k \geq 1$ and for each $l \in \{0,1,\ldots,k-1\}$ the two end-points $b_{i,j+l}$ and $b_{i',j'+l}$ ($a_{i,j+l+1}$ and $a_{i',j'+l+1}$) lie on a common edge f_p of C. It means that for each $l \in \{1,\ldots,k-1\}$, the edges $e_{i,j+l}$ and $e_{i',j'+l}$ have parallel types, and the edges $e_{i,j}$ and $e_{i',j'}$ have adjacent types, as well as the edges $e_{i,j+k}$ and $e_{i',j'+k}$. Similarly we define a *negative* (i,i')*-ladder* as an inclusion-maximal sequence $((e_{i,j},e_{i',j'}),(e_{i,j+1},e_{i',j'-1}),\ldots,(e_{i,j+k},e_{i',j'-k}))$, such that $k \geq 1$ and for each $l \in \{0,1,\ldots,k-1\}$ the two end-points $b_{i,j+l}$ and $a_{i',j'-l}$ ($a_{i,j+l+1}$ and $b_{i',j'-l-1}$) lie on a common edge f_p of C. Each (positive or negative) (i,i')-ladder corresponds to two maximal portions of the edges e_i, $e_{i'}$ which cross the same edges incident with v in the same order and from the same direction.

We call the (i,i')-ladder *crossing* if the two corresponding portions of edges are forced to cross, and *non-crossing* otherwise; see Figure 4. We can determine whether the (i,i')-ladder is crossing or not from the types of its pairs of pseudochords (we show that only for positive ladders, the other case is similar).

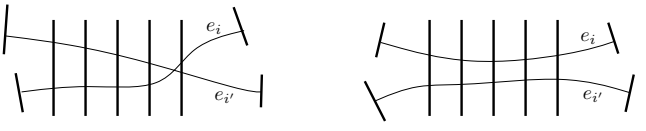


Fig. 4. A crossing and a non-crossing (i,i')-ladder (the fat lines represent distinct edges of the star $S(v)$)

Lemma 4. *Let* $L = ((e_{i,j},e_{i',j'}),(e_{i,j+1},e_{i',j'+1}),\ldots,(e_{i,j+k},e_{i',j'+k}))$ *be a positive* (i,i')*-ladder, let* $t(e_{i,j}) = (X,Z)$, $t(e_{i',j'}) = (Y,Z)$, $t(e_{i,j+k}) = (P,Q)$, *and* $t(e_{i',j'+k}) = (P,R)$. *Define* $t(L)$ *as a number from* $\{0,1\}$ *such that* $t(L) = 0$ *if and only if the sequences* (X,Y,Z) *and* (P,Q,R) *have the same orientation in* O_C, *i.e., either* (X,Y,Z) *and* (P,Q,R) *are both subsequences of* O_C *or both* (X,Z,Y) *and* (P,R,Q) *are subsequences of* O_C. *Then* L *is non-crossing if* $k+t(L)$ *is even, and crossing if* $k+t(L)$ *is odd.*

Proof. The proof is quite straightforward; the statement follows from the fact that for each $l \in \{0,1,\ldots,k-1\}$ the order of the end-points $b_{i,j+l}, b_{i',j'+l}$ on the common edge f_k of the cycle C is opposite to the order of the end-points $a_{i,j+l+1}, a_{i',j'+l+1}$ on the edge f_{k+o} ($o \in \{-1,1\}$) adjacent to f_k and representing the same edge of the star $S(v)$. □

Clearly, every pair $(e_{i,j},e_{i',j'})$ of pseudochords with adjacent or parallel types belongs to exactly one (i,i')-ladder. It follows that the set $P_{i,i'} = \{(e_{i,j},e_{i',j'}); 1 \leq j \leq s_v(e_i)+1, 1 \leq j' \leq s_v(e_{i'})+1\}$ can be uniquely partitioned into (i,i')-ladders and one-element sets consisting of pairs of pseudochords with interlacing or avoiding types. For each set Q from this partition, we are able to determine the

parity of the total number of crossings between the pairs of pseudochords from Q. Hence, we are able to determine the parity of the total number of crossings between the edges e_i and $e_{i'}$, and also a lower bound for this number.

We are now ready to describe the last steps of the recognition algorithm.

- For every two edges $e_i, e_{i'}$ $(1 \le i < i' \le \binom{n-1}{2})$ do the following:
  - determine the partition of $P_{i,i'}$ into (i,i')-ladders and pairs with interlacing or avoiding types. For each (i,i')-ladder from this partition, determine whether it is crossing or non-crossing.
  - Compute $\mathrm{CR}(e_i, e_{i'})$, the sum of the number of crossing (i,i')-ladders and the number of pairs of pseudochords from $P_{i,i'}$ with interlacing types.
  - Define $\mathrm{CR}_A(e_i, e_{i'}) \in \{0,1\}$ such that $\mathrm{CR}_A(e_i, e_{i'}) = 0$ if the edges $e_i, e_{i'}$ form a non-crossing pair in the abstract graph A and $\mathrm{CR}_A(e_i, e_{i'}) = 1$ if the edges $e_i, e_{i'}$ form a crossing pair in A.
- If there exist edges $e_i, e_{i'}$ such that $\mathrm{CR}(e_i, e_{i'}) \neq \mathrm{CR}_A(e_i, e_{i'})$, terminate and answer "NO". Otherwise answer "YES".

Clearly, if the algorithm answers "NO", the abstract graph A is not realizable. It remains to prove that if for every two edges $e_i, e_{i'}$ the equality $\mathrm{CR}(e_i, e_{i'}) = \mathrm{CR}_A(e_i, e_{i'})$ holds, then there exists a choice of the counter-clockwise orders O_{f_k} of the end-points of the pseudochords on the edges f_k, such that the induced number of crossings between any two edges $e_i, e_{i'}$ attains the lower bound $\mathrm{CR}(e_i, e_{i'})$. The orders O_{f_k}, together with the orders O_{w_k}, determine a counter-clockwise (perimetric) order PO_C of all the end-points $a_{i,j}, b_{i,j}$ on the cycle C. For each pair of the pseudochords, PO_C determines whether they cross or not. Note that for every given perimetric order PO_C the arrangement of the pseudochords can be realized, e.g., the pseudochords can be drawn as straight-line segments (i.e., as actual chords of the polygon D).

For every $k = 1, 2, \ldots, (n-1)$, the edges f_{2k-1} and f_{2k} represent the same edge, vw_k, of the graph A. Thus, the order $O_{f_{2k}}$ is an *almost-inverse* of $O_{f_{2k-1}}$, i.e., $O_{f_{2k}}$ is the inverse of the order, which we obtain from $O_{f_{2k-1}}$ by replacing each end-point $a_{i,j}$ $(b_{i,j})$ with the end-point $b_{i,j-1}$ $(a_{i,j+1})$ corresponding to the same crossing on the edge vw_k. Hence, PO_C is now uniquely determined by the orders $O_{f_2}, O_{f_4}, \ldots, O_{f_{2(n-1)}}$.

Lemma 5. *Let $O_{f_2}, O_{f_4}, \ldots, O_{f_{2(n-1)}}$ be the orders which minimize the total number of crossings between pseudochords induced by PO_C. Then for every two edges $e_i, e_{i'}$, the order PO_C induces exactly $\mathrm{CR}(e_i, e_{i'})$ crossings together on all the pairs of pseudochords from $P_{i,i'}$.*

Proof. Suppose that it is not the case. Then for some two edges $e_i, e_{i'}$, there exists an (i,i')-ladder L with at least two crossings induced by PO_C. Suppose, without loss of generality, that L is a positive ladder $((e_{i,j}, e_{i',j'}), (e_{i,j+1}, e_{i',j'+1}), \ldots, (e_{i,j+k}, e_{i',j'+k}))$. Let $q < r$ be the least integers such that PO_C induces a crossing c_q between $e_{i,j+q}$ and $e_{i',j'+q}$, and a crossing c_r between $e_{i,j+r}$ and $e_{i',j'+r}$. In the topological graph G represented by this pseudochord arrangement, the two portions $e'_i, e'_{i'}$ of the edges $e_i, e_{i'}$ between the crossings c_q and c_r form an

empty lens $L_{q,r}$, i.e., a region bounded by the curves $e'_i, e'_{i'}$, which contains no vertex of G. Hence, the total number of crossings of every other edge of G with the curves e'_i and $e'_{i'}$ is even. Assume that the lens $L_{q,r}$ is inclusion-minimal (over all pairs of edges $e_i, e_{i'}$). Then every connected component of every edge intersecting $L_{q,r}$ has one end-point on e'_i and the other end-point on $e'_{i'}$. Hence, every edge of G has the same number of crossings with e'_i as with $e'_{i'}$. It follows that by redrawing e'_i along the curve $e'_{i'}$, we decrease the total number of crossings in G by two (we get rid of the crossings c_q and c_r) and we do not change the type of any pseudochord in the corresponding star-cut representation of G; see Figure 5. The redrawing of the curve e'_i corresponds to the translations of the end-points $b_{i,j+q}, b_{i,j+q+1}, \ldots, b_{i,j+r-1}$ ($a_{i,j+q+1}, a_{i,j+q+2}, \ldots, a_{i,j+r}$) next to the end-points $b_{i',j'+q}, b_{i',j'+q+1}, \ldots, b_{i',j'+r-1}$ ($a_{i',j'+q+1}, a_{i',j'+q+2}, \ldots, a_{i',j'+r}$) in the corresponding orders O_{f_k} (the translated end-point is moved "just behind" the other end-point). We have constructed a perimetric order PO'_C which induces less crossings than PO_C, a contradiction. □

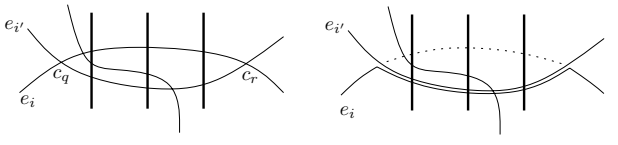


Fig. 5. An empty lens allows us to decrease the number of crossings by 2

Corollary 6. *If the algorithm answers "YES", then the abstract graph A is realizable.* □

The proof of Lemma 5 also gives an idea of an algorithmic construction of the perimetric order of a star-cut representation of a simple realization of A:

- Choose an arbitrary set of orders O_{f_2}, $O_{f_4}, \ldots, O_{f_{2(n-1)}}$ and compute the related orders $O_{f_1}, O_{f_3}, \ldots, O_{f_{2n-3}}$.
- while there exists some (i, i')-ladder with at least two induced crossings, find an inclusion-minimal lens $L_{q,r}$ and change the orders of the corresponding end-points in the corresponding orders O_{f_k}, as in the proof of Claim 5.
- Return the resulting perimetric order PO_C.

It is quite straightforward to verify that each step of the algorithm can be performed in polynomial time. Using a bounded number of quantifications over subsets (of vertices, edges, etc.) of bounded size, each step can be decomposed into a polynomial number of elementary tasks; either those solvable in constant time, or simple subroutines such as searching in a polynomial list or topological sorting of a partially ordered set. More concrete estimates on running time would require to describe the particular implementation and data structures in much more detail, and it would only increase the technical complexity of the paper.

The algorithm can be extended so that it finds some isomorphism class of the arrangement with the perimetric order PO_C. That is, it finds the order of crossings of the pseudochords with the other pseudochords. It is then an easy task to compute the orders of the crossings on the edges of the simple realization of A represented by the constructed arrangement.

Some difficulties with the computation of the orders may occur if the pseudochords were drawn as straight-line segments, because we could obtain pairs of crossings very close to each other (closer than the precision of our representation of real numbers), and they would become indistinguishable for the algorithm. So we choose a different approach and compute the orders recursively:

- Choose an arbitrary pseudochord p and from the perimetric order PO_C identify the set $\{p_1, p_2, \ldots, p_k\}$ of all pseudochords that cross p.
- Cut the circle C into two arcs, C_1 and C_2, by the end-points of p and define two circles $C'_1 = C_1 \cup p$ and $C'_2 = C_2 \cup p$. Partition the perimetric order PO_C into two orders O_{C_1} and O_{C_2} of the end-points on the arcs C_1 and C_2.
- Cut each pseudochord p_i, $i = 1, 2, \ldots, k$, into two portions with one end-point on p and the second end-point on C. Define two mutually almost-inverse orders O_p^1 and O_p^2 of these new end-points on p such that the portions of the pseudochords p_i between p and C_1 do not cross (O_p^1 is a counter-clockwise order with respect to the circle C'_1 and it can be deduced from O_{C_1}).
- Define $PO_{C'_1}$ as a concatenation of O_{C_1} and O_p^1, and $PO_{C'_2}$ as a concatenation of O_{C_2} and O_p^2.
- Recursively compute the orders of crossings on the pseudochords in the two arrangements with the perimetric orders $PO_{C'_1}$, $PO_{C'_2}$ and merge the computed orders for the portions of pseudochords p_i together.

Since we cut along each pseudochord at most once, this procedure also runs in polynomial time.

Acknowledgements

I am grateful to Pavel Valtr for introducing me to the topic of topological graphs, for many valuable suggestions and for the help with the preparation of this paper. I am also grateful to Jan Kratochvíl for helpful discussions about abstract topological graphs and NP-hardness and for presenting me his results.

References

1. Aichholzer, O., Aurenhammer, F., Krasser, H.: On the crossing number of complete graphs. Computing 76(1-2), 165–176 (2006)
2. Chrobak, M., Payne, T.H.: A linear-time algorithm for drawing a planar graph on a grid. Information Processing Letters 54, 241–246 (1995)
3. Gassner, E., Jünger, M., Percan, M., Schaefer, M., Schulz, M.: Simultaneous graph embeddings with fixed edges. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 325–335. Springer, Heidelberg (2006)

4. Guy, R.K.: A combinatorial problem. Nabla (Bulletin of the Malayan Mathematical Society) 7, 68–72 (1960)
5. Harborth, H., Mengersen, I.: Drawings of the complete graph with maximum number of crossings. In: Proceedings of the Twenty-third Southeastern International Conference on Combinatorics, Graph Theory, and Computing, Congressus Numerantium, Boca Raton, FL, vol. 88, pp. 225–228 (1992)
6. Kratochvíl, J., Matoušek, J.: String graphs requiring exponential representations. Journal of Combinatorial Theory Ser. B 53, 1–4 (1991)
7. Kratochvíl, J., Lubiw, A., Nešetřil, J.: Noncrossing subgraphs in topological layouts. SIAM Journal on Discrete Mathematics 4(2), 223–244 (1991)
8. Kratochvíl, J.: String graphs I: The number of critical nonstring graphs is infinite. Journal of Combinatorial Theory Ser. B 52, 53–66 (1991)
9. Kratochvíl, J.: String graphs II: Recognizing string graphs is NP-hard. Journal of Combinatorial Theory Ser. B 51, 67–78 (1991)
10. Kratochvíl, J.: A special planar satisfiability problem and a consequence of its NP-completeness. Discrete Applied Mathematics 52, 233–252 (1994)
11. Kynčl, J.: Enumeration of simple complete topological graphs. Electronic Notes in Discrete Mathematics 29, 295–299 (2007)
12. Kynčl, J., Valtr, P.: On edges crossing few other edges in simple topological complete graphs. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 274–284. Springer, Heidelberg (2006)
13. Pach, J., Solymosi, J., Tóth, G.: Unavoidable configurations in complete topological graphs. Discrete and Computational Geometry 30, 311–320 (2003)
14. Pach, J., Tóth, G.: Recognizing string graphs is decidable. Discrete and computational geometry and graph drawing (Columbia, SC, 2001), Discrete and Computational Geometry 28(4), 593–606 (2002)
15. Pach, J., Tóth, G.: How many ways can one draw a graph? In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 47–58. Springer, Heidelberg (2004)
16. Richter, R.B., Thomassen, C.: Relations between crossing numbers of complete and complete bipartite graphs. Amer. Math. Monthly 104(2), 131–137 (1997)
17. Schaefer, M., Sedgwick, E., Štefankovič, D.: Recognizing string graphs in NP. Journal of Computer and System Sciences 67, 365–380 (2003)
18. Schaefer, M., Štefankovič, D.: Decidability of string graphs. Journal of Computer and System Sciences 68, 319–334 (2004)
19. Shahrokhi, F., Székely, L.A., Vrt'o, I.: Crossing Numbers of Graphs, Lower Bound Techniques and Algorithms: A Survey. In: Tamassia, R., Tollis, I(Y.) G. (eds.) GD 1994. LNCS, vol. 894, pp. 131–142. Springer, Heidelberg (1995)
20. Wagner, U.: On the rectilinear crossing number of complete graphs. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, pp. 583–588. ACM, New York (2003)

A Appendix

A.1 Reduction from P3C3-SAT

First we describe the main idea of the reduction and then we show the specific modifications for each of the considered problems.

Let ϕ be a given instance of P3C3-SAT with the set of clauses C and the set of variables X. Chrobak and Payne [2] proved that it is possible to construct a

rectilinear planar drawing D_ϕ of G_ϕ on the integer $(2n-4) \times (n-2)$ grid in time $O(n)$ (where n is the number of vertices of G_ϕ).

Based on the drawing D_ϕ, we construct an abstract topological graph $A_\phi = ((V_\phi, E_\phi), R_\phi)$ as follows. We replace every clause vertex $c \in C$ by an AT-graph $H_c = ((V_c, E_c), R_c)$ and each variable vertex $x \in X$ by an AT-graph $H_x = ((V_x, E_x), R_x)$. Each graph H_c will have six *joining* vertices $L_c^{x_i(c)}$, $R_c^{x_i(c)}$, $i \in \{1,2,3\}$, where $x_1(c)$, $x_2(c)$, $x_3(c)$ are the neighbors of c in the drawing D_ϕ in clockwise order. Similarly, each graph H_x will have $2 \cdot \deg(x)$ *joining* vertices $L_x^{c_i(x)}, R_x^{c_i(x)}, i \in \{1,2,\ldots,\deg(x)\}$, where $\deg(x)$ is the number of clauses containing x and $c_1(x), c_2(x), \ldots, c_{\deg(x)}(x)$ are these clauses ordered clockwise according to the drawing D_ϕ. Then, for each clause c and variable $x \in c$ (i.e., for each edge in D_ϕ) we add a *joining* AT-graph $J_{c,x} = ((V_{c,x}, E_{c,x}), R_{c,x})$ on four vertices ($V_{c,x} = \{R_c^x, L_c^x, R_x^c, L_x^c\}$) and with two (joining) edges: if x has a positive occurrence in c, then $E_{c,x} = \{\{R_c^x, R_x^c\}, \{L_c^x, L_x^c\}\}$, otherwise $E_{c,x} = \{\{R_c^x, L_x^c\}, \{L_c^x, R_x^c\}\}$. We do not allow these two edges to intersect, so we put $R_{c,x} = \emptyset$. Note that we neither allow two edges from two different graphs H_c, H_x, $J_{c,x}$ to intersect.

Now, let $A'_\phi = ((V'_\phi, E'_\phi), R'_\phi)$, where

$$V'_\phi = \bigcup_{c \in C} V_c \cup \bigcup_{x \in X} V_x,$$

$$E'_\phi = \bigcup_{c \in C} E_c \cup \bigcup_{x \in X} E_x \cup \bigcup_{c \in C, x \in X, x \in c} E_{c,x},$$

$$R'_\phi = \bigcup_{c \in C} R_c \cup \bigcup_{x \in X} R_x.$$

In case of non-complete graphs we put $A_\phi = A'_\phi$, in case of complete graphs we well need to add all the missing edges and allow (or force) them intersect some other edges; we will specify this later.

The graphs H_c and H_x may be different for each of the considered problems, but we require that they satisfy the following common conditions (where the term "drawing" is a substitution for "realization", "simple realization", "weak realization", "simple weak realization" or "weak rectilinear realization"):

(C1) Every drawing of the graph H_c is connected (i.e., H_c need not be connected itself, but the union of the points and arcs in its drawing in the plane must be a connected set).

(C2) Suppose that H_c has a drawing where the vertices $L_c^{x_i(c)}, R_c^{x_i(c)}, i \in \{1,2,3\}$, are all incident with the outer face and their clockwise cyclic order O_c is $(Y_1, Z_1, Y_2, Z_2, Y_3, Z_3)$, where for each $i \in \{1,2,3\}$, we have $\{Y_i, Z_i\} = \{L_c^{x_i(c)}, R_c^{x_i(c)}\}$. There are exactly 8 such possible orders. H_c does not have a drawing with $O_c = (L_c^{x_1(c)}, R_c^{x_1(c)}, L_c^{x_2(c)}, R_c^{x_2(c)}, L_c^{x_3(c)}, R_c^{x_3(c)})$ and has a drawing with all the 7 remaining orders.

(X1) Every drawing of the graph H_x is connected.

(X2) Suppose that H_x has a drawing where the vertices $L_x^{c_i(x)}$, $R_x^{c_i(x)}$, for $i \in \{1, 2, \ldots, \deg(x)\}$, are all incident with the outer face and their clockwise cyclic order O_x is $(Y_1, Z_1, Y_2, Z_2, \ldots, Y_{\deg(x)}, Z_{\deg(x)})$, where for each $i \in \{1, 2, \ldots, \deg(x)\}$, we have $\{Y_i, Z_i\} = \{L_x^{c_i(x)}, R_x^{c_i(x)}\}$. Then $O_x = (L_x^{c_1(x)}, R_x^{c_1(x)}, L_x^{c_2(x)}, R_x^{c_2(x)}, \ldots, L_x^{c_{\deg(x)}(x)}, R_x^{c_{\deg(x)}(x)})$ or $O_x = (R_x^{c_1(x)}, L_x^{c_1(x)}, R_x^{c_2(x)}, L_x^{c_2(x)}, \ldots, R_x^{c_{\deg(x)}(x)}, L_x^{c_{\deg(x)}(x)})$. On the other hand, H_x has a drawing with both these cyclic orders of the joining vertices.

We claim that these conditions imply that A'_ϕ has a drawing if and only if ϕ is satisfiable (the only exception is the backward implication in the "weak rectilinear realization" case, with which we will deal separately, using more constraints on the graphs H_x and H_c):

Suppose that ϕ is satisfiable and let $f : X \to \{\text{TRUE}, \text{FALSE}\}$ be the satisfying evaluation of the variables. We replace each vertex $x \in X$ in the drawing D_ϕ by a small drawing of H_x such that the joining vertices of H_x lie on the outer face and their cyclic clockwise order is $(L_x^{c_1(x)}, R_x^{c_1(x)}, L_x^{c_2(x)}, R_x^{c_2(x)}, \ldots, L_x^{c_{\deg(x)}(x)}, R_x^{c_{\deg(x)}(x)})$ if $f(x) = \text{TRUE}$ and $(R_x^{c_1(x)}, L_x^{c_1(x)}, R_x^{c_2(x)}, L_x^{c_2(x)}, \ldots, R_x^{c_{\deg(x)}(x)}, L_x^{c_{\deg(x)}(x)})$ if $f(x) = \text{FALSE}$. Similarly, we replace each vertex $c \in C$ by a small drawing of H_c such that the joining vertices of H_c lie on the outer face and their clockwise cyclic order is $Y_1, Z_1, Y_2, Z_2, Y_3, Z_3$ where $\{Y_i, Z_i\} = \{L_c^{x_i(c)}, R_c^{x_i(c)}\}$, and $Y_i = R_c^{x_i(c)}$ if and only if the evaluation $f(x_i(c))$ satisfies the clause c. Then we draw the edges of the graphs $J_{c,x}$ along the edges of the drawing D_ϕ (from the construction it is clear that we can draw them without crossings).

Now suppose that A'_ϕ has a drawing. The 3-connectivity of G_ϕ and the conditions (C1) and (X1) imply that the drawing of each of the graphs $A'_\phi[V'_\phi \setminus V_c]$ and $A'_\phi[V'_\phi \setminus V_x]$ is connected. Since the joining edges ($E_{x,c}$) are without crossings, for each graph H_c and H_x its joining vertices lie on the boundary of a common face, which is without loss of generality the outer face. After contracting the edges of the graphs H_c and H_x and replacing each pair of parallel joining edges by a single edge we get a planar drawing of G_ϕ. The 3-connectivity of G_ϕ implies that this drawing has the same or the inverse rotation system as the drawing D_ϕ (and so we can assume that they are the same). This allows only 8 possible clockwise cyclic orders of the joining vertices of the graphs H_c and, by the condition (X2), only two such possible orders for the graphs H_x. According to the orientation of the pairs $L_x^{c_i(x)}, R_x^{c_i(x)}$ in the drawings of the graphs H_x we define an evaluation f of the variables such that $f(x) = \text{TRUE}$ if and only if $O_x = (L_x^{c_1(x)}, R_x^{c_1(x)}, L_x^{c_2(x)}, R_x^{c_2(x)}, \ldots, L_x^{c_{\deg(x)}(x)}, R_x^{c_{\deg(x)}(x)})$. These orders are uniquely "translated" by the joining edges into the cyclic clockwise orders O_c of the joining vertices of the graphs H_c. Since each of these graphs has a drawing, the cyclic order O_c corresponds to some of the 7 satisfying evaluations of the 3 variables contained in c; see Figure 6.

Now we construct the clause and variable gadgets H_c and H_x for each of the considered types of realization.

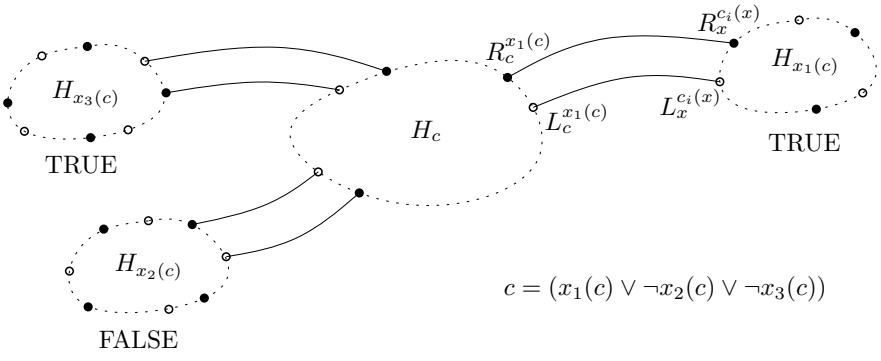


Fig. 6. Variables $x_1(c)$ and $x_2(c)$ satisfy the clause c

A.2 Realizability

For this problem we use almost the same variable and clause gadget as Kratochvíl [9]. For every $c \in C$ let

$$V_c = \bigcup_{i=1}^{3} \{D_c^i, L_c^{x_i(c)}, K_c^{x_i(c)}, R_c^{x_i(c)}, P_c^{x_i(c)}\},$$

$$d_c^i = \{D_c^i, D_c^{i+1}\}, l_c^i = \{L_c^{x_i(c)}, K_c^{x_i(c)}\}, r_c^i = \{R_c^{x_i(c)}, P_c^{x_i(c)}\},$$

$$E_c = \bigcup_{i=1}^{3} \{d_c^i, l_c^i, r_c^i\},$$

$$R_c = \bigcup_{i=1}^{3} \{\{d_c^i, l_c^i\}, \{d_c^i, r_c^i\}, \{l_c^i, l_c^{i+1}\}, \{r_c^i, r_c^{i+1}\}, \{l_c^i, r_c^{i+1}\}\}$$

(the indices are taken modulo 3). For every $x \in X$ let

$$V_x = \bigcup_{i=1}^{\deg(x)} \{A_x^i, B_x^i, L_x^{c_i(x)}, R_x^{c_i(x)}\},$$

$$l_x^i = \{L_x^{c_i(x)}, A_x^i\}, r_x^i = \{R_x^{c_i(x)}, B_x^i\},$$

$$E_x = \bigcup_{i=1}^{\deg(x)} \{\{A_x^i, B_x^i\}, \{B_x^i, A_x^{i+1}\}, l_x^i, r_x^i\},$$

$$R_x = \bigcup_{2 \le i \ne j \le \deg(x)} \{\{l_x^i, l_x^j\}, \{r_x^i, r_x^j\}, \{l_x^i, r_x^j\}\}.$$

The conditions (C1) and (X1) are obviously satisfied. The existence of the realizations of H_c for the 7 cyclic orders of the joining vertices from the condition (C2) is proved in [9] and the non-realizability of H_c with the cyclic order $O_c = (L_c^{x_1(c)}, R_c^{x_1(c)}, L_c^{x_2(c)}, R_c^{x_2(c)}, L_c^{x_3(c)}, R_c^{x_3(c)})$ is proved in [8]. The condition (X2) for the realizability of the graph H_x is proved in [9]. Note that we cannot use this variable gadget for the simple realizability problem, since for the order O_x corresponding to the positive evaluation of the variable x some pairs of edges in the realization of H_x have to cross an even number of times. However, we will use this AT-graph as the variable gadget for all three considered weak versions of realizability.

To obtain a complete AT-graph A_ϕ, we add all the missing edges to the graph A'_ϕ and force them to intersect all the other edges, i.e., we put

$$V_\phi = V'_\phi, E_\phi = \binom{V_\phi}{2},$$

$$R_\phi = R'_\phi \cup \{\{e, f\}; e \in E_\phi \setminus E'_\phi, f \in E_\phi, e \neq f\}.$$

Clearly, if A_ϕ is realizable, then A'_ϕ is realizable too, since it is an induced subgraph of A_ϕ. On the other hand, every realization of A'_ϕ can be extended into a realization of A_ϕ by drawing the remaining edges such that they intersect every other edge (although some pairs of edges may have to cross many times). This proves that the realizability is NP-hard for complete AT-graphs. The NP-completeness then follows from the fact that the realizability of AT-graphs is in NP [17].

A.3 Simple Realizability

We use the same clause gadget H_c as in the realizability case, since H_c can be simply realized for any satisfying evaluation of its variables [9]. We define H_x as follows:

$$V_x = \{C\} \cup \bigcup_{i=1}^{\deg(x)} \{L_x^{c_i(x)}, R_x^{c_i(x)}, P_x^i\},$$

$$l_x^i = \{L_x^{c_i(x)}, C\}, r_x^i = \{R_x^{c_i(x)}, P_x^i\},$$

$$E_x = \bigcup_{i=1}^{\deg(x)} \{l_x^i, r_x^i\},$$

$$R_x = \bigcup_{i=1}^{\deg(x)} \bigcup_{1 \leq j \leq \deg(x), j \neq i} \{r_x^j, l_x^i\}.$$

Figure 7 shows simple realizations of H_x with the two cyclic orders O_x from condition (X2). It remains to show that these two orders are the only possible.

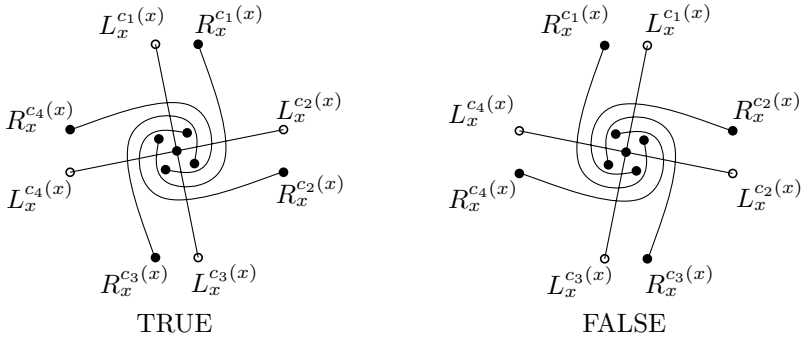


Fig. 7. A variable gadget for the simple realizability problem

Let D_x be a realization of H_x satisfying the assumptions of (X2). We may assume that all the joining vertices of D_x lie on a circle q and all the edges of D_x lie inside q. The edges l_x^i form a topological star which divides the interior of q into $\deg(x)$ regions. For each edge r_x^j there are exactly two possible orders in which it crosses the edges $l_x^i, i \neq j$, either the clockwise or the counter-clockwise order. The order also uniquely determines the position of the vertex $R_x^{c_j(x)}$ on q (according to the vertices $L_x^{c_i(x)}$). Now if the edge r_x^j crosses the edges of the star in clockwise order, then so does the edge r_x^{j+1}, since r_x^j and r_x^{j+1} must be disjoint. By induction, all the edges r_x^j cross the edges l_x^i in the same direction, so there are only two possible orders O_x. This finishes the proof of the NP-completeness of the simple realizability problem (it is trivially in NP, since the simple realizations have polynomial number of crossings).

A.4 Weak Types of Realizability

We use the same clause and variable gadgets for the weak realizability, the simple weak realizability and the weak rectilinear realizability. As we mentioned before, the variable gadget will be the same AT-graph H_x as for the realizability problem. It is easy to see that the weak realizations of H_x satisfying the assumptions of the condition (X2) can have only two possible orders of the joining vertices (depending on the orientation of the cycle $A_x^1, B_x^1, \ldots, B_x^{\deg x}$). On the other hand, H_x has a weak rectilinear realization with both these orders; see Figure 8. It follows that (X2) is satisfied for all three weak versions of realizability. However, we will need weak rectilinear realizations of H_x with another restrictions.

We define H_c as follows:

$$V_c = \bigcup_{i=1}^{3} \{L_c^{x_i(c)}, R_c^{x_i(c)}\} \cup \{X, Y, Z\},$$

$$E_c = \{a, b, e, f, u, v, x, y\}$$

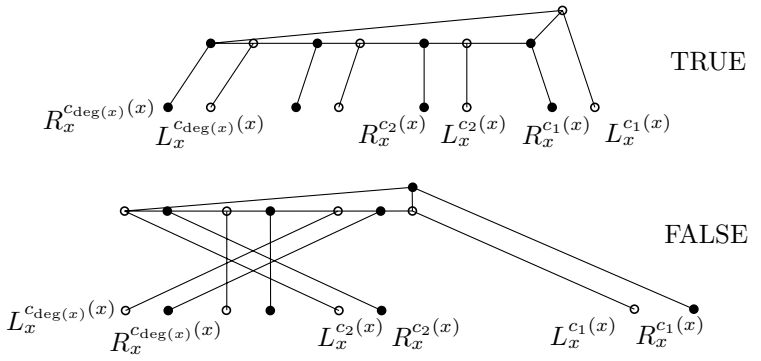


Fig. 8. A variable gadget for the weak realizability problem

where

$$a = \{L_c^{x_3(c)}, Y\}, b = \{R_c^{x_2(c)}, Y\}, e = \{L_c^{x_1(c)}, Y\}, f = \{R_c^{x_1(c)}, Y\},$$

$$u = \{R_c^{x_1(c)}, X\}, v = \{L_c^{x_1(c)}, Z\}, x = \{R_c^{x_3(c)}, X\}, y = \{L_c^{x_2(c)}, Z\},$$

$$R_c = \{\{x, y\}, \{x, b\}, \{y, a\}, \{u, a\}, \{u, b\}, \{v, a\}, \{v, b\}\}.$$

Suppose that H_c has a weak realization satisfying the assumptions of the condition (C2) and that the order of the joining vertices is ($L_c^{x_1(c)}$, $R_c^{x_1(c)}$, $L_c^{x_2(c)}$, $R_c^{x_2(c)}$, $L_c^{x_3(c)}$, $R_c^{x_3(c)}$). We can assume that all the six joining vertices lie on a common circle q and that H_c is contained inside q. All the four edges starting at the vertex Y are disjoint, hence they divide the interior of q into four regions; see Figure 9. The vertex $R_c^{x_3(c)}$ lies between $L_c^{x_3(c)}$ and $L_c^{x_1(c)}$ and the edge x can not intersect edges a and e, so x lies in the region bounded by the edges a and e. Similarly, y lies in the region bounded by b and f. It implies that x and y are disjoint. According to the order of the vertices $L_c^{x_1(c)}$, $R_c^{x_1(c)}$, $L_c^{x_2(c)}$, $R_c^{x_3(c)}$ on q, the paths xu and yv must have at least one crossing. But the only pair of the edges x, u, y, v which is allowed to intersect, is the pair $\{x, y\}$; a contradiction.

For each satisfying evaluation of the clause c, the AT-graph H_c has a weak rectilinear realization with the corresponding order of the joining vertices. See Figure 9 for the five non-symmetric cases.

The proof of the NP-hardness of the weak realizability and the simple weak realizability of AT-graphs is now finished. In case of the weak rectilinear realizability we must ensure that the edges of the joining graphs $J_{c,x}$ can be drawn as straight-line segments.

First, for each vertex v of the drawing D_ϕ, we choose a line t_v going through v such that t_v is not parallel to any edge of D_ϕ. This line determines a direction in which the corresponding gadget H_v will be oriented. For each variable vertex x we choose a line t_x such that the edge $xc_1(x)$ is the first in the clockwise order

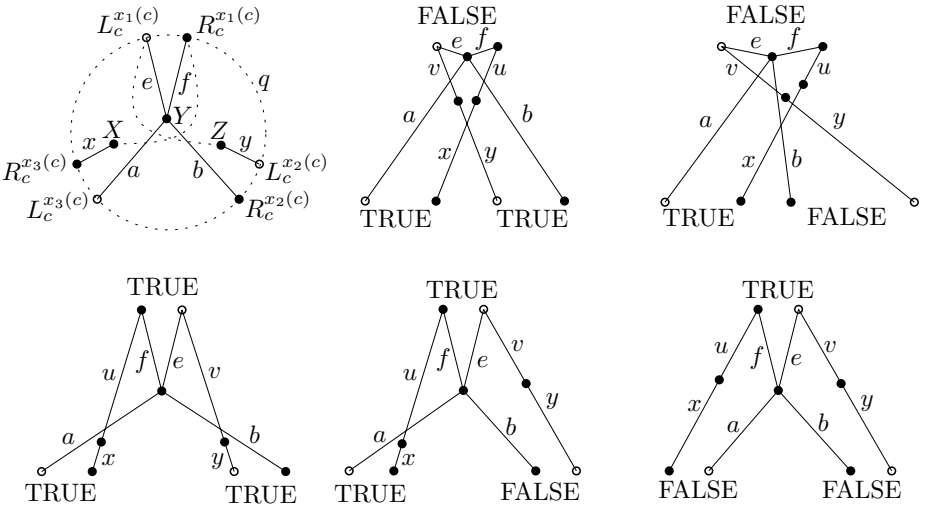


Fig. 9. A clause gadget for the weak realizability problem

of the edges $xc_i(x)$ in one of the half-planes determined by t_x. For each clause vertex c we choose a line t_c such that among the three edges incident with c one edge, $\{c, x(c)\}$, is separated from the other two edges. Then we change the labeling of the neighbors of c such that $x_1(c) = x(c)$.

Figure 9 certifies the validity of the following condition for H_c:

(C3) For each of the 7 orders of the joining vertices from condition (C2) there exists a weak rectilinear realization D_c of H_c which lies inside a rectangle M_c, and all the joining vertices of D_c lie on the perimeter of M_c on two opposite (parallel) edges, such that $L_c^{x_1(c)}$ and $R_c^{x_1(c)}$ lie on one edge, $e(M_c)$, and the other four joining vertices lie on the opposite edge, $f(M_c)$.

When drawing the AT-graph A'_ϕ, we place each clause gadget H_c over the original vertex c of D_ϕ such that $e(M_c)$ is parallel with t_c and lies in the same half-plane as the vertex $x_1(c)$, while $f(M_c)$ lies in the opposite half-plane. Then each neighbor $x_i(c)$ can be connected by a straight-line segment with the corresponding joining vertices $L_c^{x_i(c)}$ and $R_c^{x_i(c)}$ without crossing.

We deal similarly with the variable gadgets. We require the following condition to be satisfied:

(X3) For both orders of the joining vertices from condition (X2) and for every integer $k \in \{0, 1, \ldots, \deg(x)\}$ there exists a weak rectilinear realization D_x of H_x which lies inside a rectangle M_x, and all the joining vertices of D_x lie on the perimeter of M_x on two opposite (parallel) edges, such that the vertices $\{L_x^{c_i(x)}, R_x^{c_i(x)}; i \leq k\}$ lie on one edge, $e(M_x)$, and the other $2(\deg(x) - k)$ joining vertices lie on the opposite edge, $f(M_x)$.

If (X3) holds for each variable x, we place each variable gadget H_x over the vertex x of D_ϕ such that $e(M_x)$ is parallel with t_x and lies in the same half-plane as the vertex $c_1(x)$, while $f(M_c)$ lies in the opposite half-plane. Then it is safe to add all the joining edges as straight-line segments and we obtain a weak rectilinear realization of A'_ϕ.

Examples of the drawings satisfying condition (X3) for $k = 0$ are in the Figure 8. But it is not hard to transform them into the drawings satisfying (X3) for other values of k: all the intersections of the half-lines $A^i_x L^{c_i(x)}_x$, $B^i_x R^{c_i(x)}_x$ lie inside the rectangle M_x and their directions are changing monotonously with i. For a given $k \in \{0, 1, \ldots, \deg(x)\}$, we choose a direction α between the directions of the k-th and the $(k+1)$-th pair of the half-lines. We choose two lines $e(\alpha)$ and $f(\alpha)$ with the direction α such that the rectangle M_x lies inside the strip bounded by these two lines and the half-line $A^1_x L^{c_1(x)}_x$ intersects $e(\alpha)$. Then the half-lines $A^i_x L^{c_i(x)}_x$, $B^i_x R^{c_i(x)}_x$, where $i \leq k$, intersect $e(\alpha)$ and the other half-lines intersect $f(\alpha)$. We prolong the half-lines by translating the joining vertices to the corresponding intersections with the border lines $e(\alpha)$ and $f(\alpha)$. We obtain a drawing of H_x which satisfies (X3) with a given parameter k. The proof of the NP-hardness of the weak rectilinear realizability is now finished.

For the case of complete AT-graphs, we put

$$V_\phi = V'_\phi, E_\phi = \binom{V_\phi}{2},$$

$$R_\phi = R'_\phi \cup \{\{e, f\}; e \in E_\phi \setminus E'_\phi, f \in E_\phi, e \neq f\}.$$

It is now easy to prove that the resulting complete AT-graph $A_\phi = ((V_\phi, E_\phi), R_\phi)$ is weakly (simply, rectilinearly) realizable if and only if the AT-graph A'_ϕ is. Indeed, we have proved that all the three weak versions of the realizability are equivalent for the AT-graph A'_ϕ, the weak realizability of A_ϕ implies the weak realizability of its induced subgraph A'_ϕ, and every weak rectilinear realization of A'_ϕ can be extended to a weak rectilinear realization of A_ϕ by slightly perturbing the vertices into a general position and adding all the remaining edges as straight-line segments. This finishes the proof of the NP-hardness of all the three versions of the weak realizability of complete AT-graphs.

Since the weak realizability and the simple weak realizability are in NP, they are NP-complete problems for the class of AT-graphs and also for the class of complete AT-graphs.

A.5 Proof of Proposiotion 3

(1) Let G and G' be two weakly isomorphic simple complete topological graphs on n vertices. First we prove that the rotation systems $\mathcal{R}(G)$ and $\mathcal{R}(G')$ are either the same or inverse.

For $n \leq 3$ it is trivial, for $n = 4$ and $n = 5$ it follows from the fact that for the simple complete topological graphs with 4 or 5 vertices the

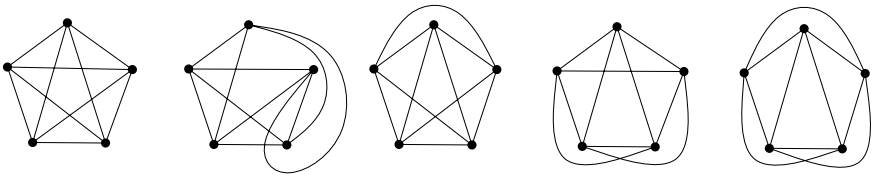


Fig. 10. All five non-isomorphic simple drawings of K_5 [5]

isomorphism classes coincide with the weak isomorphism classes: there are two non-isomorphic simple drawings of K_4 and five non-isomorphic simple drawings of K_5 (see [5] or Figure 10) and each of them is a realization of a different AT-graph.

Now we use the case $n = 5$ to extend the statement to graphs with more than five vertices. Let A be a simply realizable complete AT-graph with the vertex set $\{1, 2, \ldots, n\}$, where $n \geq 6$. We know that each complete 5-vertex subgraph of A has only two possible rotation systems. Suppose that the rotation system of $A[\{1, 2, 3, 4, 5\}]$, the induced subgraph of A with the vertices $1, 2, 3, 4, 5$, is fixed (in some simple realization of A). We show that then the rotation system of every other 5-vertex complete subgraph of A is uniquely determined.

Lemma. *Let B and C be two 5-vertex complete subgraphs of A with exactly 4 common vertices. Then the rotation system $\mathcal{R}(B)$ uniquely determines the rotation system $\mathcal{R}(C)$.*

Proof of lemma. Without loss of generality, let $V(B) = \{1, 2, 3, 4, 5\}$, $V(C) = \{1, 2, 3, 4, 6\}$ and let the rotation of the vertex 1 in $\mathcal{R}(B)$ be $(2, 3, 4, 5)$. Then the rotation of 1 in $A[\{1, 2, 3, 4\}]$ is $(2, 3, 4)$ and it must be a subsequence of a rotation of 1 in $\mathcal{R}(C)$. But this always happens for exactly one of the pair of inverse cyclic permutations of the set $\{2, 3, 4, 6\}$. It follows that the rotation of 1 in C is uniquely determined and so is the whole rotation system of C. □

By repeated use of this lemma we obtain that the rotation system of every complete subgraph of A on 5 (and also 4) vertices is uniquely determined by $\mathcal{R}(A[\{1, 2, 3, 4, 5\}])$. It remains to show that this also uniquely determines the rotation of each vertex in A. But this easily follows from the fact that a cyclic order of a finite set X is uniquely determined by the cyclic order of all 3-element subsets of X (actually, it suffices to know the orders of the triples containing one fixed vertex). It follows that a simple realization of A can have only two possible rotation systems.

Since G and its mirror image have inverse extended rotation systems, it remains to prove that $\mathcal{R}(G)$ uniquely determines the rotation $\mathcal{R}(c)$ of each crossing c of G. Let uv, wz be the edges that cross at c. Then $\mathcal{R}(c)$

is determined by the drawing of the induced subgraph $H = G[\{u, v, w, z\}]$. Since every two weakly isomorphic simple drawings of K_4 are isomorphic, and an isomorphism preserves or inverts the extended rotation system, it follows that $\mathcal{R}(c)$ is determined by $\mathcal{R}(H)$, which is trivially determined by $\mathcal{R}(G)$.

(2) The edges e, f, f' are contained in a complete 5-vertex subgraph H of G, so the order of crossings of e with f and f' is determined by the isomorphism class of H, which is determined by the AT-graph of H. □

Efficient Extraction of Multiple Kuratowski Subdivisions

Markus Chimani, Petra Mutzel, and Jens M. Schmidt

Department of Computer Science, University of Dortmund, Germany
{markus.chimani,petra.mutzel,jens.schmidt}@uni-dortmund.de

Abstract. A graph is planar if and only if it does not contain a Kuratowski subdivision. Hence such a subdivision can be used as a witness for non-planarity. Modern planarity testing algorithms allow to extract a single such witness in linear time. We present the first linear time algorithm which is able to extract multiple Kuratowski subdivisions at once. This is of particular interest for, e.g., Branch-and-Cut algorithms which require multiple such subdivisions to generate cut constraints. The algorithm is not only described theoretically, but we also present an experimental study of its implementation.

1 Introduction

A *planar drawing* of a graph is an injection of its vertices onto points in the plane, and a mapping of the edges into open curves between their endpoints. These curves are not allowed to touch each other, except in their common endpoints. Graphs which admit such a planar drawing, are called *planar graphs*, and recognizing this graph class has been a vivid research topic for the past decades. Hopcroft and Tarjan [11] showed in 1974 that this problem can be solved in linear time, using sophisticated data structures and intricate algorithms. Current planarity testing algorithms like the ones by Boyer and Myrvold [4,5] and de Fraysseix et al. [9,10] are less complex but still quite involved.

As shown by Kuratowski [13] in 1930, a graph is planar if and only if it does not contain a $K_{3,3}$ or a K_5 subdivision, i.e., a complete bipartite graph $K_{3,3}$ or complete graph K_5 with edges replaced by paths of length at least one. Such subgraphs are called *Kuratowski subdivisions*. The efficient extraction of such a witness of non-planarity was non-trivial in the context of the first linear planarity tests. A linear algorithm for such an extraction was later presented, e.g., by Williamson [15]. Modern planarity testing algorithms like the ones by Boyer and Myrvold, and de Fraysseix et al. can directly extract a single Kuratowski subdivision, if the given graph is non-planar.

In ILP-based Branch-and-Cut approaches which try to solve, e.g., the Maximum Planar Subgraph problem [12] or the Crossing Minimization problem [6], the identification of multiple such witnesses is a crucial part. Thereby, we look at some intermediate solution and try to find Kuratowski subdivisions. For each such subdivision, we can try to generate a cut constraint, necessary to efficiently

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 159–170, 2007.

solve the ILP. Experience shows that it is desirable to find multiple Kuratowski constraints at once, as they strengthen the LP-relaxation of the problem.

In the following, let $G = (V, E)$ be a non-planar undirected graph, without selfloops and multi-edges. Current planarity tests are able to extract a single Kuratowski subdivision in linear time $O(n)$, $n := |V|$. We address the problem of finding multiple Kuratowski subdivisions in efficient time. As there may exist exponentially many Kuratowski subdivisions in general, it is not practical to enumerate all of them. A basic approach would be to obtain k Kuratowski subdivisions through calling a planarity test k times and subsequently deleting an involved Kuratowski edge. This approach has a superlinear runtime of $O(kn)$, but we are not aware of any algorithm faster than this approach, up until now.

In this paper, we propose an algorithm which extracts multiple Kuratowski subdivisions in optimal time $O(n + m + \sum_{K \in \mathcal{S}} |E(K)|)$, with $\mathcal{S}$ being the set of identified Kuratowski subdivisions and $m := |E|$. This runtime is linear in the graph size and the extracted Kuratowski edges. The algorithm is based on the planarity test of Boyer and Myrvold [5] which is one of the fastest planarity tests today [3]. We will only give a short introduction into this planarity test in Section 2; for a full description of the original test see [5]. The main part of this paper focuses on the description on how to modify and extend all steps to obtain multiple subdivisions in linear time, which requires both algorithmic changes, as well as a heavily modified runtime analysis. Finally, Section 4 gives a short computational study which shows the effectiveness of this algorithm.

2 The Boyer-Myrvold Planarity Test

The test starts with a depth first search on the (not necessarily connected) input graph, which divides the edge set into DFS-forest edges and into backedges, pointing to nodes with smaller *depth first index* DFI. The aim is to construct a planar drawing based on the DFS-forest, by successively embedding all backedges in descending DFI order of their end vertices. Throughout this paper, let v be the current vertex to embed. Any backedge ending on v is called *pertinent* and will be embedded, if this is possible while maintaining planarity. In the beginning, each DFS-edge is separated from its adjacent vertex with lower DFI and joined to a new *virtual* vertex. Therefore it represents a biconnected component *(bicomp)* in the beginning, which grows when backedges are embedded.

To identify involved bicomps during such an embedding, the *Walkup* is called for each start node of a pertinent backedge. A bicomp consisting of only one DFS-edge and its adjacent vertices is called *degenerated*. The Walkup marks the involved subgraph and classifies nodes as *pertinent* and *external*: a node w is called *pertinent*, if there exists a pertinent backedge $\{w, v\}$ or if w has a child bicomp in the DFS-tree which contains a pertinent node. A node w is called *external*, if there exists a backedge $\{w, u\}$ with u having a smaller DFI than v, or if w has a child bicomp containing an external node. Bicomps are called pertinent or external if they contain pertinent or external vertices, respectively. The Walkup traverses a unique path from w to v on the external faces of bicomps

for every pertinent backedge $\{w, v\}$. We denote this path as the *backedge path* of $\{w, v\}$.

The *Walkdown* attempts to embed each pertinent backedge and merges the bicomps between its start and end vertex in the DFS-tree to a new, larger bicomp. It is invoked twice for each child bicomp of v: once in a counterclockwise direction around the external face of the child bicomp, and once in the clockwise direction. Using the classification of nodes from the Walkup, the Walkdown embeds only backedges which preserve planarity in the embedding. If any backedge cannot be embedded, the graph is not planar and a subdivision is extracted; otherwise a planar embedding is found. Since non-embeddable backedges can only occur when both Walkdowns stop on external vertices which are not pertinent, such a situation is called a *stopping configuration*. We call unembedded pertinent backedges caused by a stopping configuration *critical*. Let $b = \{w, v\}$ such a critical backedge. The first node in the backedge path of b which is contained in the same bicomp as both stopping vertices are, is called *critical node*. We denote the part of the backedge path from w to this critical node *critical back path*.

3 Extracting Multiple Kuratowski Subdivisions in Linear Time

As opposed to the Boyer-Myrvold planarity test, the number of edges cannot be bounded linearly by the number of vertices. Since every algorithm has to read the input graph and to output all identified Kuratowski subdivisions, $\Omega(n+m+\sum_{K\in\mathcal{S}}|E(K)|)$ is a lower bound for the runtime and our algorithm is therefore optimal for the extracted number of Kuratowski edges.

3.1 Overview

The original planarity test terminates when a stopping configuration is found. It is possible to extract a Kuratowski subdivision for each critical backedge of this stopping configuration. To obtain more, we have to proceed with the algorithm. This bears problems, since the embedding has to be maintained planar, which is impossible if it contains Kuratowski subdivisions. The idea is to identify all critical backedges in the given stopping configuration and delete them. After that, the bicomp B containing the stopping configuration is not pertinent anymore and it is necessary to continue at the situation directly before the planarity test descended to B. This allows finding the next stopping configuration, provided that there exists any on the current embedding step of vertex v. See Algorithm 1 for an overview of these steps for the embedding of a single vertex v.

Unfortunately, almost all time-bounds given in [5] loose validity with this approach, and a new runtime analysis of this extended algorithm is necessary. The key to a linear time bound is to compensate additional costs during Walkup, Walkdown and extraction by the amount of extracted Kuratowski edges.

We will first describe how to find the correct reentry point after a stopping configuration was found and removed. In Section 3.3, we discuss how to modify

Algorithm 1. Embedding tasks of a vertex v

```
1: for all pertinent backedges p ending at v do
2:     Walkup(p)                                              ▷ Sect. 3.3
3: end for
4: for all DFS-children c of v do
5:     stop ← Walkdown(c)                                     ▷ original Walkdown
6:     while stop ≠ ∅ do
7:         Find all critical backedges of the stopping configuration stop   ▷ Sect. 3.4
8:         Extract multiple subdivisions for each critical backedge         ▷ Sect. 3.4
9:         Delete critical backedges and update the classification of nodes ▷ see [5]
10:        Find reentry_point for further embedding                         ▷ Sect. 3.2
11:        stop ← Walkdown(reentry_point)                   ▷ iterated Walkdowns
12:    end while
13: end for
```

the Walkup, in order to allow efficient operations used in the later steps of the algorithm. Section 3.4 deals with the efficient extraction phase. Finally, the overall runtime of the extended algorithm is analyzed in Section 3.5.

Of course there are graphs with exactly one Kuratowski subdivision. Hence, we do not ensure any lower bound other than 1 for the number of extracted Kuratowski subdivisions of non-planar graphs. But in practice, the quantity is high as discussed in Section 4. Formally, our algorithm guarantees:

Lemma 1. *We find at least one unique Kuratowski subdivision for each critical backedge per stopping configuration.*

Lemma 2. *Whenever the algorithm extracts a Kuratowski subdivision using a critical backedge b, and there exists at least one additional Kuratowski subdivision without b, we will find such a subdivision.*

3.2 Finding the Reentry Point for Further Embeddings

Let v' be the virtual node of v adjacent to the DFS-child c of v from the current Walkdown. We call the bicomp which has v' as its root, the *forebear bicomp*, the others are called *non-forebear bicomps*. The Walkdown can be run unmodified, as long as no stopping configuration occurs. The same holds if a stopping configuration occurs on the forebear bicomp due to embedded pertinent backedges, since this represents the last stopping configuration in the Walkdown.

Otherwise, the Walkdown has to be modified. Let A be the non-forebear bicomp containing the stopping configuration, T the subtree of all pertinent bicomps with the bicomp containing v' as root and D the parent bicomp of A in T (cf. Figure 1). Any bicomp in T has exactly those bicomps as children which are referenced in the `PertinentRoots` lists of its nodes, as proposed in [5]. In Figure 1, the bicomp tree T consists of the (degenerated) forebear bicomp $\{v', c\}$ and the non-forebear bicomps A, B, C and D. The Walkdown stops at A, deleting the critical backedges incident to w_1 and w_2 after the extraction of all Kuratowski subdivisions induced by these backedges. Afterwards, A is not

pertinent anymore and its `PertinentRoots` list entry on the parent node z_1 in D must be deleted. As there exists another item in that list, we continue the Walkdown at z_1 and find another stopping configuration in bicomp B. The general rule is that the Walkdown continues on z_1 until the `PertinentRoots` list of z_1 is empty.

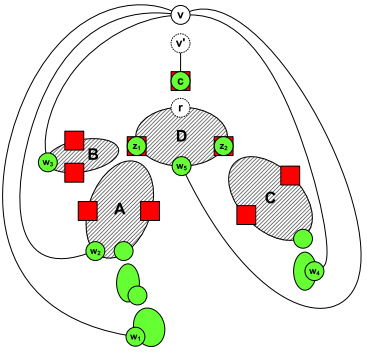


Fig. 1. Finding reentry points. Square nodes refer to external vertices; circular, light gray nodes denote pertinent vertices. Virtual vertices are depicted by a dotted line.

At last, z_1 is not pertinent anymore. Furthermore, *short-circuit edges* from the root r of D to both external vertices in each direction (z_1 and z_2) have been embedded.These short-circuit edges permit an $O(1)$-traversal to the other external vertex z_2, where the Walkdown extracts all stopping configurations of child bicomps (bicomp C in Figure 1), analogously to z_1. Finally, we check whether D itself contains a stopping configuration by extracting all remaining critical edges. In our example, the backedge starting at w_5 induces a subdivision and can be deleted after the subdivision's extraction. This procedure is iterated for the next father bicomp in the DFS-tree until the forebear bicomp is reached or a pertinent backedge is embedded. In the latter case, all preceding bicomps are embedded and the Walkdown continues at the forebear bicomp.

The crucial point in this scheme is the traversal to a bicomp, where no backedge can be embedded, i.e., a bicomp that contains a stopping configuration: we modify the embedding to what it would have been, if no critical backedges on this bicomp would have existed. Finally, the Walkdown is restarted on the very node where the previous Walkdown started to descent to this bicomp.

3.3 Walkup

Additionally to the `PertinentRoots` list and `BackedgeFlags` of the original planarity test, we now have to collect some more information during the Walkup. For every visited node n, we store a link `LinkToRoot` to the root node of the bicomp of n. This can be done efficiently by using a stack for all visited nodes of the bicomp during the Walkup. Furthermore, a list named `PertinentNodesAfterWalkup` of all pertinent nodes of each bicomp B is created. This is stored at the root node of B by collecting the nodes during the Walkup in a list. Whenever we reach the bicomp root or a node with set `LinkToRoot`, we can add the collected vertices in $O(1)$ time to the list of the bicomp root. Once established, this list is not modified until v is completely embedded.

It is useful to be able to distinguish the backedges incident to different virtual vertices v' of v, since they will be embedded in different subtrees later on. This can be done by storing v' as the `HighestVirtualNode` for each backedge $\{w, v\}$.

To obtain v' for a given backedge p, Walkup(p) marks each visited node with p. If the Walkup ends on a virtual node of v, we can store this node as the `HighestVirtualNode(`p`)`. Otherwise, Walkup(p) stopped on an already visited vertex which was traversed during the Walkup of another backedge q. Since both Walkups met, the subtrees are identical and so are the `HighestVirtualNodes` of p and q. The latter can be looked up in $O(1)$, and we hence identified `HighestVirtualNodes(`p`)`. This allows us to easily generate a list `BackedgesOnVirtualNode` for each virtual node v' of v containing the backedges belonging to the pertinent subtree with root v'.

3.4 Extraction

Overview. The extraction starts whenever the Walkdown halts on some stopping configuration in a bicomp B. We describe how the critical backedges of this stopping configuration can be computed in the next subsection *"Extraction of Critical Backedges"*. Each critical back path of those backedges induces one or more Kuratowski subdivisions of a specific minor-type, which has to be known prior to the extraction. To obtain this minor-type, a path from each stopping vertex to a node with lower DFI than v is selected in time linearly to its length.

Additionally, the *highest-xy-path* of the critical node w is needed to determine the minor-type. As defined by Boyer and Myrvold, the highest-xy-path obstructs the inner face of B and consists of the external face part on the top of the former, now embedded, bicomp which contains w. This path can be computed in $O(n)$, but this would result in a superlinear overall runtime. Hence we develop a more efficient way by first extracting the more general *highest-face-path* efficiently and use it to obtain the highest-xy-paths for all critical nodes. These steps are described in the subsections *"Extraction of the Highest-Face-Path"* and *"Extraction of all Highest-XY-Paths"*. After the minor-type is determined, all remaining parts of the Kuratowski subdivision can be extracted from the DFS-tree using only external faces of involved bicomps. This requires time linearly to their lengths. Finally, all critical backedges of the stopping configuration as well as the involved `PertinentRoots` and `BackedgeFlags` are deleted. We will give a rather high level description of the extraction, referring the reader to [7,14] for technical details and case distinctions.

Extraction of Critical Backedges. Let x and y be the two stopping vertices on the bicomp B, and r the root of B. Neither x, nor y, nor any node on the external face paths $r \to x$ and $r \to y$ can be pertinent; otherwise the Walkdown would not have stopped at x and y. The critical back paths of the critical backedges end on the external face of B between x and y. We distinguish between two cases depending on the type of B.

If B is a forebear bicomp, all pertinent backedges of the current Walkdown are contained in the `BackedgesOnVirtualNode(`r`)` list. For each entry, we can check in $O(1)$ whether it is embedded. If not, the backedge is critical. This yields an overall running time of $O(n + m)$ over all embedding steps, since all critical backedges are deleted afterwards and no further stopping configuration can exist.

If B is a non-forebear bicomp, consider the DFS-subtree T of pertinent bicomps with B as root bicomp. We start a preorder traversal through T by using the `PertinentNodesAfterWalkup` lists on the roots of all bicomps. These lists can contain nodes that are not pertinent any more due to extractions of other stopping configurations. Hence we have to check each item for pertinence; every non-pertinent entry is deleted. The remaining nodes are the critical nodes and we check their `BackedgeFlag` property. If this flag is set, the associated backedge must be critical and is therefore included in the list of critical backedges. Note that the remaining nodes, independent of their `BackedgeFlag`, may have non-empty `PertinentRoots` lists. After all critical backedges starting at the current bicomp were found, the preorder traversal iterates the process on each child bicomp given by its `PertinentRoots` lists recursively.

All tests on the nodes can be performed in constant time. The size of the tree T itself is bounded by the costs of the corresponding Walkup invocations, because at least one node was traversed for each pertinent bicomp. Moreover, a non-pertinent node in the `PertinentNodesAfterWalkup` list can only happen as a result of an earlier extracted stopping configuration. The only other reason would be that a pertinent backedge has been embedded on B, which contradicts the assumption. Each of the at most m stopping configurations in all embedding steps produces at most one non-pertinent entry in a `PertinentNodesAfterWalkup` list. Hence the overall runtime is bounded by the Walkup time.

Independent of the case distinction on B, all critical nodes in B are necessary for the minor-type classification and for the extraction of Kuratowski subdivisions. We can obtain all critical nodes in B efficiently by testing the `BackedgeFlag` for each entry of the `PertinentNodesAfterWalkup` list of r. From the above description we can conclude:

Lemma 3. *The asymptotic runtime for obtaining all critical backedges of a stopping configuration is bounded by the Walkup costs.*

Extraction of the Highest-Face-Path. In order to extract all highest-xy-paths efficiently, we first require a *highest-face-path* of the bicomp B. See Figure 2 for a visualization of the following explanations. We obtain the highest-face-path by temporarily deleting all edges incident to its root r except for the two edges $s = (r, a)$ and $t = (r, b)$ on the external face (ignoring any short-circuit edges). Thereby, B breaks into multiple sub-bicomps; we also delete all *separated* sub-bicomps, i.e., the sub-bicomps which do not contain r. Consider the inner face f containing a, r, and b. The highest-face-path is the path $a \rightarrow b$ on the boundary of f not traversing r.

It is possible to extract the highest-face-path in time $O(|B|)$, if B is properly embedded. But since the planarity test performs implicit flips on bicomps, we do not know whether the adjacency lists of the nodes are in clockwise or counterclockwise order, and we would have to establish the correct orientation for each node of B first. This requires a traversal of the underlying DFS-tree, resulting in a superlinear overall runtime. Hence, this approach is not suitable and we will identify the highest-face-path with inconsistent node orientations instead.

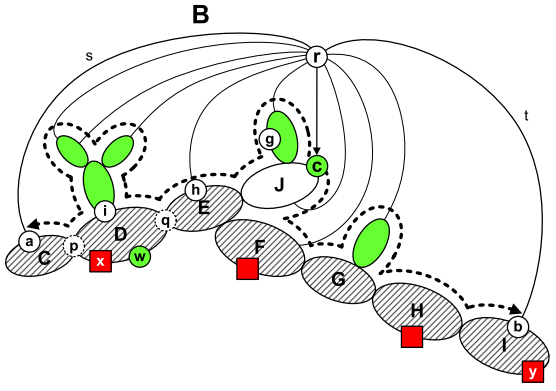


Fig. 2. The structure of the bicomp B containing former bicomps. The hatched former bicomps form the bottom chain. The extraction of the highest-face-path starts at the inner vertex c in both directions (thick dotted arrow lines) and ends on nodes a and b.

Therefore, it is not possible to easily walk along f. The idea is to reuse the external face links, which were introduced in the original planarity test, of the former, now merged bicomps in B. These *external-links* of a node referred to the two incident edges on the boundary and could be used in a traversal of the external face in order to find the correct direction to proceed, even when some nodes are not oriented correctly. Unfortunately, the Walkdown will usually modify those external-links. Therefore, we store a backup copy *old-links* of the external-links on each bicomp root during the Walkup.

To use the former external-links in a traversal inside of a non-degenerated B, we have to analyze the general structure of B first: the external face of every non-degenerated forebear bicomp contains at most one embedded backedge for each of the two Walkdowns formerly started at r. It may also contain an edge connecting the root and the non-root node with least DFI. However, in all cases these edges are incident to the virtual root node. The remaining set of edges on the external face consists of the lower parts of now connected, former bicomps. We denote this sequence of former bicomps which lie on the external face the *bottom chain* of B, cf. Figure 2. A *merge node* is a node shared between two adjacent bicomps of the bottom chain (e.g. the nodes q in Figure 2), or one of the two end nodes a and b. Given a former bicomp U in the bottom chain, the path on the upper part of U connecting the two contained merge nodes resembles the highest-xy-path of a critical back path ending at U. This fact is the key for the later extraction of all highest-xy-paths.

Let c be the unique non-virtual node of B with smallest DFI. Let E be the former bicomp of the bottom chain which contains the node with smallest DFI: if c is not contained in E, inner bicomps exist. Hence, we can summarize the necessary traversal as follows: We start with the traversal at c. If neither s nor t is an external-link of c, c is either an inner vertex or the root of E which lies on the external face of B. The former induces inner bicomps along a path from c to

the root of E. In both cases, we traverse the boundary of former bicomps in both directions. If an external-link of c is either s or t, c lies on the external face, and we have to traverse only one direction, following the other external-link of c. If we use two traversal directions, E can be determined as the last bicomp, whose root node is visited by both traversals. Starting with this root, all traversed nodes are stored in two separate lists, one for each traversal direction. We obtain the highest-face-path of B by appending the reversed second list to the first one. All walks check on each visited node z whether z is identical to a or b in $O(1)$. If so, the walk is finished. During the traversal, all visited nodes are saved on a stack. If a node is visited twice, this node is a merge node to an inner, separated subbicomp, whose boundary is not part of the highest-face-path. Then, all nodes between the two occurrences are deleted from the stack.

We store the highest-face-path on the unique vertex c in B, since later extractions might need it as well. Whenever a highest-face-path has to be computed in consequence of an embedding of B within a larger bicomp B^*, B will play the role of a former bicomp. Since we only traverse the external faces of former bicomps, we will never again traverse the interior of B. Hence, and since the traversals require $O(1)$ time for each vertex, we obtain:

Lemma 4. *All highest-face-paths which occur during the algorithm can be computed and maintained in $O(n + m)$.*

Extraction of all Highest-XY-Paths. For every given critical node w between two stopping vertices of a stopping configuration, we have to compute its highest-xy-path. Let D be the former bicomp of the bottom chain of B. By traversing the external face of D from w in parallel, using again the old-links, we find the merge nodes and extract the highest-xy-path in linear time of its length. For details see [7].

Extraction of Kuratowski Subdivisions. The prior sections dealt with the problem of efficiently obtaining and classifying multiple stopping configurations. We now address the problem to extract multiple Kuratowski subdivisions out of a single stopping configuration. Whenever a stopping configuration occurs, an appropriate critical back path for each critical backedge is computed. Along with the highest-xy-path, the minor-type of the induced Kuratowski subdivision is obtained. Additionally to the basic 9 minor-types of [5], we can define 7 more minor-types, by augmenting the types B, C, D and E_1–E_4 with a non-empty path $v \rightarrow r$ as in type A. We call the resulting minor-types AB, AC, AD and AE_1–AE_4, respectively. It turns out that the Kuratowski subdivisions of these additional minor-types constitute the largest part of the extracted subdivisions in practice, see Section 4. Clearly, more than one minor-type can exist for a single critical back path.

To further increase the number of extracted subdivisions, we will start with focussing on the critical back paths, since nearly all minor-types need them for constructing the subdivision. In general, such a path consists of external face parts between the roots of multiple consecutive bicomps. We can therefore extract the other parts of these external faces and combine these to obtain

potentially exponentially many different critical back paths, which yield different Kuratowski subdivisions. As a side effect, those subdivisions are all similar which can be beneficial for the application area of Branch-and-Cut algorithms. The same technique can be used to obtain multiple external backedge paths and multiple paths starting at the so-called *external z-nodes* [5] in the minor-types E_1–E_5 and AE_1–AE_4.

All extracted Kuratowski subdivisions of a stopping configuration are unique. This holds for subdivisions of different stopping configurations as well, except for the minor-types E_2 and AE_2, which do not include the critical back path and thus might be extracted as minor-type A later on. This can be avoided by a special marker on the external backedges, to prohibit its classification as a future critical backedge in A.

Bundle Variant. Moreover, we can extend our algorithm by a *bundle variant* in which all root-to-root paths of each involved bicomp on a critical back path are extracted. This approach increases the number of identified subdivisions dramatically, albeit on the cost of the running time. To speed up the backtracking subroutine, it is possible to use algorithms for dynamic connectivity for planar graphs [8]. This increases the overall runtime only by a factor of $log(n)$ in comparison to the linear time approach in terms of output complexity.

3.5 Runtime Analysis

All steps described so far guaranteed an overall linear runtime. It remains to show that the modified Walkup can be bound by a linear total of $O(n + m + \sum_{K \in \mathcal{S}} |E(K)|)$. We will only give a brief sketch of the proof, and omit a number of rather technical case differentiations (see [7,14]).

It is sufficient to consider the costs of the Walkup, which cannot be compensated by new embedded faces or new short-circuit edges. Therefore, we only consider Walkup costs on critical backedge paths. If these are part of stopping configurations on non-forebear bicomps, the sum of all critical backedge-path costs on all forebear bicomps can be estimated as follows: we spend at most $O(n+m+\sum_{K \in \mathcal{S}} |E(K)|)$ time on the external face, and at most $O(m)$ time on inner faces containing the forebear root. Moreover, all other costs caused by stopping configurations in non-forebear bicomps are compensated by the inevitably induced minor A which contains all other traversed edges.

Otherwise, the stopping configuration is contained in a forebear bicomp B. Since most minor-types do not contain the whole external face of B in their Kuratowski subdivisions, all not yet compensated costs arise on its external face. The only exception to this rule are the critical paths on minors E_2, AE_2, which can be bound by a linear total as well. These remaining costs are compensated by the extracted Kuratowski paths of the different minor-types. Hence we yield Theorem 1, which is optimal in terms of output complexity. Based on this, we can furthermore deduce a corresponding result for the bundle variant.

Theorem 1. *The overall running time of the algorithm is bounded by* $O(n + m + \sum_{K \in \mathcal{S}} |E(K)|)$ *and therefore linear.*

Corollary 1. *The overall running time of the bundle variant is* $O(n + m + \log n \sum_{K \in \mathcal{S}} |E(K)|)$.

4 Experimental Analysis

We implemented the algorithm and its bundle variant as part of the open-source C++-based *Open Graph Drawing Framework* [1]. All tests were performed on an Intel Core2Duo with 1.86 GHz and 2GB RAM using the GNU-compiler gcc-3.4.4 (-o1). Due to the algorithmic complexities, we simplified the steps to compute the critical backedges and highest-xy-paths by correctly orienting B in time $O(|B|)$. Although this simplification breaks the provable linear runtime, our experiments show that it does not influence the running time negatively in practice, since the number of extracted Kuratowski edges becomes the dominant term. The bundle variant uses a traditional back-tracking scheme and therefore does not guarantee the theoretical logarithmic bound. We use the Rome Graph Library [2], which contains 11528 real-world graphs with 10 to 100 nodes, 8249 of which are non-planar graphs. We also use random graphs ($n = 10 \ldots 500$, $m = 2n$) generated by OGDF. Thereby we start with an empty graph on n vertices and iteratively add an edge with random start and end node, until m unique edges are added.

Each Rome graph is processed in less than 11 ms (on average: 1.3 ms). The average amount of extracted Kuratowski subdivisions per 100-node graph is 255, containing in total 12214 Kuratowski edges. It is interesting that the average size of the subdivisions grows approximately with $n/2$ throughout all tests. More Kuratowski subdivisions are obtained by the bundle variant. Thereby, each graph is processed in less than 1 sec (but on average less than 7 ms), extracting up to 3.5 million Kuratowski edges at some graphs (see Figure 3). There are 2912 subdivisions on average per 100-node graph with 136027 Kuratowski edges.

On the random graphs, the number of identified Kuratowski subdivisions increases dramatically for the bundle variant, such that a full computation becomes prohibitive. In practice, one can of course stop the computation after a certain amount of extracted subdivisions. Hence, we restrict our test to the linear variant for these random graphs (see Figure 3). Each graph needs less than 430 ms

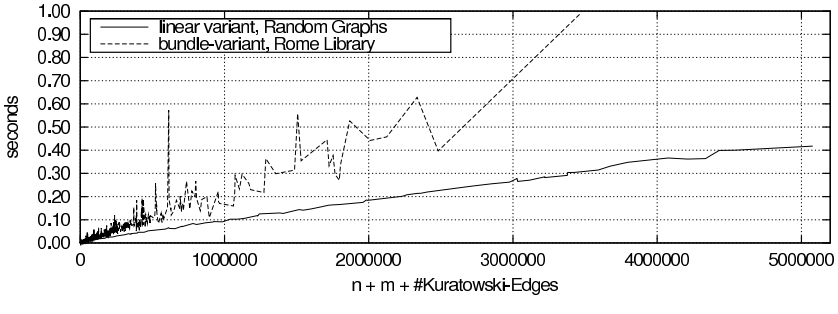


Fig. 3. Running times. The linear variant for the Rome Library would be nearly invisible in the very left corner of the figure.

(126 ms on average), extracting up to 25000 Kuratowski subdivisions per graph containing 5 million Kuratowski edges. The average number of Kuratowski subdivisions is 8813 per graph with 1.3 million Kuratowski edges.

Overall, the experiments show a linear running time, despite the aforementioned simplifications of the algorithm. The minor-types are dominated by the types AE_1–AE_4, which constitute 60%–90% of all subdivisions on graphs with at least 100 nodes.

References

1. OGDF - Open Graph Drawing Framework. University of Dortmund, Chair of Algorithm Engineering, Website under Construction (2007)
2. Di Battista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., Vargiu, F.: An experimental comparison of four graph drawing algorithms. Comput. Geom. Theory Appl. 7(5-6), 303–325 (1997)
3. Boyer, J.M., Cortese, P.F., Patrignani, M., Di Battista, G.: Stop minding your P's and Q's: Implementing a fast and simple DFS-based planarity testing and embedding algorithm. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 25–36. Springer, Heidelberg (2004)
4. Boyer, J.M., Myrvold, W.J.: Stop minding your P's and Q's: A simplified O(n) planar embedding algorithm. In: Proc. SODA 1999, pp. 140–146. SIAM, Philadelphia, PA (1999)
5. Boyer, J.M., Myrvold, W.J.: On the cutting edge: Simplified O(n) planarity by edge addition. Journal of Graph Algorithms an Applications 8(3), 241–273 (2004)
6. Buchheim, C., Chimani, M., Ebner, D., Gutwenger, C., Jünger, M., Klau, G.W., Mutzel, P., Weiskircher, R.: A branch-and-cut approach to the crossing number problem. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 37–48. Springer, Heidelberg (2006)
7. Chimani, M., Mutzel, P., Schmidt, J.M.: Efficient extraction of multiple Kuratowski subdivisions (TR). Technical Report TR07-1-002, Chair for Algorithm Engineering, Dep. of CS, University Dortmund (2007), http://ls11-www.cs.uni-dortmund.de/people/chimani/
8. Eppstein, D., Italiano, G.F., Tamassia, R., Tarjan, R.E., Westbrook, J.R., Yung, M.: Maintenance of a minimum spanning forest in a dynamic planar graph. J. Algorithms 13(1), 33–54 (1992)
9. de Fraysseix, H., de Mendez, P.O.: On cotree-critical and DFS cotree-critical graphs. Journal of Graph Algorithms an Applications 7(4), 411–427 (2003)
10. de Fraysseix, H., Rosenstiehl, P.: A characterization of planar graphs by Trémaux orders. Combinatorica 5(2), 127–135 (1985)
11. Hopcroft, J., Tarjan, R.: Efficient planarity testing. J. ACM 21(4), 549–568 (1974)
12. Jünger, M., Mutzel, P.: Maximum planar subgraphs and nice embeddings: Practical layout tools. Algorithmica 16(1), 33–59 (1996)
13. Kuratowski, K.: Sur le problème des corbes gauches en topologie. Fundamenta Mathematicæ 15, 271–283 (1930)
14. Schmidt, J.M.: Effiziente Extraktion von Kuratowski-Teilgraphen. Diploma thesis, Department of Computer Science, University of Dortmund (March 2007)
15. Williamson, S.G.: Depth-first search and Kuratowski subgraphs. J. ACM 31(4), 681–693 (1984)

Cover Contact Graphs

Nieves Atienza[2,⋆], Natalia de Castro[2,⋆], Carmen Cortés[2,⋆], M. Ángeles Garrido[2,⋆], Clara I. Grima[2,⋆], Gregorio Hernández[1], Alberto Márquez[2,⋆], Auxiliadora Moreno[2,⋆], Martin Nöllenburg[3,⋆⋆], José Ramon Portillo[2,⋆], Pedro Reyes[2,⋆], Jesús Valenzuela[2,⋆], Maria Trinidad Villar[2,⋆], and Alexander Wolff[4]

[1] Dept. Matemática Aplicada, Fac. Informática, Univ. Politécnica de Madrid, Spain
gregorio@fi.upm.es
[2] Universidad de Sevilla, Spain
{natienza, natalia, ccortes, vizuete, grima, almar, auxiliadora, josera, preyes, jesusv, villar}@us.es
[3] Fakultät für Informatik, Universität Karlsruhe, Germany
noelle@ira.uka.de
[4] Faculteit Wiskunde en Informatica, TU Eindhoven, The Netherlands
http://www.win.tue.nl/~awolff

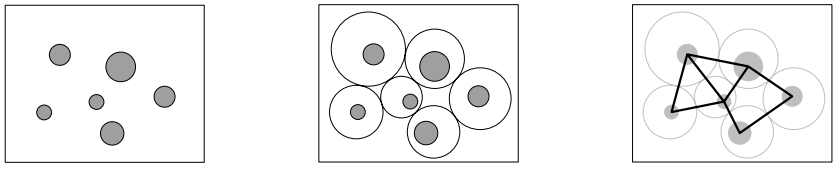
Abstract. We study problems that arise in the context of covering certain geometric objects (so-called *seeds*, e.g., points or disks) by a set of other geometric objects (a so-called *cover*, e.g., a set of disks or homothetic triangles). We insist that the interiors of the seeds and the cover elements are pairwise disjoint, but they can touch. We call the contact graph of a cover a *cover contact graph* (CCG). We are interested in two types of tasks: (a) deciding whether a given seed set has a connected CCG, and (b) deciding whether a given graph has a realization as a CCG on a given seed set. Concerning task (a) we give efficient algorithms for the case that seeds are points and covers are disks or triangles. We show that the problem becomes NP-hard if seeds and covers are disks. Concerning task (b) we show that it is even NP-hard for point seeds and disk covers (given a fixed correspondence between vertices and seeds).

1 Introduction

Koebe's theorem [9,11], a beautiful and classical results in graph theory, says that every planar graph can be represented as a *coin graph*, i.e., a contact graph of disks in the plane. In other words, given any planar graph with n vertices, there is a set of n disjoint open disks in the plane that are in one-to-one correspondence to the vertices such that a pair of disks is tangent if and only if the corresponding vertices are adjacent. Koebe's theorem has been rediscovered several times, see the survey of Sachs [12]. Collins and Stephenson [4] give an efficient algorithm for numerically approximating the radii and locations of the disks of such a

⋆ Partially supported by projects *PAI* FQM—0164 and *ORI* MTM2005-08441-C02-01.
⋆⋆ Supported by grant WO 758/4-2 of the German Research Foundation (DFG).

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 171–182, 2007.



(a) disk seeds **(b)** disk cover of (a) **(c)** CCG induced by (b)

Fig. 1. Seeds, cover, and CCG

representation of a planar graph. Their algorithm relies on an iterative process suggested by Thurston [13].

Since Koebe there has been a lot of work in the graph-drawing community dedicated to the question which planar graphs can be represented as contact or intersections graphs of which geometric object. As a recent example, Fraysseix and Ossona de Mendez [5] showed that any four-colored planar graph without an induced four-colored C_4 is the intersection graph of a family of line segments.

On the other hand, there has been a lot of work in the geometric-optimization community dedicated to the question how to (optimally) cover geometric objects (usually points) by other geometric objects (like convex shapes, disks, annuli). As an example take Welzl's famous randomized algorithm [15] for finding the smallest enclosing ball of a set of points.

In this paper we combine the two previous problems: we are looking for geometric objects (like disks or triangles) whose interiors are disjoint, that cover given pairwise disjoint objects called *seeds* (like points or disks) and at the same time represent a given graph or graph property by the way they touch each other. Other than in geometric optimization each of our covering objects contains only one of the seeds. We are not interested in maximizing the sizes of the covering objects; instead we want them to jointly fulfill some graph-theoretic property (like connectivity). Compared to previous work on geometric representation of graphs we are more restricted in the choice of our representatives.

Let us get a bit more formal. Given a set S of pairwise disjoint seeds of some type, a *cover* of S is a set $\mathcal{C}$ of closed objects of some type with the property that each object contains exactly one seed and that the interiors of no two objects intersect. Figure 1b depicts a disk cover of the disk seeds in Figure 1a. Now the *cover contact graph* (CCG) induced by $\mathcal{C}$ is the contact graph of the elements of $\mathcal{C}$. In other words, two vertices of a CCG are adjacent if the corresponding cover elements touch, i.e., their boundaries intersect. Figure 1c depicts the CCG induced by the cover in Figure 1b. Note that the vertices of the CCG are in one-to-one correspondence to both seeds and cover elements. We consider seeds to be topologically open (except if they are single points). Then seeds can touch each other. (Note that we require cover objects to be closed. This makes sure that a cover actually contains a point seed that lies on its boundary.)

In this paper we investigate the following questions.

Connectivity: Given a seed set, does it have a (1- or 2-) connected CCG?

Realizability: Given a planar graph and a set of seeds, can the given graph be realized as a CCG on the given seeds?

A third type of question is treated in the long version of this article [3]:

Enumeration: For a given number of vertices, how many graphs of a certain graph class can be realized as a CCG?

However, we do consider in this paper an interesting restriction of the above problems where seeds and cover elements must lie in the half plane $\mathbb{R}^2_+$ above and including the x-axis. Seeds are additionally restricted in that each must contain at least one point of the x-axis. In this restricted setting we call the contact graph of a cover a CCG$^+$. See Figures 7b and 9 for examples.

Our results. First, we consider arbitrary sets of point seeds, see Section 2. Concerning connectivity we show that we can always cover a set of point seeds using disks or using homothetic triangles such that the resulting CCG is 1- or even 2-connected. Our algorithms run in $O(n \log n)$ expected and $O(n^2)$ worst-case time, respectively. Concerning realizability we give some necessary conditions and then show that it is NP-hard to decide whether a given graph can be realized as a disk-CCG if the correspondence between vertices and point seeds is given. Second, we consider the restriction where we are given a set S of points on the x-axis as seeds. We show that in this case 1-connectivity is easy: we can realize C_n as a CCG on S and there are trees that can be realized as a CCG$^+$ on S. For the case that the correspondence between seeds and vertices is given, we give an algorithm that decides in $O(n \log n)$ time which trees can be realized as CCG$^+$. Third, we consider disk seeds, see Section 4. We show that even deciding whether a set of disk seeds has a connected disk-CCG is NP-hard. We can only sketch proofs here. We refer the reader to the long version [3] of this paper.

Related work. Abellanas et al. [1] proved that the following problem, which they call the *coin placement problem*, is NP-complete. Given n disks of varying radii and n points in the plane, is there a way to place the disks such that each disk is centered at one of the given points and no two disks overlap?

Abellanas et al. [2] considered a related problem. They showed that given a set of points in the plane, it is NP-complete to decide whether there are disjoint disks centered at the points such that the contact graph of the disks is connected.

Given a pair of touching (convex) cover elements, we can draw the corresponding edge in the CCG by a two-segment polygonal line that connects the incident seeds and uses the contact point of the cover elements as bend. This is a link to the problem of point-set embeddability. We say that a planar graph G is *k-bend (point-set) embeddable* if for any point set $P \subset \mathbb{R}^2$ there is a one-to-one correspondence between V and P such that the edges of G can be drawn as non-crossing polygonal lines with at most k bends. Kaufmann and Wiese [8] showed that (a) every 4-connected planar graph is 1-bend embeddable, (b) every planar graph is 2-bend embeddable, and (c) given a planar graph $G = (V, E)$ and a set P of n points on a line, it is NP-complete to decide whether G has a 1-bend embedding that maps V one-to-one on P.

2 The Seeds Are Points in the Plane

In this section we study point seeds which may take any position in the plane. If not stated otherwise our results hold for both disk covers and (homothetic) triangle covers. We focus on the two questions raised before: connectivity and realizability.

2.1 Connectivity

It is known to be NP-hard to decide whether a given set of points can be covered by a set of pairwise disjoint open disks, each centered on a point, such that the contact graph of the disks is connected [2]. In contrast to that result we give a simple sweep-line algorithm that covers point seeds by (non-centered) disks such that their contact graph is connected.

Proposition 1. *Every set S of n point seeds has a connected CCG. Such a CCG can be constructed in $O(n \log n)$ time and linear space.*

Proof. After sorting S by decreasing ordinate we proceed incrementally from top to bottom. For the first point, we place a cover element (disk or triangle, depending on the case) of fixed size with the seed as its bottommost point. If the $k-1$ topmost points are already connected, then for the k-th point p we inflate a cover element C_p with p as the bottommost point until C_p touches one of the previously placed cover elements.

The implementation for disk-CCGs is similar to Fortune's sweep [6] for constructing the Voronoi diagram of a set of weighted points. For triangle-CCGs we repeatedly determine the size of the new triangle in $O(\log n)$ time by a *segment-dragging query* [10] and two very simple ray-shooting queries. □

In fact, even more can be obtained as the following proposition assures.

Proposition 2. *Any set S of n point seeds has a biconnected CCG. Such a CCG can be constructed in $O(n^2 \log n)$ time using linear space.*

Proof. We first consider disks as cover elements. Let D_1, D_2, and D_3 be three congruent disks that touch each other. They delimit a pseudo-triangular shape R. Choose the three disks such that each disk D_i contains a unique point $p_i \in S$ and such that $S \setminus \{p_1, p_2, p_3\} \subset R$, see Figure 2 (left).

In order to cover the remaining points we assume that disks $D_4, \ldots, D_{i-1}$ have been placed such that each covers a unique point of S and touches two previously placed disks, see Figure 2 (middle). Thus the contact graph of $D_1, \ldots, D_{i-1}$ is biconnected. Let R_j be a connected component of $R \setminus \bigcup_{j=4}^{i-1} D_i$ that contains at least one uncovered point. Use Fortune's sweep [6] to compute the combined Voronoi diagram of the disks incident to R_j and the points in $S \cap R_j$. This takes $O(n \log n)$ time and the resulting Voronoi diagram has complexity $O(n)$. The part of the Voronoi diagram in R_j is the locus of the centers of all disks that lie in R_j and touch $\partial R_j \cup (S \cap R_j)$ in at least two points, where ∂R_j is the boundary of R_j. Now we make a simple but crucial observation: if D is a disk that (a) lies

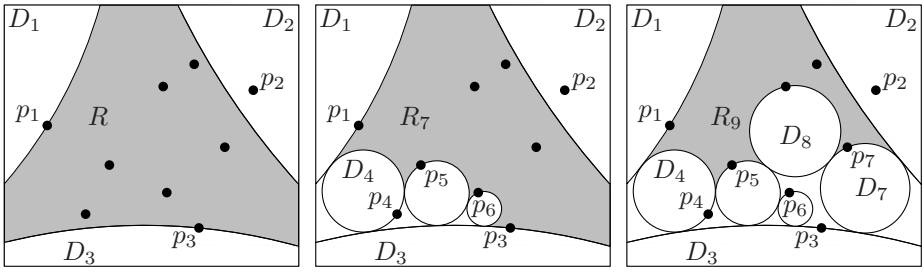


Fig. 2. Three steps in the construction of a biconnected disk-CCG

in R_j, (b) contains a seed $s \in S \cap R_j$ on its boundary, and (c) touches two of the previous disks, then D is centered at a *vertex* of the Voronoi diagram. Thus a disk $D^\star$ fulfilling (a)–(c) can be found in linear time and, by construction, does not contain any point of S in its interior. (If by any chance all such disks touch more than one point of S, we re-start the whole computation with three slightly wiggled initial disks D_1, D_2, and D_3. Then the probability of this degeneracy becomes 0.) Now set $D_i = D^\star$, and repeat the process until all seeds are covered. This takes $O(n^2 \log n)$ time in total.

The case of triangles can be handled analogously. Choosing any reference point in the triangular shape, a structure similar to the medial axis can be computed in $O(n \log n)$ and updated in $O(n)$ time in each of the $n - 3$ phases. □

2.2 Realizability

In this section we first give two necessary conditions that a planar graph must fulfill in order to be realizable as a disk-CCG on a given seed set. Then we construct a plane geometric graphs on six vertices that cannot be represented as disk-CCG. Finally we investigate the complexity of deciding realizability.

To formulate our necessary conditions for realizability we define a graph on the given seed set S. Our graph is inspired by the sphere-of-influence graph defined by Toussaint [14]. Given a seed set S and a point $p \in S$ let the *influence area* of p be the closure of the union of all empty open disks D (i.e., $D \cap S = \emptyset$) that are centered at vertices of the Voronoi region of p, see Figure 3. We call the intersection graph of these influence areas the *hyperinfluence graph* of S and denote it by $HI(S)$, see Figure 4.

Proposition 3. *Let S be a set of point seeds and let G be a graph realizable as a disk-CCG on S. Then*

(i) G is a subgraph of $HI(S)$, and
(ii) G has a plane drawing where each vertex is mapped to a unique point in S and each edge is drawn as a polygonal line with at most two segments (i.e., with at most one bend per edge).

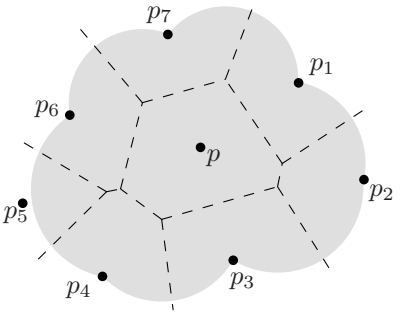


Fig. 3. Influence area of $p \in S$ (shaded)

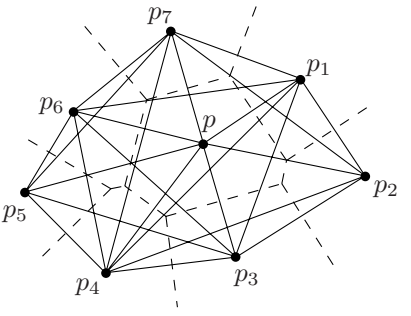


Fig. 4. The hyperinfluence graph $HI(S)$

Proof. Both facts are straightforward to obtain. (i) is based on the observation that any possible covering disk of p is contained in the influence area of p. Thus, if the covering disks of two seeds are in contact, their influence areas intersect.

(ii) is obtained by representing each edge of the CCG by two line segments that connect the seeds with the point of tangency of the covering disks. □

While Proposition 3 (ii) is difficult to verify even if all seeds lie on a line [8], Proposition 3 (i) gives us a way to show non-realizability of certain geometric graphs as the one depicted in Figure 5. That graph is connected and thus cannot be realized as a CCG with its vertices as seeds, because the shaded influence areas of p_1 and p_2 do not intersect. The graph has eight vertices. On the other hand it is easy to see that any three-vertex graph can be realized on any three-point seed set. Now it is interesting to ask for the least n for which there is an n-vertex geometric graph G such that the straight-line drawing of G is plane but G cannot be realized as CCG.

We show that there is a set $S = \{a, b, \ldots, f\}$ of six points in convex position such that their Delaunay triangulation is not representable as a CCG, see the underlying graph in Figure 6. The covering disks D_a and D_d of the points a and d must touch each other in one of two ways. Either the tangent point of the disks lies inside the convex hull of S, or D_a and D_d are very large and lie to the left of a and to the right of d, in which case they touch far above or below S, see Figure 6. In the first case there is no disk covering c and touching D_a. In the second case we can assume that the boundaries of D_a and D_d are two almost parallel lines in the vicinity of the six points. The disks D_c and D_f covering c and f must both touch D_a and D_d. But if c and f are close enough to a and d then D_c and D_f cannot be disjoint.

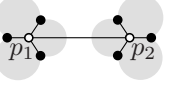


Fig. 5. Non-realizable bipartite graph

So we have seen that there are pairs of (quite small) graphs and seed sets such that the graph cannot be realized on the seed set as disk CCG. Thus we would like to decide whether a given graph is realizable as CCG on a given seed set or not. Of course Koebe's theorem [9] guarantees that for any planar graph G we can find a seed set S such that it is possible to realize G on S. However, if

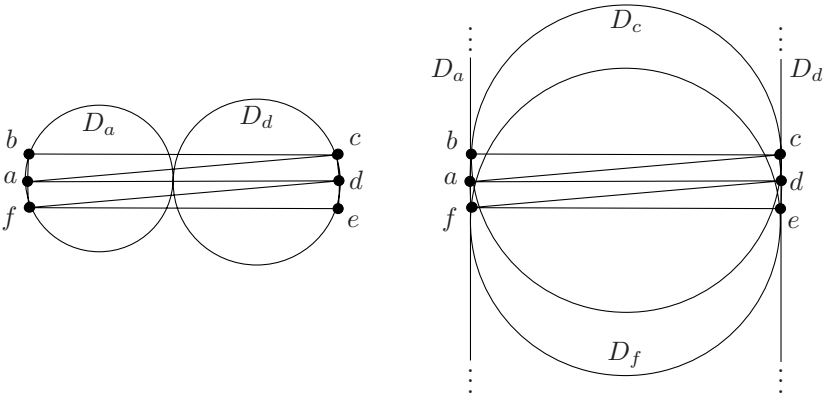


Fig. 6. Non-realizable Delaunay triangulation of six points in convex position

the seeds and the vertex–seed correspondence are given, the problem becomes NP-hard.

Theorem 1. *Given a set S of points in the plane and a planar graph $G = (S, E)$, it is NP-hard to decide whether G is realizable as disk-CCG on S.*

The proof is by reduction from the NP-hard problem Planar3SAT. There are gadgets for each variable and each clause of the given Boolean formula. The gadget of a variable v is such that it allows two combinatorially different ways to represent the given subgraph as disk-CCG. These correspond to the two Boolean values of v. The clause gadget is locally symmetric with respect to 120°-rotations and designed such that some cover disks must overlap if and only if the corresponding three literals are all false.

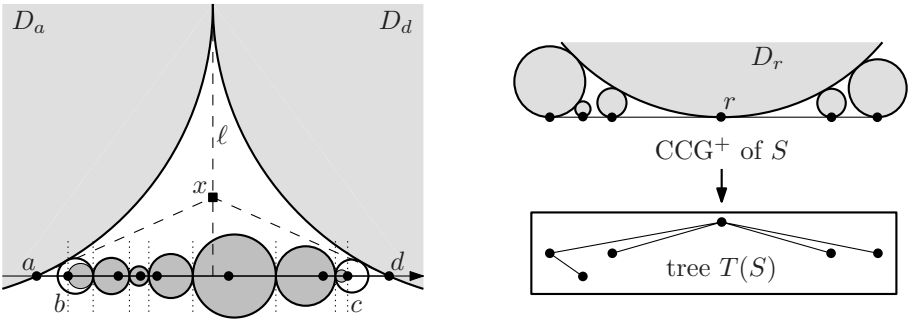
3 The Seeds Are Points on a Line

In this section, seed sets consist of points on the x-axis. Connectivity follows from some of our realizability results, so we focus on the latter. We consider the following four questions. Note that seeds now correspond to real numbers, so we can use the natural order $<$ in $\mathbb{R}$ to compare them. All covers consist of disks unless stated otherwise (e.g., in Q4).

Q1. Given a graph class $\mathcal{C}$ (e.g., the class of trees), does it hold that for any seed set S there is a graph in $\mathcal{C}$ that is realizable as CCG or CCG$^+$ on S?
We show: This is true for (cycles, CCG) and (trees, CCG$^+$).

Q2. Given a graph class $\mathcal{C}$, does it hold that for any graph G in $\mathcal{C}$ there is a seed set S such that G can be realized as CCG or CCG$^+$ on S?
We show: This is true for the combination (trees, CCG$^+$).

Q3. Let $\mathcal{C}$ be a fixed graph class. Given a graph $G \in \mathcal{C}$ with a labeling $\lambda : V \to \{1, \ldots, n\}$, is there a sequence $s_1 < \ldots < s_n$ of seeds in $\mathbb{R}^1$ and a realization of G that maps each vertex v to the corresponding seed $s_{\lambda(v)}$?
We show: There is an $O(n \log n)$ decision algorithm for (trees, CCG$^+$).



(a) C_n is realizable as CCG **(b)** tree $T(S)$ is realizable as CCG$^+$

Fig. 7. Graphs that can be realized on a given one-dimensional n-point seed set S

Q4. Let $\mathcal{C}$ be a fixed graph class. Given a seed set S and a graph $G(S, E) \in \mathcal{C}$, can G be realized on S as triangle CCG or CCG$^+$?
We show: There is an $O(n \log n)$-time decision algorithm for (trees, CCG$^+$).

Note that the above questions require more and more concrete information about the seed set, ranging from no information (Q2) via a fixed order (Q3) to complete information (Q4). We start with question Q1.

Proposition 4. *Let S be a set of n point seeds on a line, then*

(i) the n-vertex cycle C_n can be realized as CCG on S, and
(ii) there is a tree $T(S)$ that can be realized as CCG$^+$ on S.

Figures 7a and 7b give some intuition about how our algorithms work; for details see the long version of this paper [3].

In terms of this paper, a coin graph is obtained when seeds are points and cover elements are disks centered at seeds, and thus Koebe's theorem establishes that it is always possible to choose seeds in the plane such that any given plane graph is realizable as a coin graph on them. We have seen in Proposition 4 that C_n is realizable as a CCG on any seed set on a line. One can ask whether a Koebe-type theorem also holds in this restricted setting. However, Kaufmann and Wiese [7] have shown that there is a plane triangulated 12-vertex graph (see Figure 8) that cannot be drawn with only one bend per edge if vertices are restricted to a line. Now Proposition 3 (ii) implies that that graph is not realizable as CCG if seeds lie on a line. On the positive side, we can show that a Koebe-type theorem holds for the combination (trees, CCG$^+$). This is an answer to Q2 and in a way dual to Proposition 4 (ii). See Figure 9 for a sketch of our recursive construction.

Proposition 5. *For any tree T there is a seed set $S(T) \subset \mathbb{R}^1$ such that T is realizable as CCG$^+$ on $S(T)$.*

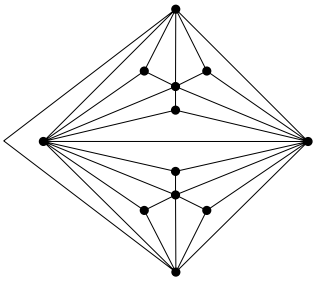


Fig. 8. Kaufmann–Wiese graph [8]

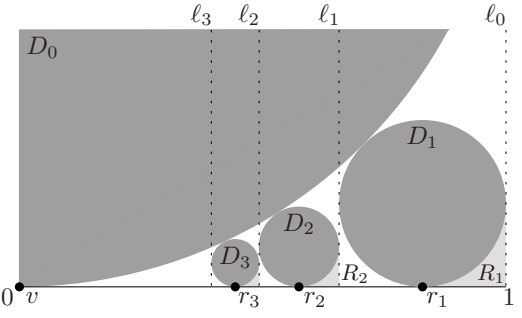


Fig. 9. Constructing a seed set $S(T)$

In Proposition 5 above, we had complete freedom to choose the seeds. Now we turn to question Q3, where we are not just given a tree, but also an order of its vertices that must be respected by the corresponding seeds. Kaufmann and Wiese [7] have investigated a related problem. They showed that it is NP-complete to decide whether the vertices of a given (planar) graph can be put into one-to-one correspondence with a given set of points on a line such that there is a plane drawing of the graph with at most one bend per edge. We call such a drawing a 1d-1BD. If additionally all bends lie on one side of the line, we call the drawing a 1d-1BD$^+$.

Note that the hardness result of Kaufmann and Wiese does not yield the hardness of the one-dimensional CCG realizability problem, since not every graph that can be one-bend embedded on a set of points on a line is realizable as CCG, let alone as CCG$^+$. Our next result explores the gap between Kaufmann and Wiese's one-dimensional embeddability problem and the situation in Proposition 5.

More formally, given an n-vertex tree T and a (bijective) labeling $\lambda : V \to \{1, \ldots, n\}$ of its vertices, we say that T is λ-*realizable* (as CCG, CCG$^+$, 1d-1BD, 1d-1BD$^+$) if there is a sequence $s_1 < \ldots < s_n$ of seeds in $\mathbb{R}^1$ and a realization of T (as CCG, CCG$^+$, 1d-1BD, 1d-1BD$^+$) that maps each vertex v to the corresponding seed $s_{\lambda(v)}$.

In order to obtain a characterization of trees that are λ-realizable as CCG$^+$, we need the following definition. Given a graph $G = (V, E)$ with vertex labeling λ, a *forbidden pair* is a pair of edges $\big\{\{a, b\}, \{c, d\}\big\}$ such that $\lambda(a) < \lambda(c) < \lambda(b) < \lambda(d)$. Note that it is impossible to embed the edges of a forbidden pair simultaneously above the x-axis.

Theorem 2. *For a λ-labeled tree T the following statements are equivalent:*

(i) T is λ-realizable as a CCG$^+$.
(ii) T is λ-realizable as a 1d-1BD$^+$.
(iii) T does not contain any forbidden pair.

Given the tree, statement (iii) can be checked in $O(n \log n)$ time using an interval tree, therefore the following corollary is straightforward.

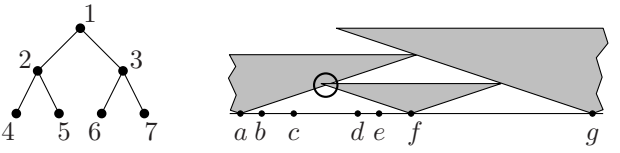


Fig. 10. Binary tree not realizable as CCG$^+$ on given seeds

Corollary 1. *Given a λ-labeled tree T, we can decide in $O(n \log n)$ time whether T is λ-realizable as CCG$^+$.*

We now turn to question Q4. So given a set of seeds S and a tree $T(S, E)$ our answer is a decision algorithm for the realizability of T as a triangle CCG$^+$ on S. Note that in our series of results about realizability we have required more and more concrete information about the seed set, ranging from no information (Proposition 5) via a fixed order (Theorem 2) to complete information now. We call a triangle *V-shaped* if it is symmetric to a vertical line and if its bottommost vertex is unique. In the following we will consider all triangles as V-shaped. First note that there are trees T and seed sets S for which the answer to question Q4 is negative even if the mapping between vertices and seeds is not fixed in advance. Figure 10 shows a complete binary tree T on seven vertices and the one-dimensional point set $S = \{a(0), b(2), c(5), d(11), e(13), f(16), g(33)\}$. A case distinction on the seed that represents the root vertex 1 shows that it is not possible to find a representation of T as a triangle CCG$^+$ on S. The example in Figure 10 shows the case where seed g represents the root. In this case any two covers of points in $S \setminus \{g\}$ that touch the cover of g will overlap, e.g., the covers of a and f.

On the other hand, there is always a tree that can be realized on a given set of seeds as Proposition 4 (ii) shows. We can give an algorithm that decides this realizability for a pair (S, T) with $T = (S, E)$ in $O(n \log n)$ time, where $n = |S|$.

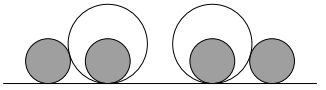
Theorem 3. *Given a set of seeds S and a tree $T = (S, E)$ we can decide in $O(n \log n)$ time whether T can be realized as a V-shaped triangle CCG$^+$ on S.*

The decision algorithm is based on the observation that the covers for the closest pair of seeds must touch each other as otherwise this CCG$^+$ would not be connected. Thus the algorithm adds the edge between the closest pair of seeds, removes one of the two seeds, and continues this process as long as it complies with T. We can use the same algorithm to generate *all* trees that can be realized as CCG$^+$ on S by branching on the seed to remove in each iteration.

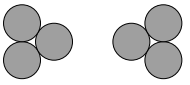
Although Theorem 3 is stated for a very restricted class of triangles, the result can easily be extended to homothetic triangles whose top sides are parallel to the x-axis.

4 The Seeds Are Disks in the Plane

In this section, we consider disks in the plane as seeds and cover them using disks, too. In contrast to point seeds the minimal size of each cover element is



(a) seed set without connected CCG$^+$



(b) seed set without connected CCG

Fig. 11. Disk seed sets without connected disk covers

now bounded from below by the size of the corresponding seed. Therefore the results in this section differ a lot from those obtained in previous sections.

Unlike the connectivity results for points we can neither guarantee the existence of a connected CCG$^+$ for disk seeds touching a line nor the existence of a connected CCG for disk seeds in the plane, see Figure 11. Deciding whether a given set of disk seeds has a connected CCG turns out to be hard.

Theorem 4. *Given a set S of disk seeds, it is NP-hard to decide whether there is a connected CCG on S, even if there are only four different seed radii.*

The proof is again by reduction from Planar3SAT. The main trick is to use what we call a *stopper element*, a cluster of three congruent pairwise touching disks as in Figure 11b. Observe that these disks can only be covered by themselves—any larger cover of any disk would intersect the others. We use small copies of these stopper elements to discretize the way in which other seeds can be covered. In the center of our clause gadget there is stopper element that is connected to the remaining cover as long as any of the corresponding three literals is true.

Concerning realizability, the hardness result of Theorem 1 clearly still holds for disk seeds. The necessary conditions for realizability in Proposition 3 can be adapted to the case of disk seeds.

5 Open Problems

This paper has opened a new field with many interesting questions.

1. We know that every 3-vertex graph can be represented as CCG on any set of three points. We have given an example of six points whose Delaunay triangulation is not representable as a CCG. What about plane geometric graphs with four or five vertices? Do they always have a representation?
2. Does any set of point seeds in convex position have a triangulation that can be represented as CCG?
3. We know that any set of point seeds has a 2-connected CCG. What about 3-connectivity?
4. Is it NP-hard to decide whether a set of disks touching a line has a connected CCG$^+$?
5. Is there an equivalent to Theorem 2 for CCG's, i.e., can we characterize vertex-labeled trees that have a realization as CCG on a set of seeds on a line which respect the vertex order prescribed by the labeling?
6. What about other classes of seeds and covers?

References

1. Abellanas, M., Bereg, S., Hurtado, F., Olaverri, A.G., Rappaport, D., Tejel, J.: Moving coins. Comput. Geom. Theory Appl. 34(1), 35–48 (2006)
2. Abellanas, M., de Castro, N., Hernández, G., Márquez, A., Moreno-Jiménez, C.: Gear system graphs. Manuscript (2006)
3. Atienza, N., de Castro, N., Cortés, C., Garrido, M. Á., Grima, C.I., Hernández, G., Márquez, A., Moreno, A., Nöllenburg, M., Portillo, J.R., Reyes, P., Valenzuela, J., Villar, M.T., Wolff, A.: Cover contact graphs. Technical Report 2007-18, Fakultät für Informatik, Universität Karlsruhe (September 2007), http://www.ubka.uni-karlsruhe.de/indexer-vvv/ira/2007/18
4. Collins, C.R., Stephenson, K.: A circle packing algorithm. Comput. Geom. Theory Appl. 25(3), 233–256 (2003)
5. de Fraysseix, H., de Mendez, P.O.: Representations by contact and intersection of segments. Algorithmica 47(4), 453–463 (2007)
6. Fortune, S.: A sweepline algorithm for Voronoi diagrams. In: SoCG 1986. Proc. 2nd Annu. Sympos. Comput. Geom., pp. 313–322 (1986)
7. Giménez, O., Noy, M.: The number of planar graphs and properties of random planar graphs. In: Martínez, C. (ed.) ICAA 2005. Proc. Internat. Conf. Anal. Algorithms, DMTCS Proceedings, vol. AD, pp. 147–156 (2005)
8. Kaufmann, M., Wiese, R.: Embedding vertices at points: Few bends suffice for planar graphs. J. Graph Algorithms Appl. 6(1), 115–129 (2002)
9. Koebe, P.: Kontaktprobleme der konformen Abbildung. Ber. Sächs. Akad. Wiss. Leipzig, Math.-Phys. Klasse 88, 141–164 (1936)
10. Mitchell, J.S.B.: L_1 shortest paths among polygonal obstacles in the plane. Algorithmica 8, 55–88 (1992)
11. Pach, J., Agarwal, P.K.: Combinatorial Geometry. John Wiley and Sons, New York (1995) (contains a proof of Koebe's theorem)
12. Sachs, H.: Coin graphs, polyhedra, and conformal mapping. Discrete Math. 134(1-3), 133–138 (1994)
13. Thurston, W.P.: The Geometry and Topology of 3-Manifolds. Princeton University Notes, Princeton (1980)
14. Toussaint, G.T.: A graph-theoretical primal sketch. In: Toussaint, G.T. (ed.) Computational Morphology: A Computational Geometric Approach to the Analysis of Form, North-Holland, pp. 229–260 (1988)
15. Welzl, E.: Smallest enclosing disks (balls and ellipsoids). In: Maurer, H.A. (ed.) New Results and New Trends in Computer Science. LNCS, vol. 555, pp. 359–370. Springer, Heidelberg (1991)

Matched Drawings of Planar Graphs*

Emilio Di Giacomo[1], Walter Didimo[1], Marc van Kreveld[2], Giuseppe Liotta[1], and Bettina Speckmann[3]

[1] Dip. di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia
{digiacomo,didimo,liotta}@diei.unipg.it
[2] Department of Computer Science, Utrecht University
marc@cs.uu.nl
[3] Department of Mathematics and Computer Science, TU Eindhoven
speckman@win.tue.nl

Abstract. A natural way to draw two planar graphs whose vertex sets are matched is to assign each matched pair a unique y-coordinate. In this paper we introduce the concept of such matched drawings, which are a relaxation of simultaneous geometric embeddings with mapping. We study which classes of graphs allow matched drawings and show that (i) two 3-connected planar graphs or a 3-connected planar graph and a tree may not be matched drawable, while (ii) two trees or a planar graph and a planar graph of some special families—such as unlabeled level planar (ULP) graphs or the family of "carousel graphs"—are always matched drawable.

1 Introduction

The visual comparison of two graphs whose vertex sets are associated in some way requires drawings of these graphs that highlight their association in a clear manner. Drawings of this type are of use for various areas of computer science, including bio-informatics, web data mining, network analysis, and software engineering. Of course each drawing individually should be as clear as possible, using, for example, few bends and crossings. But, most importantly, the positions of associated vertices in the two drawings should be "close". This makes it possible for the user to easily identify structurally identical and structurally different portions of the two graphs, or to maintain her "mental map" [17]. Structural changes between two graphs and their visualizations arise, for example, when collapsing or expanding clusters in clustered drawings, during the navigation of very large graphs with a topological window, in the analysis of the evolving relationships among the actors of a social network, and in the comparison of multiple gene trees (see, for example, [1,6,7,11,14,16,18]).

Two positions are definitely "close" if they are identical. Hence a substantial research effort has recently been devoted to the problem of computing straight-line drawings of two graphs on the same set of points. More specifically, assume

* Research partially supported by the MIUR Project "MAINSTREAM: Algorithms for massive information structures and data streams".

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 183–194, 2007.

we are given two planar graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $|V_1| = |V_2|$, together with a one-to-one mapping between their vertices. A *simultaneous geometric embedding with mapping* (introduced by Brass et al. in [3]) of G_1 and G_2 is a pair of straight-line planar drawings Γ_1 and Γ_2 of G_1 and G_2, respectively, such that for any pair of matched vertices $u \in V_1$ and $v \in V_2$ the position of u in Γ_1 is the same as the position of v in Γ_2. Unfortunately, only pairs of graphs belonging to restricted subclasses of planar graphs admit a simultaneous geometric embedding with mapping. Brass et al. [3] showed how to simultaneously embed pairs of paths, pairs of cycles, and pairs of caterpillars, but they also proved that a path and a graph or two outerplanar graphs may not admit this type of drawing. Geyer, Kaufmann, and Vrt'o [15] recently proved that even a pair of trees may not have a simultaneous geometric embedding with mapping. These negative results motivated the study of relaxations of simultaneous geometric embeddings. One possibility is to introduce bends along the edges [4,8,9,13], another, to allow that the same vertex occupies different locations in the two drawings [2,3], introducing ambiguity in the mapping.

In this paper we consider a different interpretation of two positions being "close". Instead of requiring that matched vertices occupy the same location, we assign each matched pair a unique y-coordinate. This enables the user to unambiguously identify pairs of matched vertices but, at the same time, leaves us more freedom to draw both graphs clearly. Specifically, let again $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two planar graphs with $|V_1| = |V_2|$. G_1 and G_2 are *matched* if there is a one-to-one mapping between V_1 and V_2. If a vertex $u \in V_1$ is matched with a vertex $v \in V_2$ then we say that u is the *partner* of v and that v is the partner of u. A *matched drawing* of G_1 and G_2 is a pair of straight-line planar drawings Γ_1 and Γ_2 of G_1 and G_2, respectively, such that for any pair of matched vertices $u \in V_1$ and $v \in V_2$ the y-coordinate of u in Γ_1 is the same as the y-coordinate of v in Γ_2, and this y-coordinate is unique. If two matched graphs have a matched drawing, then we say that they are *matched drawable*. Matched drawings can be viewed as a relaxation of simultaneous geometric embedding with mapping. An example of a matched drawing of two trees is shown in Fig. 1.

Results and Organization. We start by presenting pairs of graphs that are not matched drawable. In particular, in Section 2.1 we describe two isomorphic 3-connected planar graphs that both have 12 vertices and that are not matched

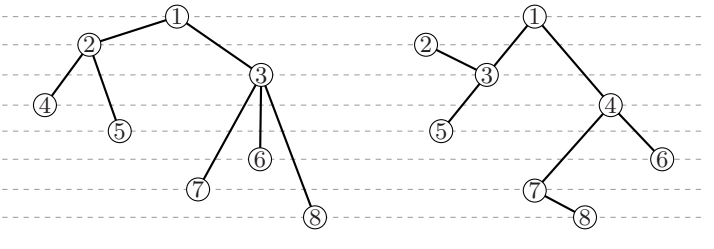


Fig. 1. A matched drawing of two trees

drawable. We also present a 3-connected planar graph and a tree that both have 620 vertices and that are not matched drawable. This construction can be found in Section 2.2.

We continue by describing drawing algorithms for classes of graphs that are always matched drawable. In particular, in Section 3.1 we show that a planar graph and an unlabeled level planar (ULP) graph that are matched are always matched drawable. In Section 3.2 we extend these results to a planar graph and a graph of the family of "carousel graphs". Finally, in Section 3.3 we prove that two matched trees are always matched drawable.

2 Graphs That Are Not Matched Drawable

2.1 Two 3-Connected Graphs

We start by stating a simple property of planar straight-line drawings.

Property 1. Let G be an embedded planar graph and let Γ be a straight-line planar drawing of G. Let u be the vertex of G with the highest y-coordinate in Γ and let v be the vertex of G with the lowest y-coordinate in Γ. Vertices u and v belong to the external face of G.

Now assume that G_1 and G_2 are two matched graphs with the following properties: (i) G_1 contains two vertex-disjoint simple cycles $C_1 = \{u_1, \ldots, u_n\}$ and $C'_1 = \{u'_1, \ldots, u'_m\}$, (ii) G_2 contains two vertex-disjoint simple cycles $C_2 = \{v_1, \ldots, v_n\}$ and $C'_2 = \{v'_1, \ldots, v'_m\}$, and (iii) u_i is the partner of v_i $(1 \leq i \leq n)$ and u'_j is the partner of v'_j $(1 \leq j \leq m)$. If Ψ_1 is a planar embedding of G_1 such that C'_1 is inside C_1 and if Ψ_2 is a planar embedding of G_2 such that C_2 is inside C'_2, then we call Ψ_1 and Ψ_2 *interlaced embeddings* and C_1, C'_1, C_2, and C'_2 *interlaced cycles*.

Lemma 1. *Let G_1 and G_2 be two matched graphs with interlaced embeddings Ψ_1 and Ψ_2. There is no matched drawing Γ_1 and Γ_2 of G_1 and G_2 such that Γ_1 preserves Ψ_1 and Γ_2 preserves Ψ_2.*

Proof. Denote by C_1, C'_1, C_2, and C'_2 the interlaced cycles of Ψ_1 and Ψ_2. Suppose by contradiction that Γ_1 and Γ_2 exist. Denote by $\overline{\Gamma_1}$ the subdrawing of Γ_1 restricted to the subgraph $C_1 \cup C'_1$ and by $\overline{\Gamma_2}$ the subdrawing of Γ_2 restricted to the subgraph $C_2 \cup C'_2$.

Since in Ψ_1 cycle C'_1 is inside cycle C_1, by Property 1 the top-most and the bottom-most vertices of $\overline{\Gamma_1}$ belong to C_1; denote these two vertices by u_t and u_b. Since $\overline{\Gamma_1}$ is planar and since the drawing of C'_1 is completely inside the drawing of C_1, every vertex u'_j of C'_1 has a y-coordinate that is greater than the y-coordinate of u_b and smaller than the y-coordinate of u_t. Since Γ_1 and Γ_2 are matched drawings, every vertex v'_j of C'_2 in $\overline{\Gamma_2}$ has a y-coordinate that is greater than the y-coordinate of v_b (i.e., the partner of u_b) and smaller than the y-coordinate of v_t (i.e., the partner of u_t). However, since in Ψ_2 cycle C_2 is inside cycle C'_2, by Property 1 the top-most and the bottom-most vertices of $\overline{\Gamma_2}$ belong to C'_2, a contradiction. □

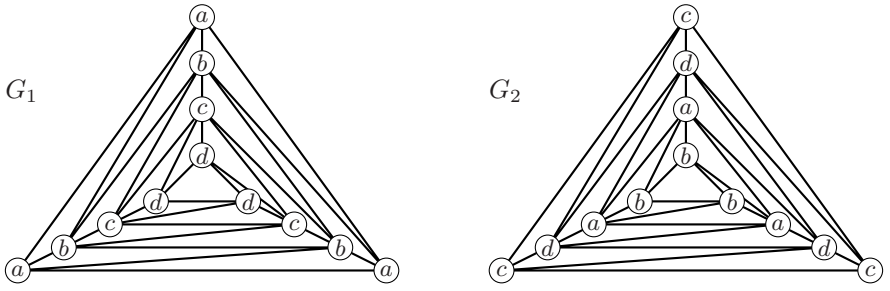


Fig. 2. Two 3-connected planar graphs that are not matched drawable. The partner of a vertex of G_1 is any vertex in G_2 that has the same label.

Theorem 1. *There exist two 3-connected planar graphs that are not matched drawable.*

Proof (sketch). Consider the two 3-connected planar graphs G_1 and G_2 in Fig. 2. The partner of a vertex of G_1 is any vertex in G_2 that has the same label. To prove that G_1 and G_2 are not matched drawable, we show that all planar embeddings of G_1 and G_2 are interlaced embeddings. The proof uses a case analysis on the choice of the external faces and is omitted for reasons of space. □

2.2 A 3-Connected Graph and a Tree

The two graphs described in Theorem 1 are both 3-connected. Hence the question arises if two planar graphs, at least one of which is not 3-connected, are always matched drawable. This is unfortunately not the case: in the following we present a planar graph and a tree that are not matched drawable.

Given a vertex v of a graph G and a drawing Γ of G, we denote by $x(v)$ and $y(v)$ the x- and y-coordinate of v in Γ. Let $T^* = (V^*, E^*)$ be the tree depicted in Fig. 3. Estrella-Balderrama et al. [10] proved the following lemma:

Lemma 2 (Estrella-Balderrama et al. [10]). *Let T^* be the tree depicted in Fig. 3. A straight-line planar drawing Γ of T^* such that $y(v_0) < y(v_7) < y(v_3) < y(v_2) < y(v_4) < y(v_1) < y(v_5) < y(v_6)$ in Γ does not exist.*

Let T^* be rooted at vertex v_0, and for each vertex v_i, denote by $d(v_i)$ the graph-theoretic distance of v_i from the root ($i = 0, 1, \ldots, 7$). We construct a tree T by using T^* as a model. T has $3^{d(v_i)}$ copies of each vertex v_i ($i = 0, 1, \ldots, 7$). The $3^{d(v_i)}$ copies of v_i are denoted as $v_{i,0}, v_{i,1}, \ldots, v_{i,3^{d(v_i)}-1}$. Vertex $v_{h,k}$ is a child of vertex $v_{i,j}$ in T if and only if v_h is a child of v_i in T^* and $j = \lfloor k/3 \rfloor$ ($0 \leq i, h \leq 7$), ($0 \leq j \leq 3^{d(v_i)} - 1$), ($0 \leq k \leq 3^{d(v_h)} - 1$). So T has one copy of v_0 whose children are the three copies $v_{1,0}$, $v_{1,1}$, and $v_{1,2}$ of v_1. The children of each copy of v_1 are three of the nine copies of v_2, and so on. Three vertices of T with the same parent are called a *triplet* of T. The total number of vertices of T is 310.

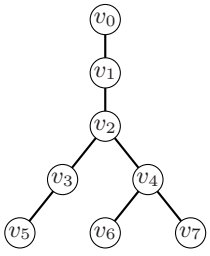


Fig. 3. A tree that does not have a straight-line planar drawing with $y(v_0) < y(v_7) < y(v_3) < y(v_2) < y(v_4) < y(v_1) < y(v_5) < y(v_6)$ [10]

Table 1. Matching between the vertices of T and the vertices of G_{103}

vertex	*copies*	*triplets*	*levels*
v_7	81	27	1 . . . 27
v_3	27	9	28 . . . 36
v_2	9	3	37 . . . 39
v_4	27	9	40 . . . 48
v_1	3	1	49
v_5	81	27	50 . . . 76
v_6	81	27	77 . . . 103

The tree T is matched with a *nested-triangles graph*, which is defined as follows. A single vertex v is a nested-triangles graph denoted by G_0. A triangulated planar embedded graph G_k $(k > 0)$ is a nested-triangles graph if the external face of G_k has exactly three vertices and the graph G_{k-1}, obtained by removing the vertices on the external face, is still a nested-triangles graph. A levelling of the vertices is naturally defined for the vertices of G_k: level i of G_k contains the vertices that are on the external face of G_i $(i = 0, 1, \ldots, k)$. Note that G_k has $3k + 1$ vertices and $k + 1$ levels. Thus, G_{103} has 310 vertices and 104 levels.

T and G_{103} are matched in the following way. Vertex v_0 is mapped to the (only) vertex of level 0. Each triplet of T is mapped to three vertices of G_{103} such that the level of these three vertices is the same in G_{103}. Also, all triplets formed by vertices that are copies of the same vertex of T^* are mapped to consecutive levels of G_{103}. The exact mapping is described in Table 1. Each row of the table refers to a different vertex of T^* and shows the number of copies of that vertex in T, the number of triplets in T, and the levels of G_{103} to which these triplets are mapped (a triplet for each level).

We now prove that, with the mapping described by Table 1, T and G_{103} are not matched drawable if we insist that the drawing of G_{103} preserves the embedding of G_{103}. We start with a useful property.

Property 2. Let $\Gamma_{G_{103}}$ be any planar straight-line drawing of G_{103} that preserves the embedding of G_{103}. For each level i $(0 \leq i \leq 103)$ there exists a vertex of level i that has y-coordinate greater than the y-coordinates of all the vertices having level less than i.

Lemma 3. *A matched drawing Γ_T and $\Gamma_{G_{103}}$ of the tree T and the graph G_{103} such that $\Gamma_{G_{103}}$ preserves the embedding of G_{103} does not exist.*

Proof (sketch). Let $\Gamma_{G_{103}}$ be any planar straight-line drawing of G_{103} that preserves the embedding of G_{103}. By exploiting Property 2, we can show that $\Gamma_{G_{103}}$ induces an ordering λ of the vertices of T along the y-direction such that there exists a subtree T' of T isomorphic to T^* for which the ordering λ restricted to the vertices of T' is the ordering given in Lemma 2 (the proof about how T' is

defined is omitted). This implies that T' (and hence T) does not have a planar straight-line drawing that respects the ordering induced by $\Gamma_{G_{103}}$. □

According to Lemma 3, T and G_{103} are not matched drawable in the case that one wants a drawing of G_{103} that preserves the embedding of G_{103}. In the following theorem we show that T and G_{103} can be used to construct a new tree and a new 3-connected planar graph that are not matched drawable even if we allow the embedding to be changed.

Theorem 2. *There exist a tree and a 3-connected planar graph that are not matched drawable.*

Proof (sketch). Let $\overline{T}$ be a tree that consists of two copies of T whose roots are adjacent. Let G be a graph obtained by taking two distinct copies of G_{103} and connecting the vertices of their external faces in such a way that the obtained graph is a triangulated planar graph. The matching of the vertices is such that a copy of T matches a copy of G_{103} as before. We observe that any embedding of G leaves one of the copies of G_{103} as in Lemma 3. □

3 Matched Drawable Graphs

In this section we describe drawing algorithms for classes of graphs that are always matched drawable. In particular, in Section 3.1 we show that a planar graph and an unlabeled level planar (ULP) graph that are matched are always matched drawable. In Section 3.2 we extend these results to a planar graph and a graph of the family of "carousel graphs". Finally, in Section 3.3 we prove that two matched trees are always matched drawable.

These results show that matched drawings do indeed allow larger classes of graphs to be drawn than simultaneous geometric embeddings with mapping (a path and a planar graph may not admit a simultaneous geometric embedding with mapping [3] and the same negative result also holds for pairs of trees [15]).

3.1 Planar Graphs and ULP Graphs

ULP graphs were defined by Estrella-Balderrama, Fowler, and Kobourov [10]. Let G be a planar graph with n vertices. A *y-assignment* of the vertices of G is a one-to-one mapping $\lambda : V \to \mathbb{N}$. A *drawing of G compatible with λ* is a planar straight-line drawing of G such that $y(v) = \lambda(v)$ for each vertex $v \in V$. A planar graph G is *unlabeled level planar* (ULP) if for any given y-assignment λ of its vertices, G admits a drawing compatible with λ.

Theorem 3. *A planar graph and an ULP graph are always matched drawable.*

Proof (sketch). Let G_1 be a planar graph and let G_2 be an ULP graph. Compute a planar straight-line drawing of G_1 such that each vertex has a different y-coordinate, for example with a slight variant of the algorithm of de Fraysseix, Pach, and Pollack [5]. The drawing of G_1 together with the mapping between

G_1 and G_2 defines a y-assignment λ for G_2. Since G_2 is ULP it admits a drawing compatible with λ. It follows that G_1 and G_2 are matched drawable. □

ULP trees are characterized in [10]. A complete characterization of ULP graphs has very recently been given in [12]. A planar graph is ULP if and only if it is either a *generalized caterpillar*, or a *radius*-2 *star*, or a *generalized degree-3 spider*. These graphs are defined as follows (see also [12]). A graph is a *caterpillar* if deleting all vertices of degree one produces a path, which is called the *spine* of the caterpillar. A *generalized caterpillar* is a graph that contains cycles of length at most 4 in which every spanning tree is a caterpillar such that no three cut vertices are pairwise adjacent and no pair of adjacent cut vertices belong to the same 4-cycle. A *radius*-2 *star* is a $K_{1,k}$, $k > 2$, in which every edge is subdivided at most once. The only vertex of degree k is called the *center* of the star. A *degree*-3 *spider* is an arbitrary subdivision of $K_{1,3}$. A *generalized degree-3 spider* is a graph with maximum degree 3 in which every spanning tree is either a path or a degree-3 spider.

Corollary 1. *Let G_1 and G_2 be two matched graphs such that G_1 is a planar graph and G_2 is either a generalized caterpillar, or a radius-2 star, or a generalized degree-3 spider. Then G_1 and G_2 are matched drawable.*

3.2 Planar Graphs and Carousel Graphs

In this section we extend the result of Theorem 3 by describing a family of graphs that also includes non-ULP graphs and whose members have a matched drawing with any planar graph. Let G be a planar graph, let v be a vertex of G, and let Γ be a planar straight-line drawing of G. Γ is *v-stretchable* if: (i) there is a vertical ray from v going to $+\infty$ that does not intersect any edge of Γ, and (ii) for any given $\Delta > 0$, there exists a value $\Delta' \geq \Delta$ such that the drawing obtained by translating each vertex u with $x(u) \geq x(v)$ to point $(x(u) + \Delta', y(u))$ is still planar. Graph G is ULP *v-stretchable* if for every given y-assignment λ of its vertices, G admits a v-stretchable drawing compatible with λ.

A *carousel graph* is a connected planar graph G consisting of a vertex v_0, called the *pivot* of G, and of a set of disjoint subgraphs $S_1, \ldots, S_k$ $(k > 1)$ such that each S_i has a single vertex v_i adjacent to v_0 $(i = 1, \ldots, k)$ and S_i is ULP v_i-stretchable. Each subgraph S_i is called a *seat* of G. Vertex v_i is called the *hook* of S_i.

Theorem 4. *Any planar graph and any carousel graph that are matched are always matched drawable.*

Proof. Let G_1 be a planar graph and let G_2 by a carousel graph. Let v_0 be the pivot of G_2 and let u be the partner of v_0 in G_1. Compute a planar straight-line drawing of G_1 such that all vertices have different y-coordinates and u has the largest y-coordinate. The drawing of G_1 together with the mapping between G_1 and G_2 defines a y-assignment λ for G_2. Clearly $\lambda(w) < \lambda(v_0) = y_M$ for all vertices $w \neq v_0$ of G_2.

In the following we describe an incremental method to compute a drawing of G_2 compatible with λ. Let $S_1, \ldots, S_k$ ($k > 1$) be the seats of G_2 and let v_i be the hook of S_i ($1 \leq i \leq k$). Let λ_i be the y-assignment of the vertices of S_i induced by λ. As a preliminary step we compute a drawing Γ_i for each S_i that is compatible with λ_i and that is v_i-stretchable. Such a drawing exists because S_i is ULP v_i-stretchable. We further assume that the distance between any two different x-coordinates is at least 1 unit.

We initialize the drawing by placing v_0 at position $(0, y_M)$, which results in drawing Γ_2^0. Drawing Γ_2^i is constructed from drawing Γ_2^{i-1} by adding drawing Γ_i at a suitable x-location and possibly translating some of its vertices further in x-direction (see Fig. 4). Hence the final drawing respects λ.

Let $\mathcal{R}_{i-1}$ be the bounding box of Γ_2^{i-1} and let (x_M, y_M) be the coordinates of its top-right corner. Further let R_i be the bounding box of Γ_i. Place the drawing Γ_i such that the left side of R_i is contained in the vertical line $x = x_M + 1$. Let R_i' be the (possibly empty) sub-rectangle of R_i delimited by the x-coordinates $x_M + 1$ and $x'_M = x(v_i) - 1$. Further let y'_M denote the maximum y-coordinate of any vertex of Γ_2^{i-1} or Γ_i different from v_0 and let $p = (x'_M + 1, y'_M)$. The line ℓ through v_0 and p crosses neither Γ_2^{i-1} nor the portion of Γ_i contained in R_i' (see Fig. 4(a)). Let q denote the intersection of ℓ with the horizontal line at $y(v_i)$ and let $\Delta = x(q) - x(v_i)$. Since Γ_i is v_i-stretchable, there exists a value $\Delta' \geq \Delta$ such that we can translate the portion of Γ_i contained in $R_i \setminus R_i'$ to the right by Δ' without creating any crossing (see Fig. 4(b)). It can easily be verified that we can now connect v_i to v_0 without creating any crossings. □

Lemma 4. *Let G be a simple cycle and let v be any vertex of G. G is ULP v-stretchable.*

Proof. Let λ be any y-assignment of the vertices of G and let u be the vertex of G that has the smallest y-coordinate. Let $u = v_0, v_1, \ldots, v_{n-1}$ be the vertices of G in the order they are encountered when walking clockwise along G. Place each vertex v_i at point $(i, \lambda(v_i))$. Clearly none of the edges (v_i, v_{i+1}) $(i = 0, 1, \ldots, n-2)$ cross each other. To avoid crossings between edge (v_0, v_{n-1}) and the other edges we translate v_{n-1} to the right until the segment connecting v_0 to v_{n-1} does not cross any other segment. It is immediate to see that such a drawing is v-stretchable for every vertex v of G. □

Corollary 2. *Let G_1 and G_2 be two matched graphs such that G_1 is a planar graph and G_2 is a cycle. Then G_1 and G_2 are matched drawable.*

The drawing techniques in [10] imply the following two lemmata.

Lemma 5. *Let G be a caterpillar and let v be a vertex of its spine. G is ULP v-stretchable.*

Lemma 6. *Let G be a radius-2 star and let v be the center of G. G is ULP v-stretchable.*

Corollary 3. *Let G_1 and G_2 be two matched graphs such that G_1 is a planar graph and G_2 is a carousel graph. If each seat of G_2 is either a caterpillar with*

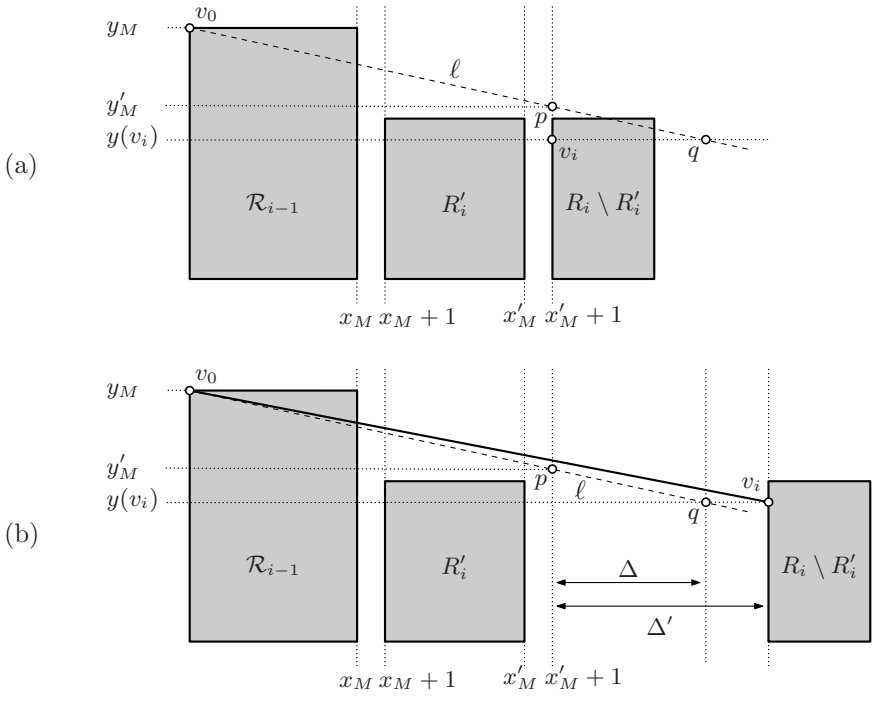


Fig. 4. Adding Γ_i to Γ_2^{i-1}

a vertex of its spine as its hook, a radius-2 star with its center as its hook, or a cycle, then G_1 and G_2 are matched drawable.

The family of carousel graphs described by Corollary 3 contains graphs that are not ULP. For example, the graph depicted in Fig. 3 is a carousel graph with pivot v_2, the three seats are caterpillars.

3.3 Two Trees

Theorem 5. *Any two matched trees are matched drawable.*

Proof. Let T_1 and T_2 any two matched trees. We prove by construction that T_1 and T_2 are matched drawable. Let the y-coordinates to be used be $1, \ldots, n$, we will assign matched vertices from T_1 and T_2 consecutively to coordinates $n, 1, n-1, 2, n-2, 3, \ldots$ until all vertices are placed.

Let T_i be a tree with a subset of its vertices placed. Then the maximal connected unplaced parts of T_i are incident to one, two, or more placed vertices. We call a maximal connected unplaced part of a tree a *chunk*.

We maintain the following invariant for T_1: after every odd placement, every chunk of T_1 is incident to at most two placed vertices of T_1. For T_2 we maintain a similar invariant: after every even placement, every chunk of T_2 is incident to at

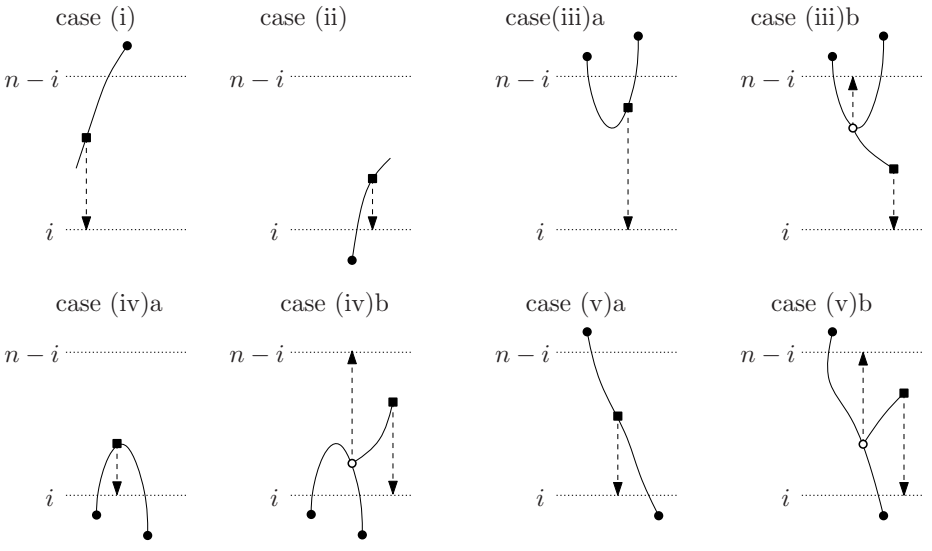


Fig. 5. The eight cases for placement at i (and at $n-i$ in three cases)

most two placed vertices of T_2. We call this the *topological invariant*. Intuitively, tree T_1 determines which vertex is placed in odd placements at n, $n-1$, $n-2$, ..., and tree T_2 determines which vertex is placed in even placements at 1, 2, 3, ... The other tree just follows with the matched vertex.

The topological invariants are needed for two reasons. Most importantly, they make sure that the algorithm cannot get stuck, in the sense that the placement of a vertex leads to an intersection. Secondly, they limit the number of cases that must be analyzed.

Consider T_1 after an odd placement and assume that it satisfies the invariant. Then a chunk can be one of five *types*: (1) it has one incident placed vertex at a high coordinate; (2) it has one incident placed vertex at a low coordinate; (3) it has two incident placed vertices at high coordinates; (4) it has two incident placed vertices at low coordinates; (5) it has one incident placed vertex at a high coordinate and one incident placed vertex at a low coordinate.

An even placement (at the bottom) may cause violation of the invariant for T_1 unless the next odd placement restores it. So for the case analysis of T_1 we will consider all possibilities of an even placement and the corresponding odd placement. If the even placement is at i, then the next odd placement is at $n-i$. There are eight cases to be distinguished for an even placement at i; they are shown in Fig. 5. In the three (.)b cases, which vertex to place at $n-i$ is determined by the fact that the topological invariant must be restored for T_1. It is the unique vertex of T_1 where the path from the just placed vertex meets the path between the two vertices that bound the chunk. It is easy to see from the figure that in the three (.)b cases the invariant can be restored for T_1 by

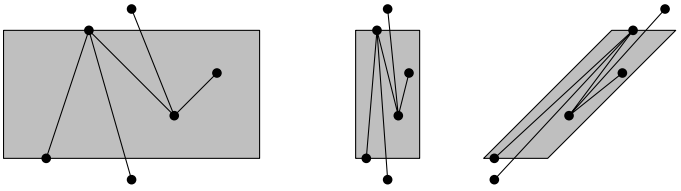


Fig. 6. Scaling and shearing a wide rectangle into a narrow parallelogram

placing this vertex at $n - i$. In the five other cases, we can assure the invariant to hold after placement at $n - i$ by choosing to place any unplaced vertex that is a neighbor of a placed vertex.

The situation is completely analogous for T_2, where an odd placement may cause a violation of the invariant if the next even placement is not chosen well.

Next we must show that there is actually space to draw the trees without crossings and with straight edges. For this we need a geometric invariant: after the placement at $n - i + 1$, there is a parallelogram between the horizontal lines at $n - i$ and i in which the whole chunk can be drawn without crossings and with straight edges. The parallelograms must have positive width and have an "alignment" that corresponds to the needs of the chunk. For example, for type (1) the incident placed vertex must be able to connect to any point on the far horizontal side of the parallelogram without going outside the parallelogram. It remains to show that every chunk can be drawn inside its parallelogram and that, if a chunk is split into several chunks, their resulting parallelograms are disjoint. In essence this is the case because any parallelogram can be scaled and sheared to fit, see Fig. 6. The formal statement of the geometric invariant and the remainder of the proof are omitted due to space limitations. □

4 Conclusions and Open Problems

In this paper we introduced the concept of matched drawings, which are a natural way to draw two planar graphs whose vertex sets are matched. Since this is the first study of these drawings, many interesting and challenging open problems remain. First of all, in the light of Theorems 2 and 4, we would like to characterize the subclass of planar graphs that admit a matched drawing with any planar graph. Secondly, the drawing techniques of Theorems 4 and 5 may give rise to drawings where the area is exponential in the size of the graphs. It would be interesting to study the area requirement of matched drawings that use straight-line edges. On a related note, some of our drawing techniques rely on a planar straight-line drawing of a planar graph where each vertex has a different y-coordinate. How big a grid is necessary to guarantee such a drawing with integer coordinates? And finally, given any two matched graphs, what is the complexity of testing whether they are matched drawable?

References

1. Brandes, U., Erlebach, T. (eds.): Network Analysis. LNCS, vol. 3418. Springer, Heidelberg (2005)
2. Brandes, U., Erten, C., Fowler, J., Frati, F., Geyer, M., Gutwenger, C., Hong, S.-H., Kaufmann, M., Kobourov, S., Liotta, G., Mutzel, P., Symvonis, A.: Colored simultaneous geometric embeddings. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 254–263. Springer, Heidelberg (2007)
3. Braß, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous planar graph embeddings. Computational Geometry: Theory and Applications 36(2), 117–130 (2007)
4. Cappos, J., Estrella-Balderrama, A., Fowler, J.J., Kobourov, S.G.: Simultaneous graph embedding with bends and circular arcs. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 95–107. Springer, Heidelberg (2007)
5. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10, 41–51 (1990)
6. Demetrescu, C., Di Battista, G., Finocchi, I., Liotta, G., Patrignani, M., Pizzonia, M.: Infinite trees and the future. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 379–391. Springer, Heidelberg (1999)
7. Di Giacomo, E., Didimo, W., Grilli, L., Liotta, G.: Graph visualization techniques for web clustering engines. IEEE Transactions on Visualization and Computer Graphics 13(2), 294–304 (2007)
8. Di Giacomo, E., Liotta, G.: Simultaneous embedding of outerplanar graphs, paths, and cycles. International Journal of Computational Geometry and Applications 17(2), 139–160 (2007)
9. Erten, C., Kobourov, S.G.: Simultaneous embedding of planar graphs with few bends. Journal of Graph Algorithms and Applications 9(3), 347–364 (2005)
10. Estrella-Balderrama, A., Fowler, J.J., Kobourov, S.G.: Characterization of unlabeled level planar trees. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 367–379. Springer, Heidelberg (2007)
11. Fernau, H., Kaufmann, M., Poths, M.: Comparing trees via crossing minimization. In: Proc. 25th Conf. on Foundations of Software Technology and Theoretical Computer Science, pp. 457–469 (2005)
12. Fowler, J.J., Kouborov, S.G.: Characterization of unlabeled level planar graphs. Technical Report TR06-04, Dep. of Computer Science, University of Arizona (2006)
13. Frati, F.: Embedding graphs simultaneously with fixed edges. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 108–113. Springer, Heidelberg (2007)
14. Friedrich, C., Eades, P.: Graph drawing in motion. Journal of Graph Algorithms and Applications 6(3), 353–370 (2002)
15. Geyer, M., Kaufmann, M., Vrt'o, I.: Two trees which are self-intersecting when drawn simultaneously. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 201–210. Springer, Heidelberg (2006)
16. Huang, M.L., Eades, P., Cohen, R.F.: WebOFDAV-navigating and visualising the web online with animated context swapping. Computer Networks and ISDN Systems 30, 638–642 (1998)
17. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. Journal of Visual Languages and Computing 6(2), 183–210 (1995)
18. North, S.: Incremental layout in dynadag. In: Brandenburg, F.J. (ed.) GD 1995. LNCS, vol. 1027, pp. 409–418. Springer, Heidelberg (1996)

Maximum Upward Planar Subgraphs of Embedded Planar Digraphs*

Carla Binucci, Walter Didimo, and Francesco Giordano

DIEI - Università degli Studi di Perugia
{binucci, didimo, giordano}@diei.unipg.it

Abstract. This paper presents an extensive study on the problem of computing maximum upward planar subgraphs of embedded planar digraphs: Complexity results, algorithms, and experiments are presented. Namely: (*i*) We prove that the addressed problem is NP-Hard; (*ii*) A fast heuristic and an exponential-time exact algorithm are described; (*iii*) A wide experimental analysis is performed to show the effectiveness of our techniques.

1 Introduction

The upward drawing convention is commonly used to display hierarchical structures so that all edges flow in a common direction according to their orientation. More precisely, let G be a directed graph (also called a *digraph*); an *upward drawing* of G is such that each edge is drawn as a simple Jordan curve monotonically increasing in the upward direction. In particular, a wide research effort has been devoted so far to the design of algorithms for computing *upward planar drawings*, i.e, upward drawings without crossings. Indeed, there is a general consensus that the number of crossings between edges is one of the most critical aesthetic requirements for the readability of a drawing. A digraph that admits an upward planar drawing is called an *upward planar digraph*. Fig. 1(a) and 1(b) show a planar digraph G and an upward planar drawing of G. The planar digraph in Fig. 1(c) is not upward planar.

Bertolazzi et al. [2] proved that if a digraph G with n vertices has a fixed planar embedding, then testing whether G admits an upward planar drawing that preserves its embedding can be done in $O(n^2)$ time. On the other side, Garg and Tamassia [10] proved that the upward planarity testing problem in the variable embedding setting (i.e., over all planar embeddings of the input digraph) is NP-Complete. In this scenario, several polynomial-time upward planarity testing algorithms have been described in the literature for specific sub-families of planar digraphs [3,8,12,14], and exponential-time algorithms for the same problem can be found in [1,4,11].

* Research partially supported by the MIUR Project "MAINSTREAM".

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 195–206, 2007.

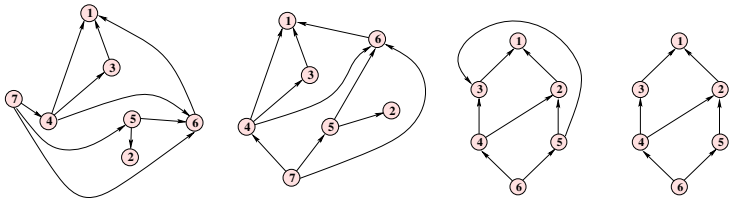


Fig. 1. (a) A planar digraph G with a given planar (bimodal) embedding. (b) An embedding preserving upward planar drawing of G. (c) A digraph G' that is not upward planar; (d) A maximum upward planar subgraph of G'.

When a planar digraph G is not upward planar, an interesting problem that naturally arises is the one of computing a *maximum upward planar subgraph* of G, i.e., an upward planar subgraph with maximum number of edges. From the application side, solving this problem is important to find large hierarchical sub-structures in the digraph and to convey them visually. In the variable embedding setting, computing a maximum upward planar subgraph is NP-Hard as an immediate consequence of the hardness of the upward planarity testing problem [10]. If the embedding of the digraph is fixed, however, the complexity of the problem is still unknown. Recall that in this case the upward planarity testing problem is polynomially solvable [2].

We present an extensive study on the problem of computing a maximum upward planar subgraph of an embedded planar digraph. Namely:

(*i*) We prove that finding a maximum upward planar subgraph of a planar digraph remains NP-Hard, even in the fixed embedding scenario (Section 3). Our proof uses a reduction from Planar 3-SAT [13]. With the same reduction we also prove that finding the maximum bimodal subgraph of an embedded planar digraph is NP-Hard. Recall that an embedded digraph is bimodal if the incoming and the outgoing edges of each vertex never alternate (see, e.g., Fig. 1(a)). Notice that the bimodality is necessary (but not sufficient) for the upward planarity.

(*ii*) Motivated by the above negative results, we describe both a polynomial-time heuristic and a branch-and-bound exact algorithm to compute a maximum upward planar subgraph of an embedded planar digraph (Section 4). The input digraph is not necessarily bimodal and acyclic. Our heuristic adopts a greedy approach for computing a large bimodal subgraph and then extracts from it an upward planar subgraph by using a combination of the techniques given in [1,2]. Notice that, in the variable embedding setting any heuristic that uses an upward planarity testing as a key tool would still require exponential time.

(*iii*) We perform a wide experimental study, which shows how our heuristic is pretty fast and effective in practice; it achieves the optimum in many cases and definitively outperforms a simple technique that incrementally tries to insert an edge per time while preserving upward planarity (Section 5).

2 Basic Definitions

We assume familiarity with basic concepts of graph planarity and graph drawing [5]. We denote by G_Φ an *embedded planar digraph*, i.e., a planar digraph G with a given planar embedding Φ, where Φ describes the set of (internal and external) faces for G in the plane. For each vertex v of G, Φ also fixes the circular clockwise ordering of the edges incident to v. An *embedding preserving subgraph* $G'_{\Phi'}$ of G_Φ is an embedded planar digraph obtained from G_Φ by removing a subset of its edges. Notice that, for each vertex v of $G'_{\Phi'}$, the circular clockwise ordering of the edges incident to v in $G'_{\Phi'}$ is the same as in G_Φ.

A vertex v of G_Φ is *bimodal* if all incoming edges of v (and hence all outgoing edges of v) appear consecutive in the circular clockwise ordering around v. If all vertices of G_Φ are bimodal, Φ is called a *planar bimodal embedding* and G_Φ is called a *planar bimodal embedded digraph*. A planar digraph G is *bimodal* if it admits a planar bimodal embedding. The digraph in Fig. 1(a) is a planar bimodal embedded digraph.

An *upward planar drawing* of G_Φ is a planar drawing of G that preserves the embedding Φ and such that all the edges of G are drawn as curves monotonically increasing in the upward direction. We say that G_Φ is *upward planar* if it admits an upward planar drawing. It is known that acyclicity and bimodality are necessary (but not sufficient) conditions for the upward planarity [2]. For example, the planar digraph in Fig. 1(c) is acyclic and bimodal, but it does not admit an upward planar drawing.

A *maximum upward planar subgraph* $G'_{\Phi'}$ of G_Φ is an embedding preserving subgraph of G_Φ with the following two properties: (a) $G'_{\Phi'}$ is upward planar; (b) $G'_{\Phi'}$ has the maximum number of edges among the embedding preserving subgraphs of G_Φ that are upward planar. Fig. 1(d) shows a maximum upward planar subgraph of the embedded digraph in Fig. 1(c).

3 Complexity Results

We define the *Fixed Embedding Maximum Upward Planar Subgraph* (*FE-MUPS*) problem as follows.

Problem FE-MUPS: *Given a pair* $\langle G_\Phi, K \rangle$*, where* $G_\Phi = (V, E)$ *is an embedded planar digraph and* K *is an integer number such that* $0 < K < |E|$*, does* G_Φ *admit an embedding preserving subgraph* $G'_{\Phi'} = (V, E')$ *such that* $|E'| = K$ *and* $G'_{\Phi'}$ *is upward planar?*

We prove that FE-MUPS is NP-Complete. The hardness proof uses a reduction from *Planar 3-SAT*, a restricted version of *3-SAT* [9]. To fix notation, we recall the definitions of 3-SAT and Planar 3-SAT.

Problem 3-SAT: *Let* $\langle X, C, \Psi \rangle$ *be a tuple such that* $X = \{x_1, \ldots, x_n\}$ *is a set of boolean variables,* $C = \{c_1, \ldots, c_m\}$ *is a set of clauses such that* $c_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ $(i \in \{1, \ldots, m\})$*, each* $\ell_{i,j}$ $(j \in \{1, 2, 3\})$ *is a literal that can be equal either to a boolean variable* $x \in X$ *or to the negation* $\overline{x}$ *of a boolean*

variable, and Ψ is a boolean formula of the form $\Psi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$. Is there a truth assignment for the variables of X such that Ψ is satisfied?

An instance of Planar 3-SAT is any instance of 3-SAT for which a special graph H_Ψ, associated with Ψ, is planar. The question of Planar 3-SAT is the same as for 3-SAT. Graph H_Ψ is defined as follows (refer to Fig. 2): For each variable $x \in X$, H_Ψ has a vertex associated with x and a vertex associated with its negation $\overline{x}$.[1] H_Ψ has a vertex for each clause $c \in C$, called a *clause-vertex*. H_Ψ has an edge $(\ell_{i,j}, c_i)$ for each literal $\ell_{i,j}$ of c_i ($i \in \{1, \ldots, m\}$, $j \in \{1, 2, 3\}$). H_Ψ has an edge $(x, \overline{x})$ for each variable $x \in X$. H_Ψ has a cycle of edges $(x_1, x_2), (x_2, x_3), \ldots, (x_{n-1}, x_n), (x_n, x_1)$.

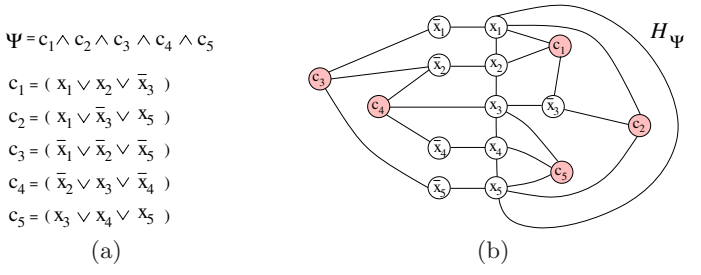


Fig. 2. (a) An instance of Planar 3-SAT. (b) The planar graph H_Ψ associated with Ψ.

Lemma 1. *Problem FE-MUPS is NP-Hard.*

Proof. We define a polynomial-time reduction from a generic instance $\langle X, C, \Psi \rangle$ of Planar 3-SAT to an instance $\langle G_\Phi, K \rangle$ of FE-MUPS, and then we show that Ψ is satisfied if and only if G_Φ has an embedding preserving upward planar subgraph with K edges. Let H_Ψ be the planar graph associated with Ψ along with an arbitrary planar embedding. The embedded planar digraph G_Φ is constructed from the embedded graph H_Ψ as follows: (refer to Fig. 3 for an illustration):

(a) Remove the cycle of edges $(x_1, x_2), (x_2, x_3), \ldots, (x_{n-1}, x_n), (x_n, x_1)$; assign to each edge $(\ell_{i,j}, c_i)$ a label equal to $\ell_{i,j}$ ($i \in \{1, \ldots, m\}, j \in \{1, 2, 3\}$), and then contract each edge $(x_r, \overline{x}_r)$, ($r \in \{1, \ldots, n\}$). Denote by y_r the vertex that originates from the contraction of $(x_r, \overline{x}_r)$, and orient every edge (y_r, c_i) from y_r to c_i (see Fig. 3(a)). The edges (y_r, c_i) will be called *variable edges*.

(b) For each clause vertex c_i add six new vertices $c_{i,1}, \ldots, c_{i,6}$ and the six directed edges $(c_i, c_{i,h})$ ($i \in \{1, \ldots, m\}, h \in \{1, \ldots, 6\}$). The new edges are embedded around c_i in such a way that there are exactly two of them between every pair of circularly consecutive incoming edges of c_i. Also, for each vertex y_r ($r \in \{1, \ldots, n\}$) add two new vertices $y_{r,1}, y_{r,2}$ and the two directed edges

[1] In the original formulation of the Planar 3-SAT problem there is only one vertex per variable, which represents both x and $\overline{x}$. Lichtenstein [13] proved that the Planar 3-SAT problem remains NP-Complete if one considers distinct vertices for x and $\overline{x}$ (see Lemma 1 of [13]); we use this variant.

$(y_{r,1}, y_r)$, $(y_{r,2}, y_r)$; these two edges are embedded around y_r in such a way that they separate the (possibly empty) sequence of edges labeled x_r from the (possibly empty) sequence of edges labeled $\overline{x}_r$ (see Fig. 3(b)). Every edge added during this step will be called a *dangling edge*.

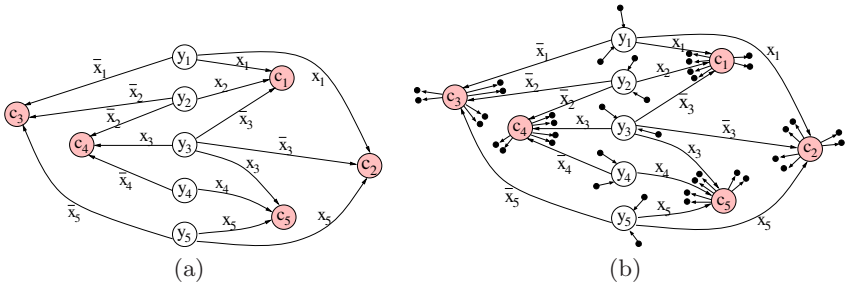


Fig. 3. Reduction from the graph associated with an instance of Planar 3-SAT to the graph of an instance of FE-MUPS. The reduction is done from planar graph H_Ψ of Fig. 2(b). The edges incident to the small black vertices are the dangling edges.

The transformation described above to construct G_Φ from H_Ψ preserves the planarity, and thus G_Φ is an embedded planar digraph. Also, this transformation can be easily performed in $O(m+n)$ time, i.e., in a time linear in the size of Ψ. To complete the reduction, we have to fix a value for K. We choose $K = 7m + 2n$.

Before proving that Ψ is satisfiable if and only if G_Φ has an embedding preserving upward planar subgraph with K edges, we prove that, with our choice of K, every embedding preserving subgraph $G'_{\Phi'}$ of G_Φ with K edges is upward planar if and only if $G'_{\Phi'}$ is bimodal. Clearly, if $G'_{\Phi'}$ is upward planar then it is necessarily bimodal. Suppose vice-versa that $G'_{\Phi'}$ has K edges and is bimodal. We claim that $G'_{\Phi'}$ contains necessarily m variable edges and all the $6m + 2n$ dangling edges. Namely, G_Φ consists of $3m$ variable edges (3 edges incident to each clause-vertex) and $6m + 2n$ dangling edges: If $G'_{\Phi'}$ had less than m variable edges, then it would have less than K edges in total. On the other hand, suppose that $G'_{\Phi'}$ consists of $m + h$ variable edges ($1 \leq h \leq 2m$). Each variable edge is an incoming edge of a clause-vertex; also, if a clause-vertex has $1 + p$ incoming edges in $G'_{\Phi'}$ ($0 \leq p \leq 2$), then it has at most $6 - 2p$ outgoing edges (i.e., incident dangling edges) in $G'_{\Phi'}$, otherwise the bimodality of the clause-vertex would be violated. Since there are exactly m clause-vertices in G_Φ, it follows that if $G'_{\Phi'}$ consisted of $m + h$ variable edges, the number of dangling edges that we can hope to have in $G'_{\Phi'}$ without violating the bimodality would be at most $6m - 2h + 2n$, and therefore the number of edges of $G'_{\Phi'}$ would be at most $m + h + 6m - 2h + 2n = 7m + 2n - h < K$, which contradicts the hypothesis that $G'_{\Phi'}$ has K edges. Hence, if $G'_{\Phi'}$ is an embedding preserving bimodal subgraph of G_Φ with K edges, it consists of exactly m variable edges and of all $6m + 2n$ dangling edges, which proves the claim. ¿From the bimodality of $G'_{\Phi'}$, there is exactly one variable edge incident to each clause-vertex and each vertex

y_r ($r \in \{1, \ldots, n\}$) cannot have incident edges labeled x_r and incident edges labeled $\overline{x}_r$ at the same time. Observe now that the undirected underlying graph of such a bimodal subgraph $G'_{\Phi'}$ does not contain simple cycles; indeed, a simple cycle can only consist of variable edges, but each variable edge connects some clause-vertex c_i to some vertex y_r, and each clause-vertex has only one incident variable edge in $G'_{\Phi'}$. Since every planar embedded bimodal digraph whose underlying graph is acyclic is also upward planar, we conclude that $G'_{\Phi'}$ is upward planar.

We now prove that if Ψ is satisfiable then there exists an embedding preserving upward planar subgraph of G_Φ with K edges. For what we have proved so far, it is sufficient to construct, from the truth assignment of Ψ, an embedding preserving bimodal subgraph $G'_{\Phi'}$ with K edges. $G'_{\Phi'}$ consists of all vertices and all dangling edges of G_Φ; furthermore, add to $G'_{\Phi'}$ m variable edges as follows: For each clause c_i of Ψ ($i \in \{1, \ldots, m\}$), select exactly one literal $\ell_{i,j}$ of c_i having value true ($j \in \{1, 2, 3\}$) and add to $G'_{\Phi'}$ the variable edge with label $\ell_{i,j}$, incident to clause-vertex c_i. By construction, $G'_{\Phi'}$ has $m + 6m + 2n = 7m + 2n = K$ edges, and the bimodality of $G'_{\Phi'}$ is implied by two properties: (i) each clause-vertex has exactly one incoming edge; (ii) each vertex y_r ($r \in \{1, \ldots, n\}$) has at most one circular sequence of consecutive outgoing edges and two circularly consecutive incoming edges, because x_r and $\overline{x}_r$ cannot be true at the same time.

Suppose vice-versa that $G'_{\Phi'}$ is an embedding preserving upward planar subgraph with K edges. As proved above, $G'_{\Phi'}$ has exactly one variable edge incident to each clause-vertex and it contains all dangling edges of G_Φ. This implies that each vertex y_r ($r \in \{1, \ldots, n\}$) cannot have incident edges labeled x_r and incident edges labeled $\overline{x}_r$ at the same time. Therefore, a valid truth assignment that satisfies Ψ can be derived by simply assigning value true to those literals that correspond to labels of the variable edges of $G'_{\Phi'}$. □

The proof of the next lemma is easy and it is omitted for space limitations.

Lemma 2. *Problem FE-MUPS belongs to NP.*

From Lemma 1 and Lemma 2 we have the following result.

Theorem 1. *Problem FE-MUPS is NP-Complete.*

With the reduction used in the proof of Lemma 1, finding an embedding preserving upward planar subgraph of G_Φ with K edges is equivalent to find an embedding preserving bimodal subgraph of G_Φ with K edges. This immediately implies that also finding a maximum embedding preserving bimodal subgraph of an embedded digraph is a hard problem. More formally, the problem *Fixed Embedding Maximum Bimodal Planar Subgraph* is defined as follows.

Problem FE-MBPS: *Given a pair $\langle G_\Phi, K \rangle$, where $G_\Phi = (V, E)$ is an embedded planar digraph and K is an integer number such that $0 < K < |E|$, does G_Φ admit an embedding preserving subgraph $G'_{\Phi'} = (V, E')$ such that $|E'| = K$ and $G'_{\Phi'}$ is bimodal?*

The NP-Hardness of FE-MBPS is implied by the proof of Lemma 1. Also, it is easy to see that FE-MBPS belongs to NP.

Theorem 2. *Problem FE-MBPS is NP-Complete.*

4 Algorithms

Motivated by Theorem 1, we designed a polynomial-time heuristic (Subsection 4.1) and an exponential-time exact algorithm (Subsection 4.2) for computing maximum upward planar subgraphs of embedded planar digraphs. Both these algorithms accept in input an embedded planar digraph G_Φ that is not necessarily acyclic and bimodal. Before describing our techniques, we observe that a straightforward algorithm to compute a maximal upward planar subgraph of G_Φ is as follows: Remove all edges from G_Φ and then try to reinsert an edge per time; each time a new edge e is selected for possible insertion, an upward planarity testing algorithm for fixed embedding is applied on the current subgraph plus edge e; if the test is positive, e is added to the subgraph otherwise e is definitively discarded. Such an algorithm, which we refer to as `SimpleAlgorithm`, is easy to implement and runs in time $O(n^3)$ if one uses the $O(n^2)$-time upward planarity testing technique of Bertolazzi et al. [2]. However, `SimpleAlgorithm` is rather slow in practice, because it applies the upward planarity testing algorithm for each edge of G_Φ (see also Section 5). Instead, we designed an algorithm that is much faster in practice and also more effective than `SimpleAlgorithm`. Furthermore, it represents a key basic tool for the design of the exact algorithm.

4.1 A Fast and Effective Heuristic

Our heuristic, which we call `BendAlgorithm`, computes a maximal upward planar subgraph of the input digraph $G_\Phi = (V, E)$ in three main steps, described below.

– Step 1 computes an embedding preserving subgraph $G'_{\Phi'} \subseteq G_\Phi$ that is bimodal and that contains as much edges as possible. Since by Theorem 2 the problem of finding a maximum bimodal subgraph of G_Φ is NP-Hard, we designed for this step an algorithm that just computes a maximal bimodal subgraph $G'_{\Phi'}$. This algorithm first removes a minimal subset of edges from G_Φ until the digraph becomes bimodal, and then it tries to reinsert each of the removed edges in a random order; an edge is reinserted iff it does not violate the bimodality. The removal of a minimal subset of edges to get the bimodality is based on a greedy procedure. Namely, let v be a vertex of G_Φ and let $E(v)$ be the circular sequence of edges incident to v. If v is not bimodal, denote by $E'(v)$ a minimum subset of edges of $E(v)$ whose removal makes v bimodal. We associate with v a cost $c(v) = |E'(v)|$. Subset $E'(v)$ is computed in time $O(|E(v)|^2)$ by considering all possible splits of $E(v)$ into two linear lists, $E_1(v)$ and $E_2(v)$, and by adding to $E'(v)$ all the incoming edges of v that belong to $E_1(v)$ and all the outgoing edges of $E(v)$ that belong to $E_2(v)$. The removal of the edges of $E'(v)$ from G_Φ makes v bimodal, because the remaining edges of $E_1(v)$ will be outgoing

edges of v, while the remaining edges of $E_2(v)$ will be incoming edges of v. Since $\sum_{v \in V} |E(v)|^2 \leq (\sum_{v \in V} |E(v)|)^2 = (2|E|)^2$, all costs $c(v)$ can be computed in time $O(|E|^2)$. At each phase of the greedy procedure, a non-bimodal vertex v with minimum cost $c(v)$ is selected, and all the edges of $E'(v)$ are temporarily removed. Also, for each edge (u, v) or (v, u) in $E'(v)$, cost $c(u)$ and set $E'(u)$ are updated. We use a binary heap priority queue to efficiently store and update the costs of the non bimodal vertices and to extract their minimum value at each phase. Hence, the greedy procedure consists of at most $|V|$ phases, each requiring $O(|E|^2) = O(|V|^2)$ time. The edges temporarily removed during the greedy procedure are possibly reinserted incrementally, one per time; this is done in $O(|E|^2)$, because the bimodality can be tested efficiently for each edge reinsertion. Hence, Step 1 takes $O(|V|^3)$ time.

– Step 2 temporarily removes from $G'_{\Phi'}$ a minimal number of edges in order to get an upward planar subgraph $G''_{\Phi''} \subseteq G'_{\Phi'}$. To do this, it applies a global strategy. More precisely, in [1] the concept of *quasi-upward planar drawing* of an embedded bimodal digraph is introduced. Roughly speaking, a quasi-upward planar drawing is an upward drawing that allows some bends along the edges, where a bend represents an inversion of direction of an edge. If one removes all the bent edges in a quasi-upward planar drawing, the remaining drawing is upward planar. In [1] an $O(|V|^2 \log |V|)$ flow-based algorithm is described for the computation of a quasi-upward planar drawing of an embedded bimodal digraph, having the minimum number of bends. In general this is not equal to determine the minimum number of edges whose removal leads to the upward planarity, but it typically behaves as an effective heuristic to this aim. Thus, to compute $G''_{\Phi''}$ we apply on $G'_{\Phi'}$ the flow-based algorithm of [1] and temporarily remove the bent edges.

– Step 3 tries to incrementally reinsert in a random order the edges removed in Step 2, by testing each edge reinsertion for upward planarity. Each test is done by applying the $O(|V|^2)$-time algorithm of Bertolazzi et al. [2]. The number of tests executed is equal to the number of edges removed during Step 2, which will be proved to be rather small in practice (see Section 5).

The following lemma summarizes the discussion above.

Lemma 3. *Let G_Φ be an embedded planar digraph and let n be the number of vertices of G_Φ.* `BendAlgorithm` *computes a maximal upward planar subgraph of G_Φ in $O(n^3)$ time.*

4.2 An Exact Algorithm

Our exact algorithm, which we call `BBAlgorithm`, is based on a branch-and-bound technique. Let $G_\Phi = (V, E)$ be the input digraph and let $E = \{e_1, \ldots, e_m\}$ be the set of its edges. To encode any subset E' of E we use an array of binary variables $X_{E'} = \{x_1, x_2, \ldots, x_m\}$, where $x_i = 0$ if edge e_i does not belong to E', and $x_i = 1$ if e_i belongs to E'. The optimal solution is an array $X_{E'}$ such that subgraph (V, E') is upward planar and the number of variables of $X_{E'}$ having value 0 is minimized (in the following a variable of value 0 will be called a *zero variable*).

The branch-and-bound tree T is a complete binary tree with levels $0, 1, \ldots, m$, where the leaves represent all subsets of E. Each leaf of T is an array $X_{E'}$, for some $E' \subseteq E$. An internal node μ of T at level i ($1 \leq i < m$) is associated with an array of i binary variables $X_\mu = \{x_1^{(\mu)}, x_2^{(\mu)}, \ldots, x_i^{(\mu)}\}$; in the subtree T_μ rooted at μ, each leaf is an array $X_{E'} = \{x_1^{(\mu)}, x_2^{(\mu)}, \ldots, x_i^{(\mu)}, x_{i+1}, \ldots, x_m\}$, i.e., it represents a subset $E' \subseteq E$ such that $e_j \in E'$ iff $x_j^{(\mu)} = 1$ ($1 \leq j \leq i$).

The algorithm visits T from the root to the leaves, and an array $\overline{X}$ corresponding to the current best solution is kept updated during the visit ($\overline{X}$ always coincides with a leaf of T). At the beginning of the visit, $\overline{X}$ is set equal to a solution computed with the `BendAlgorithm`. Each time a new internal node μ at a level i is visited, the subgraph induced by the non-zero variables $x_1^{(\mu)}, x_2^{(\mu)}, \ldots, x_i^{(\mu)}$ is tested for upward planarity; if the test is negative T_μ is cut and it will be not visited in the following; otherwise an upper bound $u(\mu)$ and a lower bound $l(\mu)$ to the number of zero variables contained in any leaf of T_μ are computed. In particular, $u(\mu)$ will correspond to the number of zero variables in some leaf $X_{E'}$ of T_μ, representing an upward planar subgraph. If $u(\mu)$ is smaller than the number of zero variables of $\overline{X}$, then $\overline{X}$ is updated with $X_{E'}$. If $l(\mu)$ is greater than or equal to the number of zero variables of $\overline{X}$, then subtree T_μ is cut and it will be not visited. To visit T the algorithm applies a depth-first-search, which uses a stack to store the visited nodes. At any time of the visit, the number of nodes stored in the stack is $O(m)$ (i.e., order of the depth of T). At the end of the visit, $\overline{X}$ will represent an optimal solution. We now describe how $u(\mu)$ and $l(\mu)$ are computed.

The upper bound $u(\mu)$ is determined by applying a technique similar to the one used by `BendAlgorithm`. Namely, we first complete the array X_μ to an array $X_{E'}$ corresponding to a maximal bimodal subgraph of the input digraph; this is done by incrementally testing for insertion all the edges $e_{i+1}, \ldots, e_m$. After that, we apply on the subgraph associated with $X_{E'}$ the flow-based algorithm in [1], so to get a quasi-upward planar drawing; the bent edges are temporarily removed and then incrementally tested for possible reinsertion with the algorithm in [2]. To guarantee that none of the edges e_j for which $x_j^{(\mu)} = 1$ is removed ($1 \leq j \leq i$), we constrain these edges to have no bend in the computed quasi-upward planar drawing; this is done by imposing a suitable flow constraint.

The lower bound $l(\mu)$ is computed by performing $m - i$ steps. For each $h = i + 1, \ldots, m$, we test if the subgraph induced by the non-zero variables of X_μ plus edge e_h is bimodal and upward planar; a negative test implies that e_h is not contained in any of the upward planar subgraphs represented by the leaves of T_μ, and then we increment $l(\mu)$ by one unit.

5 Experimental Study

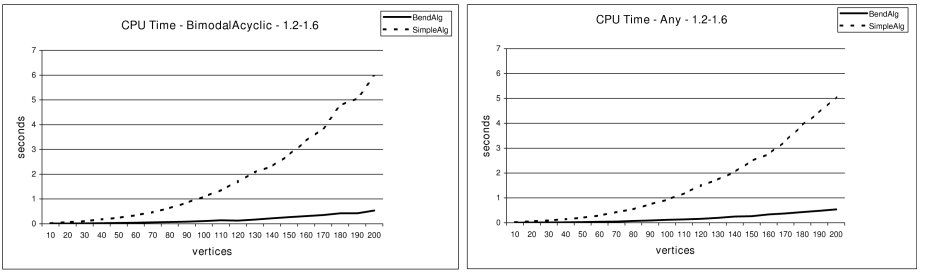
We implemented `SimpleAlgorithm`, `BendAlgorithm`, and `BBAlgorithm` and experimentally compared their performances. For the implementation we used

C++ and the GDToolkit graph drawing library[2]. The experiments were executed under Linux OS, on a machine with an Intel Centrino 1.66 GHz and 2GB of RAM. For the experiments we used three different test suites of connected planar digraphs. The first test suite, which we call `BimodalAcyclic`, is a set of 800 embedded planar digraphs that are bimodal and acyclic, and having number of vertices in $\{10, 20, \ldots, 200\}$. Since we observed that the performances of our algorithms are strongly influenced by the density of the input digraph, we generated 10 different digraphs for each fixed number of vertices and distinct density value in $\{1.2, 1.6, 2.0, 2.4\}$. Each digraph in `BimodalAcyclic` was obtained by randomly generating an embedded upward planar digraph with the algorithm described in [7] and then changing at random the orientation of the 50% of the edges, while preserving bimodality and acyclicity; if the resulting digraph was still upward planar it was discarded and generated again. The second test suite, which we call `Any`, consists of 800 embedded planar digraphs with no additional restriction. As a consequence, a digraph in this test suite is in general not bimodal and not acyclic. Again, digraphs in `Any` have number of vertices in $\{10, 20, \ldots, 200\}$ and density in $\{1.2, 1.6, 2.0, 2.4\}$. Each digraph in `Any` was generated at random by first generating a tree and then adding a number of edges between the vertices of the tree, until the desired value of density was achieved. Each edge was then randomly oriented with a uniform probability distribution. Finally, we used a third test suite, called `Rome`, derived from the well known set of graphs defined in [6], and often recognized as "Rome Graphs". The Rome Graphs have number of vertices in $[10, 100]$, are not directed and, in general, not planar. At the web site http://www.dia.uniroma3.it/∼gdt/, an oriented version of the Rome Graphs is available, where each edge has been oriented at random. We randomly selected 50 of these digraphs for each fixed number of vertices in $\{10, 15, 20, \ldots, 95, 100\}$, for a total of 1000 digraphs. Then, for each of these digraphs, we planarized it (by possibly adding dummy vertices) and randomly chose a planar embedding. The average density of such digraphs is about 1.4.

We first compared `BendAlgorithm` and `SimpleAlgorithm` on the first two test suites. `BendAlgorithm` runs pretty fast and outperforms `SimpleAlgorithm`. Indeed, `SimpleAlgorithm` executes the $O(n^2)$ upward planarity testing described in [2] for each edge insertion, while `BendAlgorithm` applies the same test only for those edges having some bends in the quasi-upward drawing computed in Step 2, which are a small percentage of the whole set of edges (around 2.5% for low density digraphs and about $8 - 10\%$ for high density digraphs). Also, the running times of Steps 1 and 2 of `BendAlgorithm` are in practice negligible with respect to the time taken from the reinsertion process in Step 3, even for the digraphs in `Any` (Steps 1-2 take about 0.01 seconds for graphs of 200 vertices and density 1.6, and 0.2 seconds for graphs of 200 vertices and density 2.4).

The effectiveness of the two heuristics is measured in terms of the size of their solutions, i.e., the number of edges in the computed maximal upward planar subgraphs; we express such a size as a percentage of the whole set of edges of the input digraph. We observed that this percentage does not depend on

[2] http://www.dia.uniroma3.it/∼gdt/



(a) (b)

Fig. 4. Average CPU time of `BendAlgorithm` (solid line) and `SimpleAlgorithm` (dashed line) on the `BimodalAcyclic` and `Any` instances. For space reasons, we group the data on densities 1.2, 1.6 and omit the data for higher densities.

the number of vertices of the input digraph, but only on the density of the digraphs (see Table 1). One can observe that `BendAlgorithm` is more effective than `SimpleAlgorithm` and the difference in the effectiveness of the two heuristics grows with the increasing of the density. We also observe that the solutions computed for the digraphs in `Any` have smaller size than those computed for the digraphs in `BimodalAcyclic`. This because the digraphs in `Any` are typically not bimodal and not acyclic, and therefore require more edges to be deleted. The computations on the `Rome` test suite confirmed the behavior of the two heuristics on the digraphs with densities $1.2-1.6$ in the `Any` test suite; `BendAlgorithm` and `SimpleAlgorithm` inserted 93.02% and 90.18% of the total edges, respectively.

Table 1. Average percentage (size) of number of edges in the solutions computed by `BendAlgorithm` and `SimpleAlgorithm` for each density value. The differences between the sizes of the solutions computed with the two algorithms are also shown.

`BimodalAcyclic`				`Any`			
Density	`BendAlg`	`SimpleAlg`	Diff	Density	`BendAlg`	`SimpleAlg`	Diff
1.2	97.87%	97.32%	0.55%	1.2	94.78%	94.18%	0.59%
1.6	95.79%	93.52%	2.27%	1.6	88.01%	85.27%	2.74%
2.0	94.42%	91.50%	2.92%	2.0	81.95%	77.79%	4.16%
2.4	93.83%	89.96%	3.87%	2.4	77.43%	73.02%	4.40%

We now analyze the performances of the exact method `BBAlgorithm` and compare its solutions with those of `BendAlgorithm`. The running time required by `BBAlgorithm` is often too long for digraphs with more than 100 edges. Therefore, we decided to run `BBAlgorithm` only on the digraphs having up to 100 vertices and density up to 1.6; also, we stopped the computation in any case after a time t of 3 minutes. In total we have 65.5% of the instances in `BimodalAcyclic` (i.e., 131 instances) and 40.5% of the instances in `Any` (i.e., 81 instances). For the `Rome` digraphs, the percentage of instances solved within t is 48.7% (i.e., 487

instances). For all solved instances, we compared the size of the optimal solutions with those computed by `BendAlgorithm` to get an estimation of how much `BendAlgorithm` approximates the optimum: `BendAlgorithm` achieves the optimum on 92.37% of the `BimodalAcyclic` instances, on 67.9% of the `Any` instances, and on 83.98% of the `Rome` instances. In the remaining 7.63% of `BimodalAcyclic` instances, the optimum has in the average 2.35% edges more than the solution of `BendAlgorithm`; this percentage grows to 4.02% for the remaining 32.1% of the `Any` instances, and to 3.78% for the remaining 16.02% of the `Rome` instances. `BendAlgorithm` is therefore a good approximation of the optimum in many cases.

References

1. Bertolazzi, P., Di Battista, G., Didimo, W.: Quasi-upward planarity. Algorithmica 32(3), 474–506 (2002)
2. Bertolazzi, P., Di Battista, G., Liotta, G., Mannino, C.: Upward drawings of triconnected digraphs. Algorithmica 6(12), 476–497 (1994)
3. Bertolazzi, P., Di Battista, G., Mannino, C., Tamassia, R.: Optimal upward planarity testing of single-source digraphs. SIAM J. Comput. 27, 132–169 (1998)
4. Chan, H.: A parameterized algorithm for upward planarity testing. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 157–168. Springer, Heidelberg (2004)
5. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing. Prentice Hall, Upper Saddle River, NJ (1999)
6. Di Battista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., Vargiu, F.: An experimental comparison of four graph drawing algorithms. Computational Geometry: Theory and Applications 7, 303–326 (1997)
7. Didimo, W.: Upward planar drawings and switch-regularity heuristics. Journal of Graph Algorithms and Applications 10(2), 259–285 (2006)
8. Didimo, W., Giordano, F., Liotta, G.: Upward spirality and upward planarity testing. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 117–128. Springer, Heidelberg (2006)
9. Garey, M.R., Johnson, D.S.: Comput. and Intract. Freeman and Co, San Francisco (1979)
10. Garg, A., Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. SIAM J. Comput. 31(2), 601–625 (2001)
11. Healy, P., Lynch, K.: Fixed-parameter tractable algorithms for testing upward planarity. International Journal of Foundations of Computer Science 17(5) (2006)
12. Hutton, M.D., Lubiw, A.: Upward planarity testing of single-source acyclic digraphs. SIAM J. Comput. 25(2), 291–311 (1996)
13. Lichtenstein, D.: Planar formulae and their uses. SIAM J. Comput. 11(2), 329–343 (1982)
14. Papakostas, A.: Upward planarity testing of outerplanar dags. In: Tamassia, R., Tollis, I(Y.) G. (eds.) GD 1994. LNCS, vol. 894, pp. 298–306. Springer, Heidelberg (1995)

Minimizing the Area for Planar Straight-Line Grid Drawings

Marcus Krug and Dorothea Wagner

Universität Karlsruhe
{krug,wagner}@ira.uka.de

Abstract. Straight-line grid drawings of bounded size is a classical topic in graph drawing. The Graph Drawing Challenge 2006 dealt with minimizing the area of planar straight-line grid drawings. In this paper, we show that it is NP-complete to decide if a planar graph has a planar straight-line drawing on a grid of given size. Furthermore, we present a new iterative approach to compactify planar straight-line grid drawings. In an experimental study, we evaluate the quality of the compactified drawings with respect to the size of the area as well as to other measures.

1 Introduction

A planar straight-line grid drawing of a planar graph is an assignment of the nodes of the graph to integral points in the plane, such that the edges of the graph are mapped to non-crossing straight-lines. The size of such a drawing is typically measured by the number of grid points contained in the bounding box of the drawing, i.e. the smallest axis-parallel rectangle covering the drawing. Minimizing the size of planar straight-line grid drawings has a long tradition. De Fraysseix et al. [1] and Schnyder [2] were the first, who independently showed that every (3-connected) plane graph with n nodes has a straight-line grid drawing on a grid of size $(2n-4) \times (n-2)$ and $(n-2) \times (n-2)$, respectively. Zhang and He [3] showed, that every plane graph with n nodes has a straight-line grid drawing on a grid of size $(n-\Delta_0-1) \times (n-\Delta_0-1)$, where $0 \leq \Delta_0 \leq \lfloor (n-2)/2 \rfloor$ is the number of cyclic faces of the graph with respect to its minimum Schnyder realizer. Minimizing the area has been shown to be NP-complete in several orthogonal drawing models, e.g. by Kramer and van Leeuwen in [4]. However, it has remained open if NP-completeness applies to minimizing the area for straight-line grid drawings as well.

In this paper, we close this gap and show that it is NP-complete to decide if a planar graph has a planar straight-line drawing on a grid of a given size. In order to come up with compact straight-line grid drawings nevertheless, we propose a novel iterative compaction algorithm. The algorithm starts with a planar straight-line grid drawing and computes a more compact drawing by iteratively evaluating new positions for each node. Moreover, we evaluate the algorithm based on a sample of 400 graphs, which were generated using a common planar

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 207–212, 2007.

graph generator. We will show that the proposed algorithm is capable of producing substantially compactified drawings with additional nice properties.

2 NP-Completeness

Let G be a planar graph. We will consider the problem of finding a planar straight-line grid drawing $\mathcal{E}$ for G with minimal size, where the size of the drawing is measured by the number of grid points inside the bounding box $\mathcal{B}(\mathcal{E})$, denoted by $|\mathcal{B}(\mathcal{E})|$. We will refer to this problem as Minimum Area Planar Straight-Line Grid Drawing and we will consider the following related decision problem:

Minimum Area Planar Straight-Line Grid Drawing
INSTANCE: A planar graph G and an integer A.
QUESTION: Is there a planar straight-line grid drawing $\mathcal{E}$ for G, such that $|\mathcal{B}(\mathcal{E})| \leq A$?

Theorem 1. Minimum Area Planar Straight-Line Grid Drawing *is NP-complete.*

Proof. We will only outline the main idea of the proof here, a full proof can be found in [5]. It is obvious, that Minimum Area Planar Straight-Line Grid Drawing is in NP. We will prove NP-hardness by reduction from 3-Partition, which is defined as follows:

3-Partition
INSTANCE: $3m$ positive integers $A = \{a_1, \ldots, a_{3m}\}$, a positive integer bound b, such that $b/4 < a_i < b/2$ for all $i = 1, \ldots, 3m$
QUESTION: Can A be partitioned into m disjoint sets $A_1, \ldots, A_m$, such that $\sum_{a_i \in A_j} a_i = b$ for all $j = 1, \ldots, m$?

In [6] Garey and Johnson proved that 3-Partition is NP-complete. It is essential for the reduction that 3-Partition is NP-complete in the strong sense, i.e. it remains NP-complete, even if b is bounded by a polynomial in m. We will assume that $b \geq 8$ since the problem is trivial for smaller values of b.

For an instance $\mathcal{I} = (A, m, b)$ of 3-Partition we will construct an instance $\mathcal{I}' = (G_\mathcal{I}, A_\mathcal{I})$ of Minimum Area Planar Straight-Line Grid Drawing, such that $A_\mathcal{I} = 3p$, where p is the smallest prime greater or equal to $l + 17$ with $l = (m + 1)(2b + 2) - m$. By the Bertrand-Chebyshev theorem such a prime number does exist in the interval $[l + 17, 2(l + 17)]$. It is vital for the proof, that the only grid of size $A_\mathcal{I}$ is a grid of width p and height three, or vice versa. Each number a_i will be represented as a path of length $2a_i$, and the graph $G_\mathcal{I}$ is given by the union of the paths representing the numbers in A and the frame graph $F_{m,b,p}$ associated with the instance $\mathcal{I}$. A drawing of the frame graph on a grid of size $p \times 3$ is illustrated in Fig. 1. It consists of a linked sequence of $m + 2$ diamond-shaped graphs $D_1, \ldots, D_{m+2}$, which are additionally belted by a path.

Each D_i ($i < m+2$) consists of two adjacent nodes with degree $2b+3$ sharing $2b+2$ common neighbors. These common neighbors are linked as two disjoint paths of length $b+1$ each. The graph D_{m+2} is constructed analogously with $l-p+1$ common neighbors. Up to rotation and reflection, there is only one way of drawing the frame graph on a grid of size $A_{\mathcal{I}}$, assuming that none of the grid points may be covered by an edge, i.e. as illustrated in Fig. 1. Since the number of nodes of $G_{\mathcal{I}}$ equals $A_{\mathcal{I}}$, this condition holds for any drawing of $G_{\mathcal{I}}$ on any grid of size $A_{\mathcal{I}}$. The given drawing of the frame graph leaves exactly m boxes

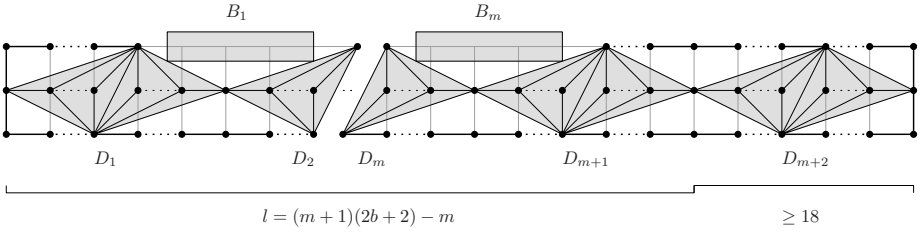


Fig. 1. The frame graph used in the reduction from 3-Partition: The diamond-shaped graphs D_i are accentuated in grey

$B_1, \ldots, B_m$ of width $2b$ and height one unoccupied, such that $B_i \cap B_j = \emptyset$ for $i \neq j$. Therefore, we obtain a one-to-one correspondence between the problem of deciding, if the set A can be partitioned according to 3-Partition and the problem of deciding, if the paths representing the numbers in A can be drawn into the boxes B_i, i.e. if $G_{\mathcal{I}}$ has a straight line drawing on a grid of size $A_{\mathcal{I}}$, which concludes the proof. □

Note, that the proof also yields that it is NP-complete to decide if a graph has a straight-line drawing on a grid with given width and height, and that the problem remains NP-complete if restricted to plane graphs, i.e. graphs with a given combinatorial embedding.

3 Compaction Algorithm

In this section, we will present an iterative algorithm for the compaction of planar straight-line grid drawings. The algorithm takes a plane graph as an input and produces a more compact drawing. It works in iterations and rounds. The number of iterations is a parameter of the algorithm. At the beginning of each iteration, we compute the bounding box of the current drawing and randomly select a grid point ϱ inside the bounding box, which we refer to as the reference point for this iteration. Each iteration then consists of a certain number of rounds. In each round, the nodes of G are examined with ascending distance to the reference point ϱ. For each node v, the algorithm considers a set P_v of possible new positions for v according to an evaluation function $\varphi_{v,\varrho}$ on P_v. Both P_v and $\varphi_{v,\varrho}$ will be explained below. For each grid point p in

P_v, the algorithm computes $\varphi_{v,\varrho}(p)$ and tentatively assigns v to the grid point with the smallest negative value. If the resulting drawing is planar v will be permanently assigned to this grid point for the rest of this round. Otherwise the grid point will be discarded and the next best grid point with respect to $\varphi_{v,\varrho}$ will be considered. If there is no grid point p in P_v such that $\varphi_{v,\varrho}(p) < 0$, the next node will be considered and no action will be taken for v. If none of the nodes could be assigned to a new grid point in one round, the current iteration ends. In our implementation we considered the set P_v of unoccupied grid points p with distance less than or equal to $D = \sqrt{2}$ to the current position of v. Note, that a different choice of P_v is also possible. The evaluation function is given by

$$\varphi_{v,\varrho}(p) = \psi_{v,\varrho}(p) - \psi_{v,\varrho}(v), \tag{1}$$

where

$$\psi_{v,\varrho}(p) = (p-\varrho)^2 + \sum_{\{v,w\}\in E} (p-w)^2 \ . \tag{2}$$

The intuition behind the evaluation function is to group the nodes around the reference point thereby compactifying the drawing while simultaneously enforcing short edges in order to prevent the algorithm from running into a local optimum. By randomly choosing reference points in each iteration, we further seek to make the algorithm more robust against local minima.

Since the number of rounds performed during one iteration is not bounded, we must ensure, that the algorithm terminates. To this end, we will consider a global evaluation function $\mathcal{F}_\varrho$ for a given drawing and a given reference point ϱ. We will define $\mathcal{F}_\varrho$ as

$$\mathcal{F}_\varrho(\mathcal{E}) = \sum_{v\in V} (v-\varrho)^2 + \sum_{\{v,w\}\in E} (v-w)^2 \ . \tag{3}$$

Suppose that the new drawing $\mathcal{E}'$ is obtained from the drawing $\mathcal{E}$ by moving the node v from p to p', such that $\varphi_{v,\varrho}(p') < 0$ as imposed by the algorithm. Then,

$$\mathcal{F}_\varrho(\mathcal{E}') - \mathcal{F}_\varrho(\mathcal{E}) = \psi_{v,\varrho}(p') - \psi_{v,\varrho}(p) = \varphi_{v,\varrho}(p') < 0 \ . \tag{4}$$

Hence, moving a node can only decrease $\mathcal{F}_\varrho$ and $\mathcal{F}_\varrho \geq 0$ by definition.

4 Experimental Evaluation

We have performed an extensive experimental study based on more than 5,000 graphs generated using a variety of different planar graph generators. However, due to limitations of space, we will only present a small fraction of our results here. We will restrict ourselves to the sample of graphs generated using the LEDA `random_planar_graph` function. We generated graphs with n nodes where $n \in \{250, 500, 750, 1000\}$. For each n we generated 20 graphs with m edges each, where $m \in \{n, \frac{3}{2}n, 2n, \frac{5}{2}n, 3n-6\}$, resulting in a total of 400 graphs. The graphs were initially drawn on a grid of size $(n-2) \times (n-2)$ using the algorithm

proposed by Schnyder in [2]. The performance of the algorithm is measured by the *compaction factor* γ, which relates the area A_0 of the initial drawing to the area A of the final drawing, i.e. $\gamma = \frac{A_0}{A}$. A high compaction factor indicates, that the algorithm substantially compactified the initial drawing. We performed 20 iterations of the compaction algorithm on all initial drawings.

The observed compaction factors range from 1.5 up to 135. For more than 50% of the graphs the compaction factors achieved exceed a value of 10, i.e. the area of the compactified drawings occupies merely less than $\frac{1}{10}$-th of the initially required area. However, 4% of the compactified drawings still occupy more than half the area of the initial drawing.

Expectedly, the observed compaction factors depend on both the number of nodes and the density of the graph. For a given number of nodes, a higher density implies less degrees of freedom for the placement of nodes. Similarly, depending on the structure of the graph, a larger number of nodes may imply longer edges for a given density and, thus, less freedom for the placement of nodes, e.g. the graph consisting of a nested sequence of k triangles. The lower and upper quartiles as well as the median of the observed compaction factors decrease as the number of nodes increases: More than 75% of the drawings of the graphs with 250 nodes achieved compaction factors larger than 10, as opposed to only 21% of the graphs with 1000 nodes. Similarly, the compaction factors decrease as the density of the graphs increases: More than 80% of the drawings of the very sparse graphs ($m = n$) and still more than 75% of those of the sparse graphs ($m = 1.5n$) achieved compaction factors of 10 and larger, as opposed to 42% for graphs with high density ($m = 2.5n$) and only 26% for the graphs with maximum density ($m = 3n - 6$).

The compactified drawings are satisfactory with respect to commonly used measures for graph drawings (Fig. 2): The aspect-ratios of the compactified drawings, i.e. the ratio of longer vs. shorter side of the bounding box, range between 1 and 4. More precisely, in 95% of the drawings the aspect ratio does not exceed a value of 2. In more than 65% of the compactified drawings the angular resolution, i.e. the smallest angle between incident edges, is not worse than in the initial drawings. As expected, the edge-length resolution, i.e. the

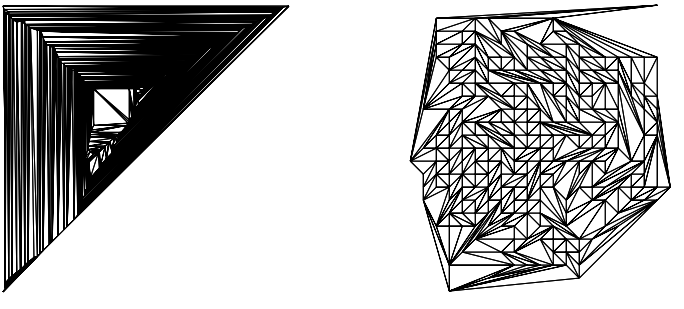


Fig. 2. Sample output of the compaction algorithm: Initial drawing (left) and scaled (by a factor of ≈ 11) compactified drawing (right); $\gamma \approx 128$

ratio of longest vs. shortest edge-length is also improved by the compactified drawings.

The running time of the algorithm is substantially influenced by the number of rounds performed per iteration. The number of rounds performed during the 20 iterations of the algorithm seems to fit the model cn^k with $k > 1$. Using non-linear least squares fitting, we obtain $c = 0.37 \pm 10^{-4}$ and $k = 1.4 \pm 10^{-4}$, i.e. the number of rounds per iteration seems to be slightly super-linear in the number of nodes. However, this number tends to decrease with each iteration, since it depends on the size of the drawing.

5 Conclusion and Outlook

We proved that it is NP-complete to decide if a given planar graph has a planar straight-line grid drawing with a given size. We further proposed an iterative algorithm for the compactification of planar straight-line grid drawings. Our experimental evaluation of the algorithm reveals, that the performance of the algorithm depends on both the number of nodes and the density of the graphs. Due to the local nature of the algorithm, the initial drawing has a great influence on the performance. In addition to that the tests for intersections which must be performed after each update of a node may be costly since they may have to be performed for a large number of edges. In order to speed up the computation, the number of edges which have to be tested for intersections can be computed using a logical indexing approach which uses the combinatorial embedding of the graph. We were able to show, that the algorithm performs well on a large number of graphs, i.e. more than 50% of the compactified drawings require less than $\frac{1}{10}$-th of the space required for the initial drawing, starting from an already compact drawing.

References

1. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10(1), 41–51 (1990)
2. Schnyder, W.: Embedding planar graphs on the grid. In: SODA 1990. Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 138–148 (1990)
3. Zhang, H., He, X.: Canonical ordering tree and its applications in graph drawing (2003)
4. Kramer, M., van Leeuwen, J.: The NP-completeness of finding minimum area layouts for VLSI-circuits (to appear). Technical Report RUU-CS-82-06, Institute of Information and Computing Sciences, Utrecht University (1982)
5. Krug, M.: Minimizing the area for planar straight-line grid drawings. Master's thesis, Universität Karlsruhe (2007)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., New York (1990)

On Planar Polyline Drawings

Huaming Zhang and Sadish Sadasivam

Computer Science Department
University of Alabama in Huntsville
Huntsville, AL, 35899, USA
{hzhang,ssadasiv}@cs.uah.edu

Abstract. We present a linear time algorithm that produces a planar polyline drawing for a plane graph with n vertices in a grid of size bounded by $(p+1) \times (n-2)$, where $p \leq (\lfloor \frac{2n-5}{3} \rfloor)$. It uses at most $p \leq \lfloor \frac{2n-5}{3} \rfloor$ bends, and each edge uses at most one bend. Compared with the area optimal polyline drawing algorithm in [3], our algorithm uses a larger grid size bound in trade for a smaller bound on the total number of bends. Their bend bound is $(n-2)$. Our algorithm is based on a transformation from Schnyder's realizers [6,7] of maximal plane graphs to transversal structures [4,5] for maximal internally 4-connected plane graphs. This transformation reveals important relations between the two combinatorial structures for plane graphs, which is of independent interest.

1 Introduction

We focus on planar graph drawings. Such graphs can be drawn without any edge crossings. Several styles of drawings [1] have been introduced. Common objectives include small area, few bends and good angular resolution. We deal with polyline drawings [1]. A polyline drawing is a drawing of a graph in which each edge is represented by a polygonal chain and every vertex is placed on a grid point. Bonichon et al. [3] presented a linear time algorithm that produces polyline drawings for a graph with n vertices within a grid of area $(n - \lfloor \frac{p}{2} \rfloor - 1) \times (p+1)$, where $p \leq \frac{2n-5}{3}$. It is area optimal and each edge has at most one bend. However the total number of bends used by this algorithm could be $(n-2)$.

Our goal is to have a tradeoff between the grid size and the number of bends. We present a linear time algorithm that produces a polyline drawing in a grid with size bounded by $(p+1) \times (n-2)$, where $p \leq \lfloor \frac{2n-5}{3} \rfloor$, and each edge uses at most one bend. Although the grid size is not as good as the algorithm in [3], our algorithm only needs at most $p \leq \lfloor \frac{2n-5}{3} \rfloor$ bends.

A maximal plane graph G is associated with *realizers* $\mathcal{R}$ [6,7], which is a partition of the set of interior edges into three particular trees. A maximal internally 4-connected plane graph G' with four exterior vertices is associated with *transversal structures* $\mathcal{T}$ [4,5], which is a partition of the set of interior edges into two *st*-graphs. In this paper, we introduce a transformation from a maximal plane graph G to a maximal internally 4-connected plane graph G' with

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 213–218, 2007.

four exterior vertices by a certain number of operations. These operations are determined by a realizer of G and can be done in linear time. Then our algorithm uses the derived G' and its transversal structure to obtain the polyline drawing.

The present paper is organized as follows. In Section 2, we recall a few definitions. In Section 3, we present the transformation from a realizer to a transversal structure. Then we present our drawing algorithm.

2 Preliminaries

The graphs are simple graphs. We abbreviate "counter clockwise" and "clockwise" as ccw and cw respectively.

Definition 1. *[6,7] Let G be a maximal plane graph of n vertices with three exterior vertices v_1, v_2, v_3 in ccw order. A* realizer $\mathcal{R}(G) = \{T_1, T_2, T_3\}$ *of G is a partition of its interior edges into three sets T_1, T_2, T_3 of directed edges such that the following holds: (1) for each $i \in \{1, 2, 3\}$, the interior edges incident to v_i are in T_i and directed toward v_i; (2) for each interior vertex of G, v has exactly one edge leaving v in each of T_1, T_2, T_3. The ccw order of the edges incident to v is: leaving in T_1, entering in T_3, leaving in T_2, entering in T_1, leaving in T_3 and entering in T_2. Each entering block could be empty.*

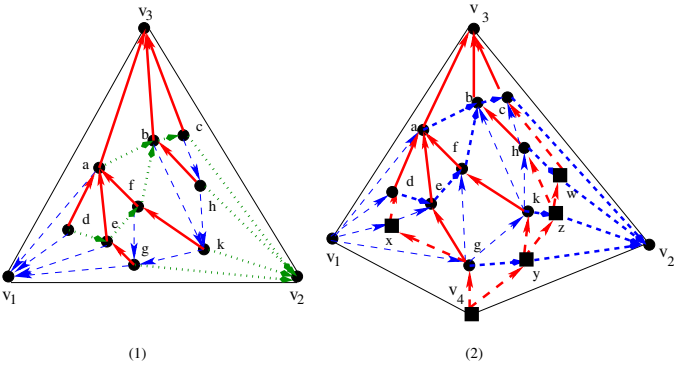


Fig. 1. (1) A maximal plane graph G and a realizer $\mathcal{R}(G)$ of G. (2) A maximal internally 4-connected plane graph G' with four exterior vertices and a transversal structure $\mathcal{T}(G')$ for G'.

Schnyder presented a linear time algorithm to construct a realizer for G. An example of a maximal plane graph G, and one of its realizers is given in Fig. 1 (1). Next, we introduce the concept of *transversal structures* [4,5].

Definition 2. *let G' be a maximal internally 4-connected plane graph with four exterior vertices v_1,v_4,v_2, and v_3 in ccw order. A* transversal structure $\mathcal{T}(G')$ *of G' is a partition of its interior edges into two sets, say in red and blue edges, such that the following conditions are satisfied:*

1. *In cw order around each interior vertex v, its incident edges form: a non empty interval of red edges entering v, a non empty interval of blue edges entering v, a non empty interval of red edges leaving v, and a non empty empty interval of blue edges leaving v.*
2. *All interior edges incident to v_3 are red edges entering v_3, all interior edges incident to v_4 are red edges leaving v_4, all interior edges incident to v_1 are blue edges leaving v_1, and all interior edges incident to v_2 are blue edges entering v_2. Each such block is non empty.*

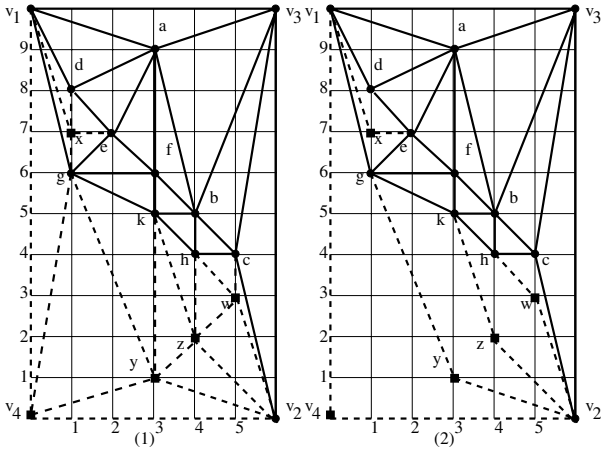


Fig. 2. (1) A straight-line grid drawing of the graph G' in Fig. 1 (2). (2) A polyline drawing of G in Fig. 1 (1).

Fig. 1 (2) shows an example of a transversal structure $\mathcal{T}(G')$ for a maximal internally 4-connected plane graph G' with four exterior vertices. The subgraph of G' with all its red-colored edges (blue colored edges respectively) and all its four exterior edges is called a *red map* (*blue map* respectively) of G', it is denoted by G'_r (G'_b respectively). For any interior vertex v of G', let $P_b(v)$ be the unique path from v_1 to v_2 in G'_b such that, the subpath of $P_b(v)$ from v_1 to v is the rightmost one before arriving at v, and the subpath of $P_b(v)$ from v to v_2 is the leftmost one after leaving v. Let $y(v)$ be the number of faces in G'_b enclosed by the path (v_1, v_4, v_2) and $P_b(v)$. Similarly, for any interior vertex v of G', let $P_r(v)$ be the unique path from v_4 to v_3 in G'_r such that, the subpath of $P_r(v)$ from v_4 to v is the rightmost one before arriving at v, and the subpath of $P_r(v)$ from v to v_3 is the leftmost one after leaving v. Let $x(v)$ be the number of faces in G'_r enclosed by the path (v_4, v_1, v_3) and $P_r(v)$.For example, vertex k in Fig. 1 (2) satisfies $P_r(k) = (v_4, y, k, f, a, v_3)$, so that $x(k) = 3$; and $P_b(k) = (v_1, g, k, b, c, v_2)$, so that $y(k) = 5$. Let $x(\mathcal{T}(G'))$ be the number of interior faces of G'_r. $y(\mathcal{T}(G'))$ be the number of interior faces of G'_b. For the vertices v_1, v_2, v_3, v_4, we define $x(v_1) = 0, y(v_1) = y(\mathcal{T}(G')), x(v_4) = 0, y(v_4) = 0$,

$x(v_3) = x(\mathcal{T}(G')), y(v_3) = y(\mathcal{T}(G'))$, and $x(v_2) = x(\mathcal{T}(G')), y(v_2) = 0$. We have the following lemma from [4]:

Lemma 1. *Let G' be a maximal internally 4-connected plane graph with 4 exterior vertices v_1, v_4, v_2, v_3 in ccw order. Then:*

1. *G' admits a transversal structure $\mathcal{T}(G')$, which is computable in linear time.*
2. *Applying $\mathcal{T}(G')$, for each vertex v, embed it in the grid point $(x(v), y(v))$. For each edge of G', simply connect its end vertices by a straight line. The drawing is a straight-line grid drawing for G'. Its drawing size is $x(\mathcal{T}(G')) \times y(\mathcal{T}(G'))$. This drawing is computable in linear time.*

Fig. 2 (1) presents a straight-line grid drawing of the graph of G' in Fig. 1 (2), by applying Lemma 1 to $\mathcal{T}(G')$ in Fig. 1 (2).

3 Transformation from Realizers to Transversal Structures and Its Application in Planar Polyline Drawing

Let G be a maximal plane graph with 3 exterior vertices v_1, v_2, v_3 in ccw order. Let $\mathcal{R}(G) = \{T_1, T_2, T_3\}$ be one of its realizers. T_i is rooted at v_i. Next, we illustrate how to transform a realizer for G to a transversal structure for a targeted maximal internally 4-connected plane graph G' with 4 exterior vertices. Our transformation uses a tree from $\mathcal{R}(G)$. Subject to a color and index rotation, we only need to show the case of using T_3. Let v be a leaf node of T_3. v is an interior vertex of G. Let $p_1(v)$ and $p_2(v)$ be its parents in T_1 and T_2 respectively. The face f enclosed by $\{v, p_1(v), p_2(v)\}$ is an interior face of G. Consider the edge $e = (p_1(v), p_2(v))$. According to the property of realizer, e cannot be in T_3. Furthermore, e cannot be (v_1, v_3), neither can it be (v_2, v_3).

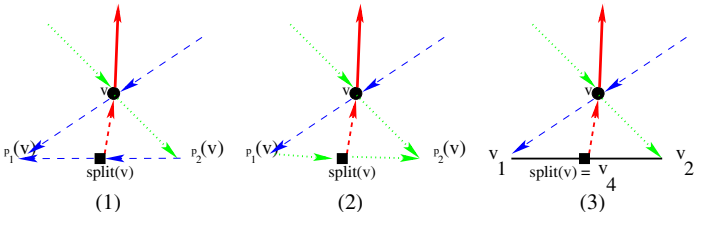


Fig. 3. Step 1

We complete the transformation in the following three steps. We will use G' to denote both the target graph and the intermediate forms.

Step 1: For every leaf node v of T_3, insert a vertex $split(v)$ in the middle of $e = (p_1(v), p_2(v))$. $split(v)$ splits e into two edges. Let the two edges keep the original color and directions as e in G. Add a directed edge from $split(v)$ to v,

and color it by red. We have three different cases, as illustrated in (1), (2) and (3) of Fig. 3. Note that, in Fig. 3 (3), for the case where $e = (v_1, v_2)$, we denote the inserted vertex by v_4. v_4 is an exterior vertex of G'. G' has 4 exterior vertices v_1, v_4, v_2, v_3 in ccw order.

Step 2: For each leaf v of T_3, still consider the edge $e = (p_1(v), p_2(v))$, as if it were not split. There are three cases to consider:

Case 1: $e = (v_1, v_2)$. No additional operation needed.

Case 2: e is in T_1. e is adjacent to another triangle g. Let u be the vertex $\notin \{p_1(v), p_2(v)\}$ in g. According to the properties of realizer, only five scenarios are possible. They are shown in Fig. 4. In Fig. 4 (1) or (2), we add a directed edge from u to $split(v)$, and color it by red. In Fig. 4 (3) or (4), consider $p_2(v)$, it must also be a leaf in T_3. Therefore, in Step 1, a vertex $split(p_2(v))$ and an edge $(split(p_2(v)), p_2(v))$ have been inserted for it already. In this step, we further add an edge, directed from $split(p_2(v))$ to $split(v)$, and color it by red. Fig. 4 (5) is similar to Fig. 4 (3) or (4) except that $u = v_2$.

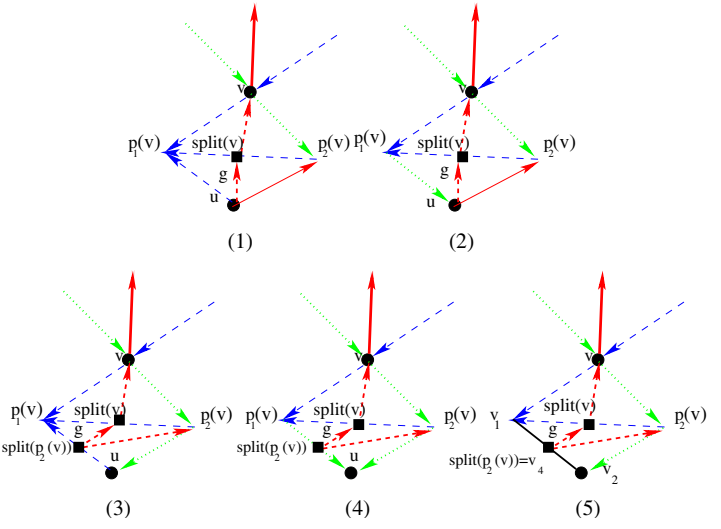


Fig. 4. Case 2 of Step 2

Case 3: e is in T_2. This case is similar to Case 2.

Step 3: Reverse the direction of the blue-colored edges in G'. Recolor the green-colored edges by blue.

The above coloring and directions of the edges of G' is denoted by $\mathcal{T}_3(G')$. (If we use T_1, T_2 instead, then we denote it by $\mathcal{T}_1(G'), \mathcal{T}_2(G')$ instead). The proof of the following lemma is omitted here due to space limitation.

Lemma 2. *Let G be a maximal plane graph with n vertices. v_1, v_2, v_3 be its exterior vertices in ccw order. $\mathcal{R}(G) = \{T_1, T_2, T_3\}$ be one of its realizers. T_i is*

rooted at v_i. Let l_i be the number of leaves of T_i, $i \in \{1, 2, 3\}$. Then for the above introduced transformation:

1. *$\mathcal{T}_i(G')$ is a transversal structure of G'. $x(\mathcal{T}_i(G')) = l_i + 1$ and $y(\mathcal{T}_i(G')) = (n-2)$, where $i \in \{1, 2, 3\}$.*
2. *The transformation from the realizer $\mathcal{R}(G)$ to $\mathcal{T}_i(G')$, $i \in \{1, 2, 3\}$ can be done in linear time.*

For the maximal plane graph G in Fig. 1 (1), Fig. 1 (2) shows a transversal structure $\mathcal{T}_3(G')$, constructed as above by using T_3 in the realizer $\mathcal{R}(G) = \{T_1, T_2, T_3\}$. The inserted vertices are represented by black squares. The inserted red-colored edges are drawn in dashed lines.

Applying Lemma 1 to $\mathcal{T}_i(G')$, we obtain a straight-line grid drawing of G'. By removing the inserted edges and the inserted vertices, but keeping the split edges in the drawing of G', it becomes a polyline drawing of G. It is easy to see that, only an edge in G which has had a vertex inserted in it during the transformation maybe drawn as two-segment polylines. The total number of such edges is l_i, i.e., the number of leaves in T_i. In [2], Bonichon et al. proved that in any realizer, $l_1 + l_2 + l_3 \leq (2n-5)$. Combined with Lemma 2, we have the following theorem:

Theorem 1. *A plane graph G with n vertices admits a polyline drawing in a grid with size bounded by $(p+1) \times (n-2)$, where $p \leq \lfloor \frac{2n-5}{3} \rfloor$. The number of bends is at most p, and each edge has at most one bend. The drawing can be constructed in linear time.*

Fig. 2 (2) shows a polyline drawing of the original graph G in Fig. 1 (1), where the edges represented as two-segment polylines are drawn in dashed lines.

References

1. di Battista, G., Eades, P., Tamassia, R., Tollis, I.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, Englewood Cliffs (1998)
2. Bonichon, N., Le Saëc, B., Mosbah, M.: Wagner's theorem on realizers. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 1043–1053. Springer, Heidelberg (2002)
3. Bonichon, N., Le Saëc, B., Mosbah, M.: Optimal area algorithm for planar polyline drawings. In: Kučera, L. (ed.) WG 2002. LNCS, vol. 2573, pp. 35–46. Springer, Heidelberg (2002)
4. Fusy, É.: Transversal structures on triangulations, with application to straight-line drawing. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 177–188. Springer, Heidelberg (2006)
5. He, X.: On finding the rectangular duals of planar triangular graphs. SIAM Journal on Computing 22, 1218–1226 (1993)
6. Schnyder, W.: Planar graphs and poset dimension. Order 5, 323–343 (1989)
7. Schnyder, W.: Embedding planar graphs on the grid. In: Proc. of the First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 138–148. SIAM, Philadelphia (1990)

Constrained Stress Majorization Using Diagonally Scaled Gradient Projection

Tim Dwyer and Kim Marriott

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

Abstract. Constrained stress majorization is a promising new technique for integrating application specific layout constraints into force-directed graph layout. We significantly improve the speed and convergence properties of the constrained stress-majorization technique for graph layout by employing a diagonal scaling of the stress function. Diagonal scaling requires the active-set quadratic programming solver used in the projection step to be extended to handle separation constraints with scaled variables, i.e. of the form $s_i y_i + g_{ij} \leq s_j y_j$. The changes, although relatively small, are quite subtle and explained in detail.

Keywords: constraints, graph layout.

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 minimize 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 $\frac{1}{{d_{ij}}^2}$, 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.

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 [4]. The basic idea is to modify the iterative step in *stress majorization* [5, Ch. 8] to solve a one-dimensional quadratic objective subject to the separation constraints for that dimension. Separation constraints allow a wide variety of aesthetic requirements—such as downward-pointing edges in directed graphs, alignment or distribution of nodes, placement of nodes in horizontal or vertical bands, non-overlap of nodes, orthogonal ordering between nodes, containment of nodes in clusters, containment in a page, and edge straightening without

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 219–230, 2007.

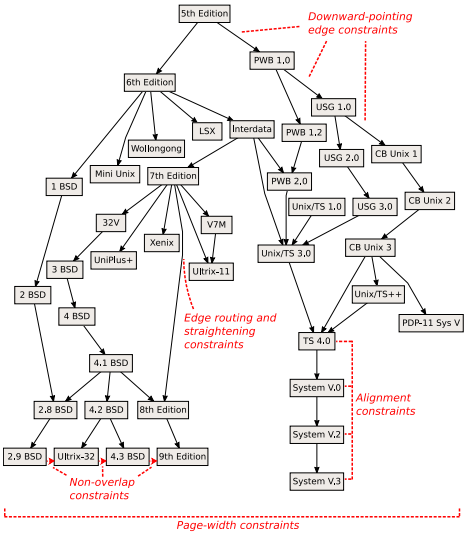


Fig. 1. Drawing of a directed graph illustrating the flexibility of constrained stress majorization. Separation constraints encode the aesthetic requirements that: (1) directed edges point downwards; (2) selected nodes are horizontally or vertically aligned; (3) the drawing fits within the page boundaries; and (4) nodes do not overlap edges or other nodes. The "history of unix" graph data is from `http://www.graphviz.org` and this drawing originally appeared in [3].

introduction of additional edge crossings–to be integrated into force-directed layout [4]. Thus, *constrained stress majorization* provides an extremely flexible basis for handling application specific layout conventions and requirements.

In majorization the value of the stress function (1) is reduced by alternately minimizing quadratic functions in the horizontal and vertical axes that bound the stress functions. These quadratic functions have the form:

$$f(x) \equiv \frac{1}{2}x^T Qx + x^T b \tag{2}$$

where, for a graph with n nodes, x is the n dimensional vector of node positions in the current dimension; the $n \times n$ Hessian matrix Q is the graph Laplacian (see below); and the linear term b is 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 [6]). Constrained stress majorization extends this by additionally requiring that the solution returned satisfies the separation constraints for that dimension.

In [4] we gave a specialized gradient-projection-based method for solving this particular kind of quadratic program (QP) which was significantly faster than standard QP algorithms. However, gradient projection (GP) based methods, like other iterative optimization methods based on steepest descent, can display poor convergence when working with badly conditioned Hessian matrices. A standard

technique to improve convergence is to scale the variables so that the diagonal entries of the scaled Hessian matrix are all equal. This works particularly well if the Hessian, with entries Q_{ij}, is diagonally dominant, i.e. $|Q_{ii}| \geq \sum_{j \neq i} |Q_{ij}|$, which the graph Laplacian is by its definition:

$$Q_{ij} = \begin{cases} -w_{ij} & i \neq j \\ \sum_{k \neq i} w_{ij} & i = j \end{cases}$$

The main contribution of this paper is to demonstrate that using such diagonal scaling with GP is nearly twice as fast as the original unscaled GP algorithm and, even more importantly, the rate of convergence is more robust. The main technical difficulty is the need to modify the projection step to handle constraints of the form $s_i x_i + g \leq s_j x_j$ where s_i and s_j are the positive scaling factors for x_i and x_j, respectively. We detail the necessary modifications to the projection algorithm. Although these modifications are quite subtle, they make little difference to the implementation difficulty. Thus, there seems no reason to use the original unscaled GP algorithm in preference to the GP algorithm with diagonal scaling presented here. Another contribution of the paper is to provide more details of the gradient projection algorithm presented in [4].

2 Diagonally-Scaled Gradient Projection

The core step in constrained stress majorization is to solve a quadratic program with an objective of the form given in Equation 2 subject to some separation constraints $c \in C$ on the variables where each separation constraint c is of form $x_i + g \leq x_j$ where g is the minimum gap between the variables x_i and x_j. We call this the Quadratic Programming with Separation Constraints (QPSC) problem.

Previously we gave an iterative *gradient-projection* algorithm for solving a QPSC problem [4]. This works by first decreasing $f(x)$, by moving x in the direction of steepest descent, i.e. opposite to the gradient $\nabla f(x) = Qx + b$. While this guarantees that—with appropriate selection of step-size α—the stress is decreased by this first step, the new positions may violate the constraints. This is corrected by applying the function *project*, which returns the closest point $\bar{x}$ to x which satisfies the separation constraints, i.e. it projects x on to the feasible region. A vector p from the initial position $\hat{x}$ to $\bar{x}$ is then computed and the algorithm ensures monotonic decrease in stress when moving in this direction by computing a second stepsize $\beta = \arg\min_{\beta \in [0,1]} f(x + \beta p)$ which minimizes stress in this interval.

Unfortunately, GP-based methods, like other iterative methods based on steepest descent, can display poor convergence when working with poorly conditioned Hessian matrices. One remedy is to perform a linear scaling on the problem. The basic idea is to use an $n \times n$ scaling matrix S and transform the problem into one on new variables y s.t. $x = Sy$.

If we choose $S = Q^{-1} = (\nabla^2 f(x))^{-1}$ then steepest descent on the transformed problem is equivalent to performing Newton's method on the original problem.

Thus, at least in the unconstrained problem convergence will be quadratic. However, computing the inverse of Q is quite expensive and it also means that scaling of the separation constraints results in full-fledged linear constraints, so that the projection operation becomes considerably more complex and expensive.

Thus, an approach which approximates Q^{-1} is often used in practice [7]. Specifically, we choose S to be a diagonal matrix such that the diagonal entries in S^TQS are all 1, i.e. $S_{ii} = \frac{1}{\sqrt{Q_{ii}}}$ and for $i \neq j$, $S_{ij} = 0$. This is called diagonal scaling. We refer below to the diagonal entries in S as $s_i = S_{ii}$. Note that for all i, $s_i > 0$ and, clearly, S is very quick to compute.

It is straightforward to change the main gradient-projection routine, *solve_QPSC*, from [4] to use diagonal scaling. The modified routine is given in Fig. 2.

The chief difficulty is modifying the projection routine *project* called by *solve_QPSC*. We have that $x_i = s_i y_i$ so a separation constraint of form $x_i + g \leq x_j$ becomes, in the scaled space, $s_i y_i + g \leq s_j y_j$. We call such linear inequalities *positively scaled separation constraints*.

After computing an unconstrained descent direction the scaled GP algorithm calls *project* to find the nearest point to $d = \hat{y} - \alpha g$ satisfying the positively scaled separation constraints C'. That is, it must solve:

$$\begin{array}{l}\text{minimize } F(y) = \sum_{i=1}^{n}(y_i - d_i)^2 \\ \text{subject to positively scaled separation constraints } C'\end{array}$$

In [4] we described an *active-set* algorithm for incrementally finding a solution to the projection problem subject to (unscaled) separation constraints. Here we extend this to handle positively scaled separation constraints. Although the changes are minor, they are quite subtle. The complete algorithm is given in Fig. 2. Note that if c is a positively scaled separation constraint of form $su + g \leq tv$ we refer to u, v, s, t and g by lv_c, rv_c, ls_c, rs_c, gap_c, respectively.

The method works by building up *blocks* of variables spanned by a tree of *active* (or set at equality) constraints. At any point in time the block to which a variable y_i belongs is given by blk_i. If a block has k variables the tree of active constraints has $k - 1$ linear equations so variable elimination can be used to eliminate all but one variable and the position of all other variables is a linear function of that single unknown *reference variable*. This contrasts to the unscaled case in which the variables are simple offsets from the reference variable and are not scaled.

For each block B the algorithm keeps: the set of variables V_B in the block; the set of active constraints C_B; the current position Y_B of the block's reference variable; and the scaling factor S_B for the reference variable. For each variable y_i in block $B = blk_i$ we have a variable dependent scaling factor a_i and offset b_i giving its position relative to Y_B, i.e. it is an invariant that $y_i = a_i Y_{blk_i} + b_i$. As we shall see it is also an invariant that $a_i = \frac{S_{blk_i}}{s_i}$.

Each block B is placed at the position minimizing $F = \sum_{i \in V_B}(y_i - d_i)^2$ subject to the active constraints C_B. Now,

$$\frac{\partial F}{\partial Y_B} = \sum_{i \in V_B} \frac{\partial y_i}{\partial Y_B} \frac{\partial F}{\partial y_i} = \sum_{i \in V_B} a_i \frac{\partial F}{\partial y_i} = \sum_{i \in V_B} a_i(2(y_i - d_i)) = \sum_{i \in V_B} 2a_i(a_i Y_B + b_i - d_i)$$

The minimum occurs when $\frac{\partial F}{\partial Y_B} = 0$ so the optimum value is $Y_B = \frac{AD_B - AB_B}{A2_B}$ where $AD_B = \sum_{i \in V_B} a_i d_i$, $AB_B = \sum_{i \in V_B} a_i b_i$, and $A2_B = \sum_{i \in V_B} a_i^2$.

Initially, each variable y_i is placed in its own block B_i where it is the block's reference variable. This is done in the procedure *init_blocks* called at the start of *solve_QPSC*. After this the blocks persist between the calls to *project* and are incrementally modified in the routine *project*.

The function *project*(C, d) works as follows. First the routine *split_blocks* updates the position of each block B to reflect the changed value of d. The routine then splits the block if this will allow the solution to be improved. The procedure *split_block* is straightforward. The only point to note is that we define $left(c, B)$ to be the variables in V_B connected to the variable lv_c by constraints in $C_B \setminus \{c\}$ and we define $right(c, B)$ symmetrically.

Determining where and when to split a block is a little more difficult. It is formalized in terms of Lagrange multipliers. Recall that if we are minimizing function F with a set of convex equalities C over variables y, then we can associate a variable λ_c called the Lagrange multiplier with each $c \in C$. Given a configuration y^* feasible with respect to C we have that y^* is a locally minimal solution iff there exist values for the Lagrange multipliers satisfying for each y_i that

$$\frac{\partial F}{\partial y_i}(y^*) = \sum_{c \in C} \lambda_c \frac{\partial c}{\partial y_i}(y^*) \tag{3}$$

Furthermore, if we also allow inequalities, the above statement continues to hold as long as $\lambda_c \geq 0$ for all inequalities c of form $t \geq 0$. By definition an inequality c which is not active has $\lambda_c = 0$. Thus we need to split a block at active constraint c if $\lambda_c < 0$ since this tells us that by moving the two sub-blocks apart we can improve the solution.

One key to the efficiency of the projection algorithm is that the Lagrange multipliers can be computed efficiently for the active constraints in a block in linear time using the procedure *comp_dfdv*. The justification for this is the following lemma which is proved in [8]:

Lemma 1. *Let y^* place all blocks at their optimum position. If c is an active constraint in block B then*

$$\lambda_c \quad = \quad - \sum_{k \in left(c,B)} \frac{1}{s_k} \frac{\partial F}{\partial y_k}(y_k^*) \quad = \quad \sum_{k \in right(c,B)} \frac{1}{s_k} \frac{\partial F}{\partial y_k}(y_k^*)$$

Of course, after moving the blocks to their new location and perhaps splitting some blocks, there is no guarantee that the placement satisfies all of the constraints. Thus, after splitting the procedure *project* repeatedly modifies the blocks until a feasible solution is reached. The constraints are processed in decreasing order of violation until no more violated constraints are found and therefore a feasible solution has been obtained.

procedure *solve_QPSC*($Q,\ b,\ C,\ x$)
$s \leftarrow (\frac{1}{\sqrt{Q_{11}}}, \frac{1}{\sqrt{Q_{22}}}, \ldots, \frac{1}{\sqrt{Q_{nn}}})$
$S \leftarrow n \times n$ diagonal matrix with $S_{ii} = s_i$
global $y \leftarrow Sx$
init_blocks()
$Q' \leftarrow S^T Q S$
$b' \leftarrow Sb$
$C' \leftarrow \{s_i y_i + g \leq s_j y_j | (x_i + g \leq x_j) \in C\}$
repeat
  $g \leftarrow Q'y + b'$
  $\alpha \leftarrow \frac{g^T g}{g^T Q' g}$
  $\hat{y} \leftarrow y$
  $d \leftarrow \hat{y} - \alpha g$
  $nosplit \leftarrow project(C', d)$
  $\bar{y} \leftarrow y$ (y modified by *project*)
  $p \leftarrow \hat{y} - \bar{y}$
  $\beta \leftarrow \min(\frac{g^T d}{d^T Q' p}, 1)$
  $y \leftarrow \hat{y} - \beta p$
until $\|\hat{y}, y\|$ sufficiently small and $nosplit$
return $S^{-1}y$

function *project*(C, d)
$nosplit \leftarrow split\_blocks(d)$
$c \leftarrow \max_{c \in C}\ violation(c)$
while $violation(c) \geq 0$ **do**
  if $blk_{lv_c} \neq blk_{rv_c}$ **then**
    merge_block(c)
  else *expand_block*(c)
  $c \leftarrow \max_{c \in C}\ violation(c)$
return $nosplit$

procedure *init_blocks*()
for $i = 1, ..., n$ **do**
  let B_i be a new block s.t.
  $V_{B_i} \leftarrow \{i\}$
  $Y_{B_i} \leftarrow y_i$
  $S_{B_i} \leftarrow s_i$
  $AD_{B_i} \leftarrow y_i$
  $A2_{B_i} \leftarrow 1$
  $AB_{B_i} \leftarrow 0$
  $C_{B_i} \leftarrow \emptyset$
  $a_i \leftarrow 1$
  $b_i \leftarrow 0$
  $blk_i \leftarrow B_i$
return

procedure *split_blocks*(d)
$nosplit \leftarrow true$
for each active block B **do**
  $AD_B \leftarrow \sum_{i \in V_B} a_i d_i$
  $AB_B \leftarrow \sum_{i \in V_B} a_i b_i$
  $A2_B \leftarrow \sum_{i \in V_B} a_i^2$
  $Y_B \leftarrow \frac{AD_B - AB_B}{A2_B}$
  for $i \in V_B$ **do**
    $y_i \leftarrow a_i Y_B + b_i$
  for each $c \in C_B$ **do** $\lambda_c \leftarrow 0$
  choose $v \in V_B$
  $comp\_dfdv(v, C_B, NULL)$
  $sc \leftarrow \min_{c \in C_B} \lambda_c$
  if $\lambda_c \geq 0$ **then break**
  $nosplit \leftarrow false$
  $split\_block(c)$
return $nosplit$

function *violation*(c) =
let $c \equiv s_i y_i + g \leq s_j y_j$ in
$s_j y_j - (s_i y_i + g)$

procedure *merge_block*(c)
let $c \equiv s_i y_i + g \leq s_j y_j$
$LB \leftarrow blk_i$
$RB \leftarrow blk_j$
for $k \in V_{RB}$ **do**
  $blk_k \leftarrow LB$
  $a_k \leftarrow S_{LB}/s_k$
  $b_k \leftarrow b_k + g$
  $AB_{LB} \leftarrow AB_{LB} + a_k b_k / s_k$
  $AD_{LB} \leftarrow AD_{LB} + a_k d_k$
  $A2_{LB} \leftarrow AD_{LB} + a_k a_k$
$Y_{LB} \leftarrow \frac{AD_{LB} - AB_{LB}}{A2_{LB}}$
$C_{LB} \leftarrow C_{LB} \cup C_{RB} \cup \{c\}$
$V_{LB} \leftarrow V_{LB} \cup V_{RB}$
for $i \in V_{LB}$ **do**
  $y_i \leftarrow a_i Y_B + b_i$
return

procedure *expand_block*($\tilde{c}$)
$B \leftarrow blk_{lv_{\tilde{c}}}$
for each $c \in C_B$ **do** $\lambda_c \leftarrow 0$
$comp\_dfdv(lv_{\tilde{c}}, C_B, NULL)$
$[v_1, ..., v_k] := comp\_path(lv_{\tilde{c}}, rv_{\tilde{c}}, C_B)$
$ps \leftarrow \{c \in C_B \mid \exists j \text{ s.t. } lc_c = v_j \text{ and } rc_c = v_{j+1}\}$
if $ps = \emptyset$ **then error** % constraints unsatisfiable
$sc \leftarrow \min_{c \in ps} \lambda_c$
$split\_block(sc)$
$merge\_block(\tilde{c})$
return

procedure *split_block*(c)
$B \leftarrow blk_{lv_c}$
let RB be a new block s.t.
$S_{RB} \leftarrow S_B$
$V_{RB} \leftarrow left(c, B)$
$C_{RB} \leftarrow \{c' \in C_B \mid lv_{c'}, rv_{c'} \in V_{RB}\}$
for $i \in V_{RB}$ **do** $blk_i \leftarrow RB$
$AD_{RB} \leftarrow \sum_{i \in V_{RB}} a_i d_i$
$AB_{RB} \leftarrow \sum_{i \in V_{RB}} a_i b_i$
$A2_{RB} \leftarrow \sum_{i \in V_{RB}} a_i^2$
$Y_{RB} \leftarrow \frac{AD_{RB} - AB_{RB}}{A2_{RB}}$
for $i \in V_{RB}$ **do** $y_i \leftarrow a_i Y_{RB} + b_i$
let LB be a new block s.t.
symmetric construction to RB
return

function *comp_dfdv*($i, AC, \tilde{c}$)
$dfdv \leftarrow \frac{2}{s_i}(y_i - d_i)$
for each $c \in AC$ s.t. $i = lv_c$ and $c \neq \tilde{c}$ **do**
  $\lambda_c \leftarrow comp\_dfdv(rv_c, AC, c)$
  $dfdv \leftarrow dfdv + \lambda_c$
for each $c \in AC$ s.t. $i = rv_c$ and $c \neq \tilde{c}$ **do**
  $\lambda_c \leftarrow -comp\_dfdv(lv_c, AC, c)$
  $dfdv \leftarrow dfdv - \lambda_c$
return $dfdv$

Fig. 2. Diagonal scaling Gradient-Projection-based algorithm to find an optimal solution to a QPSC problem with variables $x_1, \ldots, x_n$, symmetric positive-semidefinite matrix Q, vector b and separation constraints C over the variables

If a constraint c is violated there are two cases. Either the variables in c, lv_c and rv_c, belong to different blocks, in which case *merge_block* is used to merge the two blocks, or else lv_c and rv_c, belong to the same block, in which case *expand_block* is used to modify the block.

The code for *merge_block* is relatively straightforward. If the merge is because of the violated constraint $c \equiv s_i y_i + g \leq s_j y_j$ then it merges the block $RB = blk_j$ into block $LB = blk_i$ (the direction is arbitrary and in practice we always move variables from the smaller to the larger block). The reference variable Y_{LB} becomes the reference variable of the new block. Now, rewriting the active version of c, $s_j y_j = s_i y_i + g$, in terms of Y_{LB} and Y_{RB} gives $s_j(a_j Y_{RB} + b_j) = s_i(a_i Y_{LB} + b_i) + g$. Thus,

$$Y_{RB} = \frac{s_i a_i}{s_j a_j} Y_{LB} + \frac{s_i b_i - s_j b_j + g}{s_j a_j} = \frac{S_{LB}}{S_{RB}} Y_{LB} + \frac{s_i b_i - s_j b_j + g}{S_{RB}}.$$

Taking $a = \frac{S_{LB}}{S_{RB}}$ and $b = \frac{s_i b_i - s_j b_j + g}{S_{RB}}$, we can express the variables of RB in terms of the reference variable $Y_B = Y_{LB}$ as:

$$y_k = a_k Y_{RB} + b_k = a_k(aY_{LB} + b) + b_k = a'_k Y_{LB} + b'_k$$

where

$$a'_k = (a_k a) = \frac{S_{RB}}{s_k}\frac{S_{LB}}{S_{RB}} = \frac{S_{LB}}{s_k} = \frac{S_B}{s_k}$$

and $b'_k = a_k b + b_k$.

The procedure *expand_block*$(b, \tilde{c})$ is probably the most complex part of the algorithm. It deals with a case where a previously constructed block now causes a constraint $\tilde{c}$ between two variables in the block to be violated. To fix this we must identify where to split the current block and then rejoin the sub-blocks using $\tilde{c}$, in effect expanding the block to remove the violation by choosing a different spanning tree of active constraints for the block. To do so, the algorithm computes the best constraint *sc* in the active set on which to split based on its Lagrange multiplier, λ_c. The intuition for this is that the value of λ_c gives the rate of increase of the goal function as a function of c_{gap}. Thus, the smaller the value of λ_c the better it is to split the block at that constraint. However, not all constraints in the active set are valid points for splitting. Clearly we must choose a constraint that is on the path between the variables $lv_{\tilde{c}}$ and $rv_{\tilde{c}}$. The call to the function *comp_path* returns the list of variables $[v_1, ..., v_k]$ on this path. Furthermore, to be a valid split point the constraint c must be oriented in the same direction as $\tilde{c}$, i.e. for some j, $lc_c = v_j$ and $rc_c = v_{j+1}$. If there are no such constraints then the constraints (and the original separation constraints) are infeasible so the algorithm terminates with an error. Otherwise, the split constraint *sc* is simply the valid split constraint with the least Lagrange multiplier. The remainder of *expand_block* uses *split_block* to split the block by removing *sc* from the active set C_B and then uses *merge_block* to rejoin the two sub-blocks with constraint $\tilde{c}$.

Clearly, *project* will only terminate if either no constraints are violated, or *expand_block* terminates with an error. We show that if *expand_block* gives rise

to an error then the original separation constraints are unsatisfiable. It gives rise to an error if there is a scaled constraint $\tilde{c}$ of form $s'_1 v_1 + g \leq s'_n v_n$ and a path of active constraints from v_1 to v_n of form

$$s'_2 v_2 + g_1 \leq s'_1 v_1, s'_3 v_3 + g_2 \leq s'_2 v_2, ..., s'_n v_n + g_{n-1} \leq s'_{n-1} v_{n-1}$$

since the orientation of the constraints is opposite that of $\tilde{c}$. Thus, a consequence of the path constraints is that $s'_n v_n + g' \leq s'_1 v_1$ where $g' = \sum_{i=1}^{n-1} g_i$. The current placement of v_1 and v_n satisfies $s'_n v_n + g' = s'_1 v_1$ but does not satisfy $s'_1 v_1 + g \leq s'_n v_n$. Thus $s'_n v_n + g' = s'_1 v_1$ and $s'_1 v_1 + g > s'_n v_n$ and so $s'_n v_n + g' + g > s'_n v_n$. Thus, $g + g' > 0$ and so the original scaled constraints are unsatisfiable since $s'_n v_n + g' \leq s'_1 v_1$ and $g + g' > 0$ implies $s'_1 v_1 + g > s'_n v_n$ which contradicts $s'_1 v_1 + g \leq s'_n v_n$. This also means the original separation constraints are unsatisfiable since we have in the unscaled space $x'_1 + g \leq x'_n$ and $x'_1 + g' \geq x'_n$.

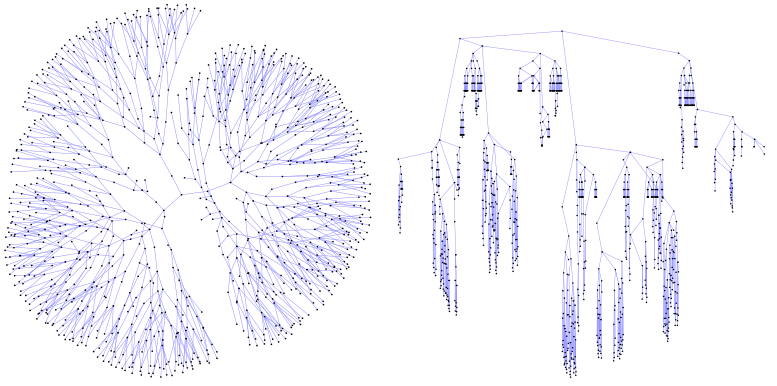
Thus, *project* always returns a feasible solution if one exists. The feasible solution is optimal in the case that *nosplit* is true and the solution has not changed. Thus although the call to *project* is not initially guaranteed to return the optimal solution it will converge towards it. Using this it is relatively straightforward to show that *solve_QPSC* converges towards the optimal solution.

Unfortunately, as for the unscaled gradient projection algorithm, we have yet to provide a formal proof of termination of the *project* function, though we conjecture that it does always terminate. The potential problem is that a constraint may be violated, added to the active set, then removed from the active set due to block expansion, and then re-violated because of other changes to the block. Note that we have tried thousands of very different examples and have never encountered non-termination.

Another potential source of non-termination, which arises in most active set approaches, is that it may be possible for the algorithm to cycle by removing a constraint because of splitting, and then be forced to add the constraint back again. This can only occur if the original problem contains constraints that are redundant in the sense that the set of equality constraints corresponding to the separation constraints C, namely $\{u + a = v \mid (u + a \leq v) \in C\}$, contains redundant constraints. We could remove such redundant separation constraints in a pre-processing step by adding ϵ^i to the gap for the i^{th} separation constraint or else use a variant of lexico-graphic ordering to resolve which constraint to make active in the case of equal violation. We can then show that cycling cannot occur. In practice however we have never found a case of cycling.

3 Results

To investigate the effect of diagonally scaled gradient projection on running time and convergence of constrained stress-majorization layout, we compared it against a number of other optimization methods for various graphs with a range of degree distributions. Table 3 gives results in terms of running times and numbers of iterations for a selection of graphs all of size around $|V| = 1000$. The optimization methods tested were:



(a) Unconstrained (b) Constrained

Fig. 3. A randomly generated tree as used in our tests, with 1071 nodes of varying degree, drawn with and without constraints. The vertical constraints enforce downward-pointing edges while the horizontal constraints are simply generated by an in-order traversal of the tree.

CG. Unconstrained conjugate gradient (as recommended by [6] for (unconstrained) functional majorization).

Int. Pnt. A commercially available QP solver based on the interior point method (Mosek[1]).

Unscaled GP. Gradient projection without scaling.

Scaled GP. Gradient projection with scaling.

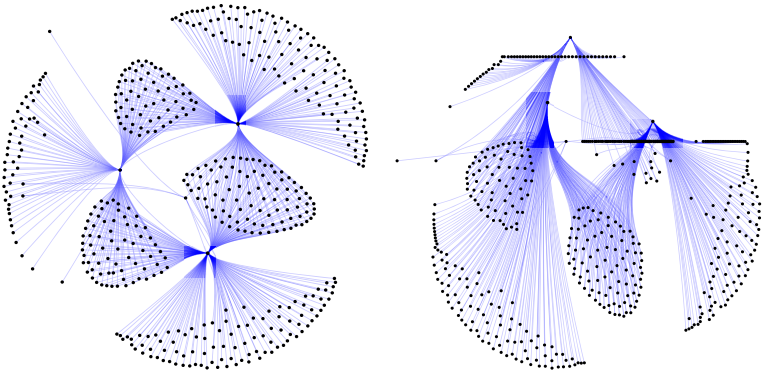
Four different graphs were chosen with a range of different node-degree distributions. The graphs were a randomly generated tree with $|V| = 1071$ and node degree ranging from 1 to 4 (Fig. 3); an Erdõs–Rényi random graph of poisson degree distribution [9] and $|V| = 1000$; a random graph with power-law degree distribution generated using the Barabási–Albert model [10] (e.g. Fig. 4); and a graph from the Matrix Market[2] that we have used before in performance testing of constrained layout methods [4].

For all methods except CG (which can not easily be extended to handle constraints) we ran both with and without a basic set of downward pointing edge constraints [4]. For the tree graph we also included ordering constraints over the x-node positions based on a simple in-order traversal of the graph. The constraints were chosen to be simple to generate, easy to visually verify, and to be similar to the types of constraints that might be useful in practical layout situations.

Numbers of stress-majorization iterations are given for each graph, with and without constraints. These are the same across all solvers since each solves the quadratic-program subproblems to optimality. For CG and GP solver methods we also give the total number of iterations required. This helps to explain the

[1] `http://www.mosek.org`

[2] `http://math.nist.gov/MatrixMarket/`



(a) Unconstrained (b) Constrained

Fig. 4. A randomly generated scale-free graph as used in our tests. It has 500 nodes with power-law distribution of degree and is drawn with and without vertical downward-pointing edge constraints.

Table 1. Results of applying stress majorization using various different techniques to solve the quadratic problems at each iteration

Graph	Constraints Hor.	Vert.	Solver	Stress Maj. Iterations	Total Iterations	Total time (secs)
Random	0	0	CG	48	646	8.82
Tree	0	0	Int. Pnt.	48	N/A	42.69
	0	0	Unscaled GP	48	1607	19.97
$\|V\| = 1071$	0	0	Scaled GP	48	833	13.88
	1070	1070	Int. Pnt.	38	N/A	341.69
	1070	1070	Unscaled GP	38	2650	33.12
	1070	1070	Scaled GP	38	1071	17.31
Poisson	0	0	CG	83	908	12.52
random	0	0	Int. Pnt.	83	N/A	62.17
	0	0	Unscaled GP	83	1907	23.51
$\|V\| = 1000$	0	0	Scaled GP	83	1244	19.34
	0	1478	Int. Pnt.	46	N/A	175.93
	0	1478	Unscaled GP	46	2336	20.88
	0	1478	Scaled GP	46	1717	15.81
Power-law	0	0	CG	91	983	13.45
random	0	0	Int. Pnt.	91	N/A	68.21
	0	0	Unscaled GP	91	2140	26.3
$\|V\| = 1000$	0	0	Scaled GP	91	1287	20.43
	0	1598	Int. Pnt.	101	N/A	390.07
	0	1598	Unscaled GP	101	1914	48.9
	0	1598	Scaled GP	101	1717	28.21
Bus 1138	0	0	CG	48	848	10.58
	0	0	Int. Pnt.	48	N/A	49.08
$\|V\| = 1138$	0	0	Unscaled GP	48	1904	25.03
	0	0	Scaled GP	48	875	16.49
	0	1458	Int. Pnt.	43	N/A	190.06
	0	1458	Unscaled GP	43	2697	36.12
	0	1458	Scaled GP	43	1148	19.97

differences in running time between the different methods. Without constraints CG was clearly fastest, solving the problem in fewer iterations and having to do slightly less work in each iteration. This is to be expected since CG is known to

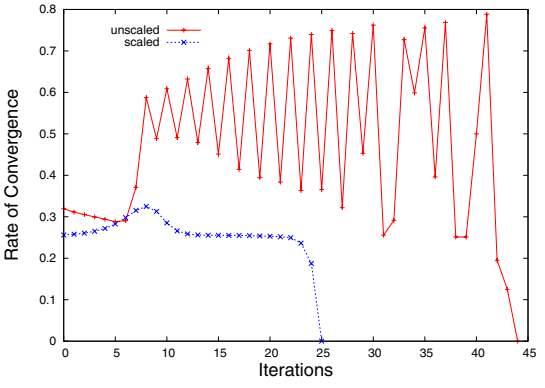


Fig. 5. Rate of convergence $\frac{|x_{k+1}-x^*|}{|x_k-x^*|}$ shown for each iteration k of the first gradient-projection iterate when applying stress majorization to the 1138bus graph. Note that x^* is simply taken as the final configuration before the threshold is reached so the final tail-off in both curves should be disregarded.

achieve super-linear convergence. Of the remaining methods, across all graphs, constrained or not scaled GP was the fastest (converging in significantly fewer iterations), followed by unscaled GP and the interior point method was slowest by several fold. In all cases scaling improved the running time by at least 20%. Interestingly, the improvement in speed in GP when scaling was applied was more marked when constraints were also solved, e.g. for the tree example it was almost twice as fast. To check scalability we also repeated the tests for random graphs between 50 and 2000 nodes and the speed improvement observed with scaling remained a fairly constant factor between 1.5 and 2.

Fig. 5 gives a graphic explanation of how scaling improves the convergence of the GP method. The figure shows rate of convergence by iteration for the first QP solved by the GP method in a stress majorization layout of the 1138bus graph. Convergence rate is, as usual, defined as the distance from an optimal solution at iteration $k+1$ divided by the distance at iteration k. As shown in Fig. 2 we stop the GP procedure when the descent vector has length smaller than some threshold τ and for this test, to ensure a reasonable number of iterations we set τ very small (10^{-15}). With scaling convergence is roughly constant and the threshold is reached after 25 iterations. Without scaling, the convergence rate oscillates and the threshold is not reached until 44 iterations.

4 Conclusion

Constrained stress majorization is a promising new technique for integrating application specific layout constraints into force-directed graph layout. The method previously suggested for solving the special kind of quadratic program arising in constrained stress majorization is a specialized gradient projection algorithm for separation constraints. We have demonstrated that by performing diagonal

scaling on the quadratic programming and generalizing the projection algorithm to handle positively scaled separation constraints, we can significantly improve the speed and convergence properties of the constrained stress-majorization technique. Importantly, this improvement comes at very little extra implementation effort. Thus, we believe that gradient projection with diagonal scaling is the method of choice for solving constrained stress majorization.

Our results have greater scope than graph layout since constrained stress majorization is immediately applicable to constrained multidimensional scaling (as the two problems are analogous). We also believe that the use of diagonal scaling may benefit other force-directed layout methods that are based on steepest descent.

References

1. Fisk, C.J., Isett, D.D.: ACCEL: automated circuit card etching layout. In: DAC 1965. Proceedings of the SHARE design automation project, pp. 9.1–9.31. ACM Press, New York (1965)
2. Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Information Processing Letters 31, 7–15 (1989)
3. Dwyer, T., Marriott, K., Wybrow, M.: Integrating edge routing into force-directed layout. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 8–19. Springer, Heidelberg (2007)
4. Dwyer, T., Koren, Y., Marriott, K.: IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. IEEE Transactions on Visualization and Computer Graphics 12, 821–828 (2006)
5. Borg, I., Groenen, P.J.: Modern Multidimensional Scaling: theory and applications, 2nd edn. Springer, Heidelberg (2005)
6. Gansner, E., Koren, Y., North, S.: Graph drawing by stress majorization. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 239–250. Springer, Heidelberg (2005)
7. Bertsekas, D.P.: Nonlinear Programming. Athena Scientific (1999)
8. Dwyer, T., Marriott, K.: Constrained stress majorization using diagonally scaled gradient projection. Technical Report 217, Clayton School of IT, Monash University (2007)
9. os, P.E., Rényi, A.: On random graphs. Publ. Math. Inst. Hungar. Acad. Sci. 5, 17–61 (1960)
10. Barabási, A.L., Reka, A.: Emergence of scaling in random networks. Science 286, 509–512 (1999)

Line Crossing Minimization on Metro Maps*

Michael A. Bekos[1], Michael Kaufmann[2], Katerina Potika[1], and Antonios Symvonis[1]

[1] National Technical University of Athens, School of Applied Mathematics & Physical Sciences, 15780 Zografou, Athens, Greece
mikebekos@math.ntua.gr, epotik@cs.ntua.gr, symvonis@math.ntua.gr
[2] University of Tübingen, Institute for Informatics, Sand 13, 72076 Tübingen, Germany
mk@informatik.uni-tuebingen.de

Abstract. We consider the problem of drawing a set of simple paths along the edges of an embedded underlying graph $G = (V, E)$, so that the total number of crossings among pairs of paths is minimized. This problem arises when drawing metro maps, where the embedding of G depicts the structure of the underlying network, the nodes of G correspond to train stations, an edge connecting two nodes implies that there exists a railway line which connects them, whereas the paths illustrate the lines connecting terminal stations. We call this the *metro-line crossing minimization problem (MLCM)*.

In contrast to the problem of drawing the underlying graph nicely, MLCM has received fewer attention. It was recently introduced by Benkert et. al in [4]. In this paper, as a first step towards solving MLCM in arbitrary graphs, we study path and tree networks. We examine several variations of the problem for which we develop algorithms for obtaining optimal solutions.

Keywords: Metro Maps, Crossing Minimization, Lines, Paths, Trees.

1 Motivation

We consider a relatively new problem that arises when drawing metro maps or public transportation networks in general. In such drawings, we are given an undirected embedded graph $G = (V, E)$, which depicts the structure of the underlying network. In the case of metro maps, the nodes of G correspond to the train stations whereas an edge connecting two nodes implies that there exists a railway line which connects them. The problem we consider is motivated by the fact that an edge within the underlying network may be used by several metro lines. Since crossings are often considered as the main source of confusion in a visualization, we want to draw the lines along the edges of G, so that they cross each other as few times as possible.

* This work has been funded by the project PENED-2003. PENED-2003 is co - funded by the European Social Fund (75%) and Greek National Resources (25%).

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 231–242, 2007.

In the graph drawing literature, the focus has been so far exclusively on drawing the underlying graph nicely and not on how to embed the bus or the metro lines along the underlying network. The latter problem was recently introduced by Benkert et. al in [4]. Following their approach, we assume that the underlying network has already received an embedding. The problem of determining a solution of the general metro-line routing problem, in which the graph drawing and line routing are solved simultaneously would be of particular interest as a second step in the process of automated metro map drawing.

2 Problem Definition

We are given an undirected embedded graph $G = (V, E)$. We will refer to G as the *underlying network*. We are also given a set $\mathcal{L} = \{l_1, l_2, \ldots, l_k\}$ of simple paths of G (in the following, referred to as *lines*). Each line l_i consists of a sequence of edges $e_1 = (v_0, v_1), \ldots, e_d = (v_{d-1}, v_d)$ of G. The nodes v_0 and v_d are referred to as the *terminals* of line l_i. We also denote by $|l_i|$ the length of line l_i. The main task is to draw the lines along the edges of G, so that the number of crossings among pairs of lines is minimized. We call this the *metro-line crossing minimization problem (MLCM)*. Formally, the MLCM problem is defined as a tuple $(G, \mathcal{L})$, where G is the underlying network and $\mathcal{L}$ is the set of lines.

One can define several variations of the MLCM problem based on the type of the underlying network, the location of the crossings and/or the location of the terminals. In general, the underlying network is an undirected graph. In this paper, as a first step towards solving MLCM problem in arbitrary graphs, we study path and tree networks.

For aesthetic reasons, we insist that the crossings between lines that traverse a node of the underlying network should not be hidden under the area occupied by that node. This implies that the relative order of the lines should not change within the nodes and therefore, all possible crossings have to take place along the edges of the underlying network.

In our approach, we assume that the nodes are drawn as rectangles, which is a quite usual convention in metro maps. Each line that traverses a node u has to touch two of the sides of u at some points (one when it "enters" u and one when it "leaves" u). These points are referred to as *tracks*. In general, we may permit tracks to all four side of the node, i.e. a line that traverses a node may use any side of it to either "enter" or "leave". This model is referred to as 4-*side* model (see Figure 1). A more restricted model referred to as 2-*side* model is the one, where all lines that traverse a node use only its left and right sides (see Figure 2). In the latter case, we only allow tracks at the left and right sides of the node. Note that a solution for the MLCM problem should first specify the number of tracks that enter each side of each station and, for each track, the line of $\mathcal{L}$ that uses it.

A further refinement of the MLCM problem concerns the location of the terminals at the nodes. A particularly interesting case - that arises under the 2-side model - is the one where the lines that terminate at a station occupy its

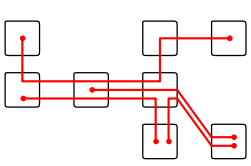


Fig. 1. 4-side model

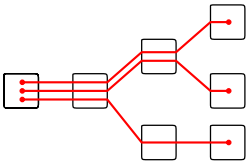


Fig. 2. 2-side model

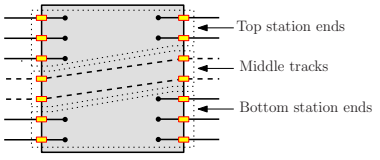


Fig. 3. Station ends, middle tracks

topmost and bottommost tracks, in the following referred to as *top* and *bottom station ends*, respectively. The remaining tracks on the left and right sides of the station are referred to as *middle tracks* and are occupied by the lines that traverse the station. Figure 3 illustrates the notions of station ends and middle tracks on the left and right sides of a station (solid lines correspond to lines that terminate, whereas the dashed lines correspond to lines that traverse the station). Based on these we introduce the following two variants of the MLCM problem:

(a) *The MLCM problem with terminals at station ends* (MLCM-SE), where we ask for a drawing of the lines along the edges of G so that (i) all lines terminate at station ends and (ii) the number of crossings among pairs of lines is minimized.
(b) *The MLCM problem with terminals at fixed station ends* (MLCM-FixedSE), where all lines terminate at station ends and the information whether a line terminates at a top or at a bottom station end in its terminal stations is specified as part of the input. We ask for a drawing of the lines along the edges of G so that the number of crossings among pairs of lines is minimized.

2.1 Related Literature

The problem of drawing a graph with a minimum number of crossings has been extensively studied in the graph drawing literature. For a quick survey refer to [2] and [6]. However, in the problems we study in this paper we assume that the underlying graph has already received an embedding and we seek to draw the lines along the graph's edges, so that the number of crossings among pairs of lines is minimized.

This problem was recently introduced by Benkert et. al in [4]. In their work, they proposed a dynamic-programming based algorithm that runs in $O(n^2)$ time for the *one-edge layout problem*, which is defined as follows: Given a graph $G = (V, E)$ and an edge $e = (u, v) \in E$, let L_e be the set of lines that traverse e. L_e is divided into three subsets L_u, L_v and L_{uv}. Set L_u (L_v) consists of the lines that traverse u (v) and terminate at v (u). Set L_{uv} consists of the lines that traverse both u and v and do not terminate either at u or at v. 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, 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 number of pairs of intersecting lines is

then determined by inserting the lines of L_u into the order S_v and by inserting the lines of L_v into the order S_u. The task is to determine appropriate insertion orders so that the number of pairs of intersecting lines is minimized. However, Benkert et. al [4] do not address the case of larger graphs and they leave as an open problem the case where the lines that terminate at a station occupy its station ends.

For the latter problem, Asquith et al. [1] proposed an integer linear program, which always determines an optimal solution regardless the type of the underlying network. They mention that their approach can be generalized to support the case where the set of the lines consists of subgraphs of the underlying network of maximum degree 3.

A closely related problem to the one we consider is the problem of drawing a metro map nicely, widely known as *metro map layout problem*. Hong et al. [5] implemented five methods for drawing metro maps using modifications of spring-based graph drawing algorithms. Stott and Rodgers [9] have approached the problem by using a hill climbing multi-criteria optimization technique. The quality of a layout is a weighted sum over five metrics that were defined for evaluating the niceness of the resulting drawing. Nöllenburg and Wolff [8] specified the niceness of a metro map by listing a number of hard and soft constraints and they proposed a mixed-integer program which always determines a drawing that fulfills all hard constraints (if such exists) and optimizes a weighted sum of costs corresponding to the soft constraints.

In Section 3, we consider the MLCM problem on a path. We show that the MLCM-SE problem is NP-Hard and we present a polynomial time algorithm for the MLCM-FixedSE problem. In Section 4, we consider the MLCM problem on a tree and we present polynomial time algorithms for two variations of it. We conclude in Section 5 with open problems and future work. Due to lack of space, Theorem proofs are either sketched or omitted. Detailed proofs can be found in [3].

3 The Metro-line Crossing Minimization Problem on a Path

We first consider the case where the underlying network G is a path and its nodes are restricted to lie on a horizontal line. We adopt the 2-side model where each line uses the left side of a node to *"enter"* it and the right one to *"leave"* it. Then, assuming that there exist no restrictions on the location of the line terminals at the nodes, it is easy to see that there exist solutions without any crossing among lines. So, we further assume that the lines that terminate at a station occupy its top and bottom station ends. In particular, we consider the MLCM-SE problem on a path. Since the order of the stations is fixed as part of the input of the problem, the only remaining choice is whether each line terminates at the top or at the bottom station end in its terminal stations. In the following, we show that under this assumption, the problem of determining a

solution so that the total number of crossings among pairs of lines is minimized is NP-Hard, by reducing to it the *fixed linear crossing number problem* [7].

Definition 1. *Given a simple graph $G = (V, E)$, a* linear embedding *of G is a special type of embedding in which the nodes of V are placed on a horizontal line L and the edges are drawn as semicircles either above or bellow L.*

Definition 2. *A* node ordering *(or a* node permutation*) of a graph G is a bijection $\delta : V \to \{1, 2, \ldots, n\}$, where $n = |V|$. For each pair of nodes u and v, with $\delta(u) < \delta(v)$ we shortly write $u < v$.*

Masuda et al. [7] proved that it is NP-Hard to determine a linear embedding of a given graph with minimum number of crossings, even if the ordering of the nodes on L is fixed. The latter problem is referred to as *fixed linear crossing number problem*.

Theorem 1. *The MLCM-SE problem on a path is NP-Hard.*

Proof. Let I be an instance of the fixed linear crossing number problem, consisting of a graph $G=(V, E)$ and a horizontal input line L, where $V=\{u_1, u_2, \ldots, u_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. Without loss of generality, we assume that $u_1 < u_2 < \ldots < u_n$. We construct an instance I' of the MLCM-SE problem on a path as follows: The underlying network $G' = (V', E')$ is a path consisting of $n+2$ nodes and $n+1$ edges, where $V' = V \cup \{u_0, u_{n+1}\}$ and $E' = \{(u_{i-1}, u_i);\ 1 \leq i \leq n+1\}$. The set of lines $\mathcal{L}$ is partitioned into two sets $\mathcal{L}^A$ and $\mathcal{L}^B$:

- $\mathcal{L}^A$ consists of a sufficiently large number of lines (e.g. $2nm^2$ lines) connecting u_0 with u_{n+1}.
- $\mathcal{L}^B$ contains m lines $l_1, l_2, \ldots, l_m$ one for each edge of G. Line l_i which corresponds to edge e_i of G, has terminals at the end points of e_i.

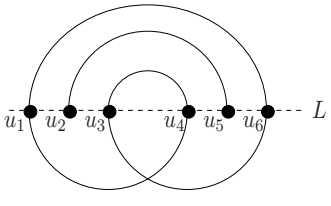


Fig. 4. A linear embedding

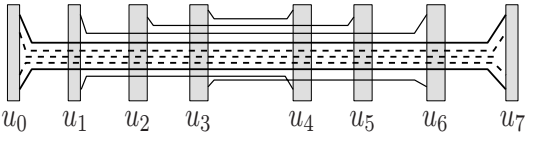


Fig. 5. An instance of MLCM-SE problem

Figures 4 and 5 illustrate the construction. First observe that all lines of $\mathcal{L}^A$ can be routed "in parallel" without any crossing among them (see Figure 5). Also observe that in an optimal solution none of the lines $l_1, l_2, \ldots, l_m$ crosses the lines of $\mathcal{L}^A$, since that would contribute a very large number of crossings. Thus, in an optimal solution each line of $\mathcal{L}^B$ has both of its terminals either at top or at bottom station ends. So, in a sense, we exclude the case where a line

$l_i \in \mathcal{L}^B$ has one of its terminals at a top station end, whereas the second one at a bottom station end. It is easy to see now that there exists an one-to-one correspondence between the crossings among the edges of I and the crossings among the lines in I', as desired. □

3.1 The Metro-line Crossing Minimization Problem with Fixed Positioned Terminals

Theorem 1 implies that, unless $P = NP$, we can not efficiently determine an optimal solution of MLCM-SE problem on a path. The main reason for this is that the information whether each line terminates at the top or at the bottom station end in its terminal stations is not known in advance. In the following, we assume that this information is part of the input, which is a reasonable assumption, since terminals may represent physical locations within a station. In particular, we show that the MLCM-FixedSE problem on a path can be solved in polynomial time.

To simplify the description of our algorithm, we assume that each node u_i of the path G is adjacent to two nodes u_i^t and u_i^b, each of which will be the terminal of the lines that terminated at the top and bottom terminal tracks of node u_i, respectively[1]. In the drawing of G, u_i^t is placed directly on top of u_i (*top leg* of u_i), whereas u_i^b directly bellow it (*bottom leg* of u_i), see Figure 6a. So, instead of restricting each line to terminate at a top or at a bottom station end in its terminal stations, we will equivalently consider that it terminates to two leg nodes. We refer to this special type of graph which is implied by the addition of the leg nodes as *caterpillar with at most two legs per node*.

A caterpillar with at most two legs per node consists of two sets of nodes. The first set, denoted by V_b, contains n nodes $u_1, u_2, \ldots, u_n$ (referred to as *backbone nodes*), which form a path. In the embedding of G, these nodes are collinear and more precisely they are located on a horizontal line so that $u_1 < u_2 < \ldots < u_n$. The second set of nodes, denoted by V_l, contains n' nodes $v_1, v_2, \ldots v_{n'}$ of degree 1 (referred to as *leg nodes* or simply as *legs*) each of which is connected to one backbone node. In the embedding of G, we assume that for each backbone u one of its legs is placed directly on top of it, whereas the second one directly bellow it. Since each backbone node is adjacent to at most two legs, $n' \leq 2n$.

If v is a leg node, we will refer to its neighbor backbone node as $bn(v)$. Edges that connect backbone nodes are called *backbone edges*. Edges that connect backbone nodes with legs are called *leg edges*.

Definition 3. *Let $l \in \mathcal{L}$ be a line that connects two terminals v and v'. If v is located to the left of v' in the embedding of the underlying network, i.e. $v < v'$, then we consider v to be the* origin *of line l, whereas v' to be its* destination. *We also denote by $\mathcal{L}_i^t$ ($\mathcal{L}_i^b$) the lines that have as origin the top (bottom) leg node adjacent to backbone node u_i.*

[1] In the degenerated case, where there exists no lines terminating either at the top or bottom terminal tracks of node u_i, we assume that either u_i^t or u_i^b does not exist, respectively.

Definition 4. *Let l and l' be a pair of lines that have the same origin w and destination nodes v and v', respectively. We say that l* precedes *l', if when we start moving from w along the external face of G in counterclockwise direction we meet v before v'. The notion of precedence defines an order $\preceq$ among the lines that have the same origin, namely $l \preceq l'$, if and only if l precedes l'.*

Lemma 1. *The number of tracks in the left and right side of each backbone node that are needed in order to route all lines in $\mathcal{L}$ can be computed in $O(n+\sum_{i=1}^{|\mathcal{L}|} |l_i|)$ time.*

Proof. The number of tracks in the right side of the leftmost backbone node u_1 is $|\mathcal{L}_1^t| + |\mathcal{L}_1^b|$. Due to the fact that no lines have as terminal a backbone node, the same number of tracks are needed in the left side of node u_2. We index the needed tracks from top to bottom (refer to Figure 6b). We compute the number of tracks in the left side of any backbone node u_i as the number of lines originating at nodes $< u_i$ and destined for nodes $\geq u_i$. Similarly, we compute the number of tracks in the right side of any backbone node u_i as the number of lines originating at nodes $\leq u_i$ and destined for nodes $> u_i$.

Assuming that $\mathcal{L}_{u_i}$ is the set of lines that traverse a backbone node u_i, then the tracks at the left and right side of backbone node u_i can be computed in $O(|\mathcal{L}_{u_i}|)$ time, yielding to a total $O(n + \sum_{i=1}^{|\mathcal{L}|} |l_i|)$ time. □

The lines of $\mathcal{L}$ are drawn incrementally by performing a left to right pass over the set of backbone nodes and by extending them from station to station with small horizontal or diagonal line segments. Therefore, each line $l \in \mathcal{L}$ is drawn as a polygonal line.

In each leg edge, that connects leg node v to $bn(v)$, we use $|\mathcal{L}_v|$ tracks indexed from right to left (refer to Figure 6b), where set $\mathcal{L}_v$ consists of the lines that either originate at or are destined for leg node v. These tracks will be used in order to route the lines that either originate at or are destined for leg node v.

In each backbone node u_i, we have to route the newly "introduced" lines, i.e. the ones that originate either at the top or at bottom leg of u_i. This procedure is illustrated in Figure 6b. We first consider the top leg node u_i^t of u_i. We sort the set $\mathcal{L}_i^t$ of the lines that originate at u_i^t in increasing order $\preceq$ of their destinations and store them in $Sort(\mathcal{L}_i^t)$. Based on this sorting we route the j-th line l in $Sort(\mathcal{L}_i^t)$ through the j-th rightmost track at the top of u_i. l is then routed to the j-th top track in the right side of u_i. We proceed by considering the bottom leg node u_i^b of u_i. Again, we sort the set $\mathcal{L}_i^b$ of the lines that originate at u_i^b in decreasing order $\preceq$ of their destinations and store them in $Sort(\mathcal{L}_i^b)$. Based on the sorting, we route the j-th line l in $Sort(\mathcal{L}_i^b)$ through the j-th rightmost track at the the bottom of u_i and then to the j-th bottom track in the right side of u_i. We then route the lines that go from the tracks of the left side to the tracks of the right side of u_i, by preserving their relative positions.

The next step is to route the lines from the right side of u_i to the left side of u_{i+1}. This is done by performing three passes over the set of tracks of the right side of u_i.

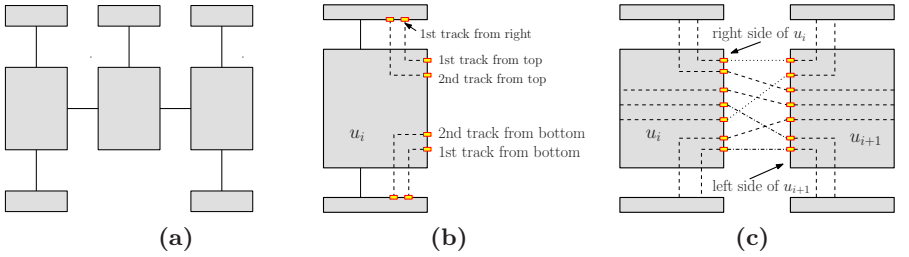


Fig. 6. (a) A caterpillar with at most 2 legs per node, (b) Introducing new lines to a station, (c) Routing lines along a backbone edge

In the first pass, we consider the tracks of the right side of u_i from top to bottom and we check whether the line l that occupies the j-th track is destined for the leg node u^t_{i+1}. In this case, we route l to the topmost available track of the right side of u_{i+1} and then to the leftmost available track in the leg edge which connects u_{i+1} with u^t_{i+1} (see the dotted lines of Figure 6c). In the second pass, we consider the remaining tracks of the right side of u_i from bottom to top and we check whether the line l that occupies the j-th track is destined for the leg node u^b_{i+1}. In this case, we route l to the bottommost available track of the right side of u_{i+1} and then to the leftmost available track in the leg edge which connects u_{i+1} with u^b_{i+1} (see the dash dotted lines of Figure 6c).

The remaining tracks of the right side of u_i are obviously occupied by the lines that are not destined for either u^t_{i+1} or for u^b_{i+1}. We consider these tracks from top to bottom and we route the line l that occupies the j-th track to the topmost available track of the right side of u_{i+1} (see the dashed lines of Figure 6c). The construction of our algorithm guarantees the following two properties:

> **Property of common destinations:** Lines that are destined for the same top (bottom) leg node u^t_i (u^b_i) do not cross each other along the backbone edge which connects u_{i-1} with u_i.
> **Property of parallel routing:** Two lines that both traverse a backbone node u_i (i.e. none of them are destined either for u^t_i or for u^b_i) do not cross each other along the backbone edge which connects u_{i-1} with u_i.

By combining the property of *common destinations* and the property of *parallel routing*, we easily obtain the following lemma.

Lemma 2. *In a solution produced by our algorithm the followings hold:*

(i) Two lines l and l' cross each other at most once.
(ii) Two lines l and l' with the same origin do not cross each other.
(iii) Two lines l and l' with the same destination do not cross each other.
(iv) Let l and l' be two lines that cross each other and let l (l') be destined for leg node v (v'), where v is to the left of v' in the embedding of G. Then, l and l' will cross along the backbone edge which connects u_{k-1} and u_k, where $u_k = bn(v)$.

By using Lemma 2, we can show that our algorithm produces an optimal solution, in terms of line crossings. Theorem 2 summarizes our result.

Theorem 2. *An instance $(G, \mathcal{L})$ of the MLCM-FixedSE problem on an n-node path P can be solved in $O(n + \sum_{i=1}^{|\mathcal{L}|} |l_i|)$ time.*

4 The Metro-line Crossing Minimization Problem on a Tree

In this Section, we consider the MLCM problem on a tree $T = (V, E)$, where $V = \{u_1, \ldots, u_n\}$ and $E = \{e_1, \ldots, e_{n-1}\}$. In the embedding of T, we assume that the neighbors of each node u of T are located either to the left or to the right of u. In particular, we consider a "*left-to-right tree structured network*" to represent the underlying network. In such a network, we do not allow lines which make "right-to-right" or "left-to-left" turns, which implies that all lines should be x-monotone. This assumption is motivated by the fact that a train can not make an 180° turn within a station. We seek to route all lines along the edges of T, so that the total number of crossings along the lines is minimum.

We adopt the 2-side model, where each line uses the left side of a node to *"enter"* it and the right one to *"leave"* it. We refer to the edges that are adjacent to the left (right) side of node u in the embedding of T as *incoming (outgoing) edges* of u. Since we assume that the lines are x-monotone, the notions of the origin and the destination of a line, as defined in Section 3.1, also apply in the case of line crossing minimization on "left-to-right tree structured network".

We consider the case where all terminals are located only at nodes of degree 1 and the lines can terminate at any track of their terminal stations[2].

Assuming that the edges of T are directed from left to right in the embedding of T, we first perform a topological sorting over the nodes of T. We will use this sorting later on when we route all lines along the edges of T. We proceed by numbering all nodes of T with outdegree zero[3] according to the order of appearance when moving clockwise along the external face of T starting from the first node obtained from the topological sort. Note that such a numbering is unique and we refer to it as the *Euler tour numbering* of the destination nodes.

Since the number of lines that "enter" an internal node is equal to the number of lines that "leave" it, we simply have to specify either the order of the lines that enter the node or the corresponding order when they leave it. Recall that we do not permit crossings inside the nodes. As in the preceding section, we route the lines along the edges of T incrementally. We consider the nodes of T in their topological order. This ensures that whenever we consider the next node u all of its incoming lines have already been routed up to its left neighbor nodes. We distinguish the following cases:

[2] Recall that, in the case of a path network, this problem was quite easy due to the structure of the path.

[3] Such nodes are possible line destinations.

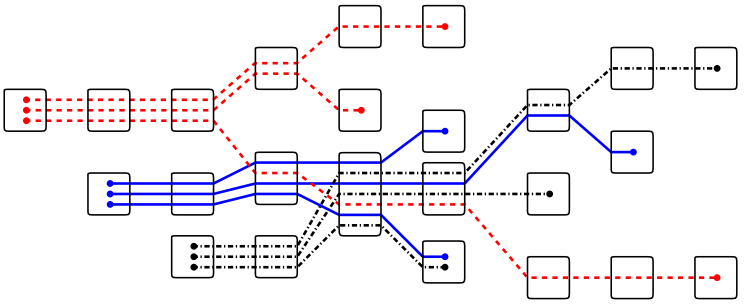


Fig. 7. A sample routing obtained from our algorithm

Case 1: $indegree(u) = 0$
If node u is of indegree zero (i.e. u is a leaf containing the origins of some lines), we simply sort the lines that originate from u based on the Euler tour numbering of their destinations in ascending order.

Case 2: $indegree(u) = 1$
We simply pass the lines from the left neighbor node of u to u without introducing any crossing (i.e. by keeping the order of the lines unchanged).

Case 3: $indegree(u) > 1$
In the case where node u is of indegree greater than one, we have to "merge" its incoming lines and thus, we may introduce crossings. We "stably merge" the incoming lines based on the Euler tour numbering of their destinations so that:

- Lines coming along the same edge do not change order.
- If two lines with the same destination come along different edges, the one coming from the topmost edge is considered to be smaller.

Figure 7 illustrates a sample routing produced by our algorithm. We use different types of lines to denote lines that originate at a common leaf node. The construction of our algorithm supports the following Lemma:

Lemma 3. *In a solution produced by our algorithm the following hold:*

(i) Two lines l and l' cross each other at most once.
(ii) Two lines l and l' with the same origin do not cross each other.
(iii) Two lines l and l' with the same destination do not cross each other.
(iv) Let l and l' be two lines that cross each other. Then, l and l' will cross along their leftmost common edge.
(v) Let l and l' be two lines that cross each other. Then, l and l' will cross just before entering their leftmost common node.

By using Lemma 3, we can show that our algorithm produces an optimal solution, in terms of line crossings. Theorem 3 summarizes our result.

Theorem 3. *Assuming that each line terminates at leaf nodes, an instance* $(T, \mathcal{L})$ *of the MLCM problem on a "left-to-right"* n*-node tree* T *can be solved in* $O(n + \sum_{i=1}^{|\mathcal{L}|} |l_i|)$ *time.*

4.1 The MLCM-SE and MLCM-FixedSE Problems on a Tree

Since a path can be viewed as a degenerated case of a tree, Theorem 1 implies that MLCM-SE problem on a tree is NP-Hard . However, for the MLCM-FixedSE problem we can obtain a polynomial time algorithm adopting a similar approach as the one of Section 3.1. For each node u of T we introduce at most four new nodes u_L^t, u_L^b, u_R^t and u_R^b adjacent to u. Node u_L^t (u_L^b) is placed on top (bellow) and to the left of u in the embedding of T and contains all lines that originate at u's top (bottom) station end. Similarly, node u_R^t (u_R^b) is placed on top (bellow) and to the right of u in the embedding of T and contains all lines that are destined for u's top (bottom) station end. In the case where any of the u_L^t, u_L^b, u_R^t or u_R^b does not contain any lines we ignore its existence. So, instead of restricting each line to terminate at a top or at a bottom station end in its terminal stations, we equivalently consider that it terminates to some of the newly introduced nodes. Note that the underlying network remains a tree after the introduction of the new nodes, so our algorithm can be applied in this case, too. The following Theorem summarizes our result.

Theorem 4. *An instance* $(T, \mathcal{L})$ *of the MLCM-FixedSE problem on a "left-to-right"* n*-node tree* T *can be solved in* $O(n + \sum_{i=1}^{|\mathcal{L}|} |l_i|)$ *time.*

5 Conclusions

Clearly, our work is a first step towards solving the MLCM problem and its variants in arbitrary graphs. Extending the work of Benkert et al. [4] we studied path and tree networks. However, we did not consider the case where the underlying network is an arbitrary graph. Additionally, for the case where the underlying network is a tree we only considered the case, where the terminals are located at nodes of degree 1. No results are known regarding the case where we permit terminals at internal nodes of the tree. Another line of research would be to develop approximation algorithms for the MLCM-SE problem on paths and trees. The problem of determining a solution of the general metro-line routing problem, in which the graph drawing and line routing are solved simultaneously is also of particular interest as a second step in the process of automated metro map drawing.

References

1. Asquith, M., Gudmundsson, J., Merrick, D.: An ILP for the line ordering problem. Technical Report PA006288, National ICT Australia (2007)
2. Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall, Englewood Cliffs (1999)

3. Bekos, M.A., Kaufmann, M., Potika, K., Symvonis, A.: Line crossing minimization on metro maps. Technical Report WSI-2007-03, University of Tübingen (2007)
4. Benkert, M., Nöllenburg, M., Uno, T., Wolff, A.: Minimizing intra-edge crossings in wiring diagrams and public transport maps. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 270–281. Springer, Heidelberg (2007)
5. Hong, S.-H., Merrick, D., Nascimento, H.A.D.d.: The metro map layout problem. In: Churcher, N., Churcher, C. (eds.) invis.au 2004. Australasian Symposium on Information Visualisation, CRPIT, ACS, vol. 35, pp. 91–100 (2004)
6. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
7. Masuda, S., Nakajima, K., Kashiwabara, T., Fujisawa, T.: Crossing minimization in linear embeddings of graphs. IEEE Trans. Comput. 39(1), 124–127 (1990)
8. Nöllenburg, M., Wolff, A.: A mixed-integer program for drawing high-quality metro maps. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 321–333. Springer, Heidelberg (2006)
9. Stott, J.M., Rodgers, P.: Metro map layout using multicriteria optimization. In: Proc. 8th International Conference on Information Visualisation, pp. 355–362. IEEE Computer Society, Los Alamitos (2004)

Algorithms for Multi-criteria One-Sided Boundary Labeling*

Marc Benkert[1,**], Herman Haverkort[2], Moritz Kroll[1], and Martin Nöllenburg[1,**]

[1] Faculty of Informatics, Karlsruhe University, P.O. Box 6980, 76128 Karlsruhe, Germany
http://i11www.iti.uka.de/algo/group
[2] Department of Computing Science, TU Eindhoven, Postbus 513, 5600 MB Eindhoven, Netherlands

Abstract. We present new algorithms for labeling a set P of n points in the plane with labels that are aligned to the left of the bounding box of P. The points are connected to their labels by curves (leaders) that consist of two segments: a horizontal segment, and a second segment at a fixed angle with the first. Our algorithm finds a collection of non-intersecting leaders that minimizes the total number of bends, the total length, or any other 'badness' function of the leaders. An experimental evaluation of the performance is included.

1 Introduction

Presentations of visual information often make use of textual labels for features of interest within the visualizations. Examples are found in diverse areas such as cartography, anatomy, engineering, sociology etc. Graphics in these areas may have very dense regions in which objects need textual labels to be fully understood. A lot of research on automatic label placement has concentrated on placing labels inside the graphic itself, see the bibliography on map labeling by Wolff and Strijk [6]. However, this is not always possible: sometimes the labels are too large, the labeled features lie to close to each other, or the underlying graphic should remain fully visible. In such cases it is often necessary to place the labels next to the actual illustration and connect each label to its object by a curve—see Figure 1. This is also denoted as a *call-out*, and the curves are called *leaders*. Geographic maps that depict metropolitan areas and medical atlases are examples where call-outs are used.

To produce a call-out, we have to decide where exactly to place each object's label and how to draw the curves such that the connections between objects and labels are clear and the leaders do not clutter the figure. Clearly, leaders should

* This work was started on the 9th "Korean" Workshop on Computational Geometry and Geometric Networks at Schloss Dagstuhl, Germany, 2006.

** Supported by grant WO 758/4-2 of the German Science Foundation (DFG) and partially by EU under grant DELIS (contract no. 001907).

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 243–254, 2007.

Fig. 1. Examples of call-outs with bends of 90° (*po*-leaders) or 120° (*do*-leaders), respectively. The leaders for Haverdorf are *direct* leaders.

not intersect each other to avoid confusion, and several authors have designed algorithms to produce non-intersecting leaders in several settings. Fekete and Plaisant [5] label point objects with polygonal leaders with up to two bends in an interactive setting, Ali et al. [1] describe heuristics to label points with straight-line and rectilinear leaders. Bekos et al. use rectilinear leaders with up to two bends. They study settings with labels arranged on one, two, or four sides of the bounding box of the illustration [4], in multiple stacks to the left [2], or where the objects to be labeled are polygons rather than points [3]. Maybe surprisingly, relying exclusively on straight-line leaders is not always the best choice. The reason is that the variety of different slopes among the leaders may clutter the figure, especially if the number of labels is large. Leaders tend to look less disturbing if their shape is more uniform and a small number of slopes is used, like with rectilinear leaders. On the other hand, leaders appear easier to follow if their bends are smooth, so 90° angles may rather be avoided.

In this work we study how to label points with labels on one side of the illustration and leaders with at most one bend. Bekos et al. [4] only studied how to minimize the total leader length with rectilinear leaders in this setting; their algorithm runs in $O(n^2)$ time. In this paper we consider other optimization criteria, we consider leaders with smoother bends (using obtuse angles), and for the case of rectilinear leaders with minimum total length, we improve the running time to $O(n \log n)$. We will now state our problem more precisely.

Problem statement. We are given a set P of n points and n disjoint rectangles, possibly of different sizes, called *labels*. The right edges of the labels all lie on a common vertical line, which lies to the left of all points in P. No two labels touch each other.

Labels can be connected to points by *leaders* that consist of two line segments: a horizontal segment, called the *arm*, that is attached to the right edge of the label and extends to the right, and a second segment, called the *hand*, that connects the arm to the point. In all leaders the angle between the arm and the hand must be some constant α. If $\alpha = 90°$ the leaders are called *po-leaders*; if $\alpha > 90°$, we call them *do-leaders*[1]. Both leader types are illustrated in Figure 1. If the arm connects the label directly to the point, omitting a hand, the leader is a *direct leader*. When α is fixed, a leader l is fully specified by its point $p(l)$ and the height (y-coordinate) of its arm. We assume that the 'badness' of a leader l is given by a function $bad(l)$. Natural choices for $bad(l)$ would be, for example, the

[1] Following the naming scheme of Bekos et al. [4].

length of l or the number of bends (0 or 1), or functions taking the interference of leaders with the underlying map into account. A *labeling* L is a set of n leaders that connects all points to a unique label and all labels to a unique point. If no two leaders in L intersect each other, we say that L is *crossing-free*.

The problem we want to solve is the following: for a given set of points, a given set of labels, a given angle α, and a given badness function $bad()$, find a crossing-free labeling L such that $\sum_{l \in L} bad(l)$ is minimized.

Our results. In Section 2 we present algorithms for *po*-leaders ($\alpha = 90°$): an $O(n^3)$-time algorithm that works with arbitrary badness functions, and an $O(n \log n)$-time algorithm for labelings with minimum total leader length (thus improving the $O(n^2)$-bound of Bekos et al. [4]).

In Section 3 we present algorithms for *do*-leaders ($\alpha > 90°$): again first a general algorithm, which runs in $O(n^5)$ time, and then a faster algorithm for minimum total leader length, which takes $O(n^2)$ time. In Section 4 we present the results of some preliminary experiments with our algorithms, and in Section 5 we briefly discuss possible extensions.

2 One-Sided Boundary Labeling Using *po*-leaders

In this section we study how to compute an optimal crossing-free labeling with leaders that have 90° bends. In Section 2.1 we describe a general solution that works for any badness function $bad()$. In Section 2.2 we will give a faster solution for the case where $bad(l)$ is simply the length of l.

For simplicity we assume that no two points lie on a horizontal or a vertical line and no point lies on a horizontal line with an edge of a label (otherwise care should be taken to break ties in a consistent manner).

2.1 A Dynamic Program for General Badness Functions

We present a dynamic programming solution based on the following idea. Let r be the rightmost point to be labeled. Consider any optimal crossing-free labeling L; let ℓ be the label associated with r in L. Then L consists of an optimal leader l connecting ℓ to r, an optimal crossing-free labeling for the remaining labels and points below the arm of l, and an optimal crossing-free labeling for the remaining labels and points above the arm of l—see Figure 2.

Consider the subdivision of the plane into $O(n)$ strips, induced by the horizontal lines through the points and the horizontal edges of the labels. Note that the bottommost strip is unbounded in downward direction, and the topmost strip is unbounded in upward direction. To decide which labels and points lie below the leader l to r, we only need to know in which strip the arm of l lies; we do not need to know where exactly it is in the strip. When an arm lies on a strip boundary, we can consider it to lie in the strip above the boundary or in the strip below; the choice determines whether a point on the strip boundary is considered to lie above or below the leader.

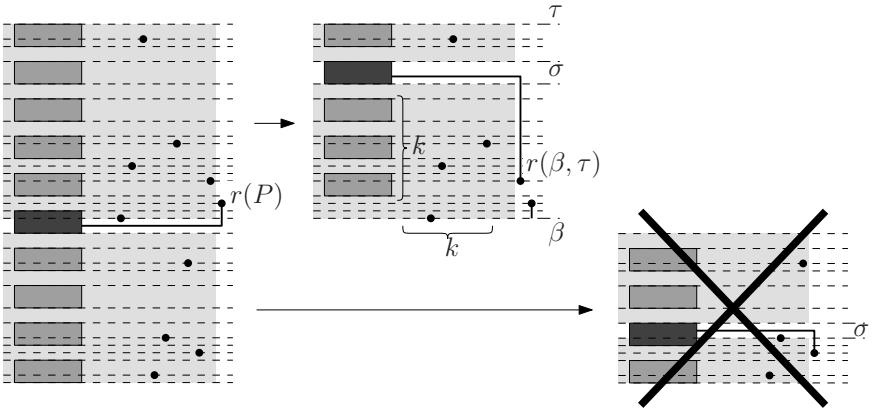


Fig. 2. The recursive structure of an optimal solution. By the choice for the strip that contains the arm of the leader to the rightmost point, the problem is separated into two subproblems. As illustrated by strip σ in the lower subproblem, not all choices for the separating strip σ yield feasible subproblems: in this case there are two points and only one label below σ.

Hence an optimal crossing-free labeling can be found by trying all possible choices of the strip σ in which to place the arm of the leader to r, and for each choice, compute the optimal leader to r that has its arm in σ, and compute the optimal crossing-free labelings below and above the arm recursively. Note that we only need to consider *feasible choices* of σ, that is, choices of σ such that the number of labels and the number of points below σ and to the left of r are the same (for other choices of σ no labeling would be possible). In this case, as can be seen in Figure 2, the points to be matched below σ are simply the leftmost k points in the region defined by the strips below σ, where k is the number of labels below σ; analogously, the points to be labeled above σ are the leftmost points in the region defined by the strips above σ.

Let us denote by $S(\beta, \tau)$ the set of strips between strip β (bottom) and τ (top), excluding β and τ. Let $r(\beta, \tau)$ be the k-th leftmost point in $S(\beta, \tau)$, where k is the number of labels $k(\beta, \tau)$ that lie completely inside $S(\beta, \tau)$. Our recursive approach thus solves subproblems of the following form: for the set of strips $S(\beta, \tau)$, compute the optimal matching between the labels that lie completely inside $S(\beta, \tau)$ and the matching number of leftmost input points inside (and on the boundary of) $S(\beta, \tau)$. The minimum total badness $BAD[\beta, \tau]$ of the optimal crossing-free labeling for $S(\beta, \tau)$ is zero if $k(\beta, \tau) = 0$, and otherwise it can be expressed as:

$$\min_{\text{feasible } \sigma \in S(\beta,\tau)} bad(l^*(r(\beta, \tau), \sigma)) + BAD[\beta, \sigma] + BAD[\sigma, \tau]$$

where $l^*(r(\beta, \tau), \sigma)$ is the optimal leader to $r(\beta, \tau)$ with its arm in strip σ.

Theorem 1. *Assume we are given a set of points P, a set of labels as described in Section 1, and a badness function $bad()$ such that we can determine, in $O(n)$*

time, the badness and the location of an optimal po-leader to a given point with its arm in a given height interval (independent of the location of other leaders). We can compute a crossing-free labeling with po-leaders for P with minimum total badness in $O(n^3)$ time and $O(n^2)$ space.

Proof. We first sort all labels and points by y-coordinate, and all points by x-coordinate, which requires $O(n \log n)$ time. We also compute and store $l^*(p, \sigma)$ and $bad(l^*(p, \sigma))$ for every point p and every strip σ, in $O(n^3)$ time and $O(n^2)$ space. Then we compute the optimal crossing-free labeling by dynamic programming with memoization. Apart from the recursive calls, solving a subproblem requires deciding for which choices of σ the number of labels below σ matches the number of points below σ, and looking up $l^*(r(\beta, \tau), \sigma)$ and $bad(l^*(r(\beta, \tau), \sigma))$ for those strips. Given the list of all points sorted by x-coordinate and the list of labels and points by y-coordinate, we can construct a list of all labels and points in the given subproblem sorted by y-coordinate in $O(n)$ time. By scanning this list, we can determine in $O(n)$ time which choices of σ yield feasible subproblems. The number of different subproblems that need to be solved is quadratic in the number of strips, so we need to solve $O(n^2)$ subproblems which are solved in $O(n)$ time each, taking $O(n^3)$ time in total. □

2.2 A Sweep-Line Algorithm for Minimizing the Total Leader Length

For the special case of minimizing the total leader length one can do better than in $O(n^3)$ time. We will give an algorithm that runs in $O(n \log n)$ time and show that this bound is tight in the worst case. However, before giving our algorithm, we first prove the following Lemma, which we need for the proof of correctness of our fast algorithms in this section and in Section 3.2.

Lemma 1. *For any labeling L^* with po- or do-leaders that may contain crossings and has minimum total leader length, there is a crossing-free labeling L whose total leader length does not exceed the total leader length of L^*. This labeling L can be constructed from L^* in $O(n^2)$ time.*

The idea for proving this lemma is to show that we can eliminate all crossings in L^* by iteratively swapping the labels of two points whose leaders intersect. Any of these swaps does not increase the total leader length; the complete proof can be found in a full version of this paper.

We now describe our $O(n \log n)$-time algorithm to compute a crossing-free labeling with *po*-leaders of minimum total length. The algorithm first scans the input to divide it into parts that can be handled independently; then it uses a sweep line algorithm for each of these parts.

The initial scan works as follows. Consider the horizontal strips defined in the previous subsection. We traverse these strips in order from bottom to top, counting for each strip σ:

- pa_σ: number of points above σ (incl. any point on the top edge of σ);
- ℓa_σ: number of labels above σ (incl. any label intersecting σ);

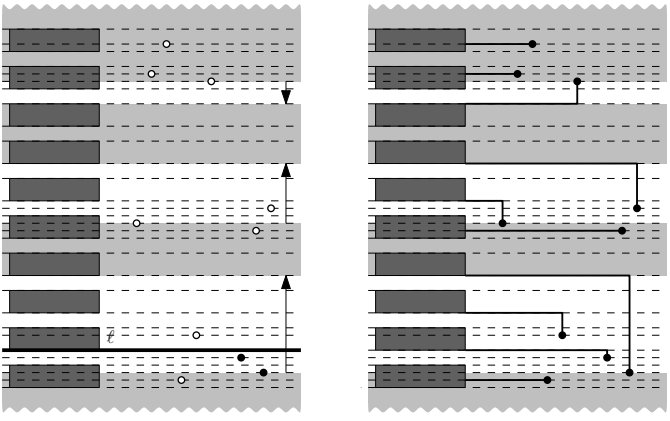


Fig. 3. Left: Classification of strips in the plane sweep algorithm: neutral strips are shaded, downward and upward strips are marked by arrows. When the sweep line reaches the label ℓ, the two black points are in W. Right: The completed minimum-length labeling.

- pb_σ: number of points below σ (incl. any point on the bottom edge of σ);
- ℓb_σ: number of labels below σ (incl. any label intersecting σ).

Note that for every strip, $pa_\sigma + pb_\sigma = n$, and $\ell a_\sigma + \ell b_\sigma$ is either n or $n+1$. We classify the strips in three categories and then divide the input into maximal sets of consecutive strips of the same category (see Figure 3):

- *downward:* strips s such that $pa_\sigma > \ell a_\sigma$ (and therefore $pb_\sigma < \ell b_\sigma$);
- *upward:* strips s such that $pb_\sigma > \ell b_\sigma$ (and therefore $pa_\sigma < \ell a_\sigma$);
- *neutral:* the remaining strips; these have $pa_\sigma = \ell a_\sigma$ and/or $pb_\sigma = \ell b_\sigma$.

Neutral sets are handled as follows: any point p that lies in the interior of a neutral set is labeled with a direct leader.

Points in an upward set S (including any points on its boundary) are labeled as follows. We use a plane sweep algorithm, maintaining a waiting list W of points to be labeled, sorted by increasing x-coordinate. Initially W is empty. We sweep S with a horizontal line from bottom to top. During the sweep two types of events are encountered: *point events* (the line hits a point p) and *label events* (the line hits the bottom edge of a label ℓ). When a point event happens, we insert the point in W. When a label event happens, we remove the leftmost point from W and connect it to ℓ with the shortest possible leader. Using the leftmost point for labeling ℓ prevents producing crossings in the further run of our algorithm.

Points in downward sets are labeled by a symmetric plane sweep algorithm, going from top to bottom.

Theorem 2. *Given a set of points P and a set of labels as described in Section 1, computing a crossing-free labeling with po-leaders of minimum total length for P takes $\Theta(n \log n)$ time and $O(n)$ space in the worst case.*

The proof of Theorem 2 will be available in a full version of the paper and shows that the algorithm sketched above produces a crossing free labeling of minimum length.

3 One-Sided Boundary Labeling Using *do*-leaders

In this section we study how to compute an optimal labeling with leaders that have bends with a fixed angle $\alpha > 90°$. In section 3.1 we describe a general solution that works for any badness function $bad()$. In section 3.2 we will give a faster solution for the case where $bad(l)$ is simply the length of l. For simplicity we assume that no two points lie on a line that makes an angle of 0°, 90°, or α with the x-axis, and no point lies on a horizontal line with an edge of a label (otherwise care should be taken to break ties in a consistent manner).

3.1 A Dynamic Program for General Badness Functions

We use the same approach as for *po*-leaders, solving subproblems of the form: for a given region R, label the k points with the k labels in that region, where R is bounded from above and below by two leaders, and R is bounded on the right by the vertical line through the rightmost point connected to those leaders. In fact a subproblem was fully defined by specifying the strips β and τ that contain the arms of the leaders: this determined which labels lie inside R, and consequently which point defines the vertical boundary line on the right.

In addition to specify β and τ we now also have to specify the points b and t to which the leaders that bound a subproblem are connected. This is illustrated by Figures 4a and 4b: the subproblem defined by β, τ, b and t contains the point r while the subproblem defined by β, τ, b' and t contains the point r' instead. The total number of different subproblems may thus increase to $O(n^4)$.

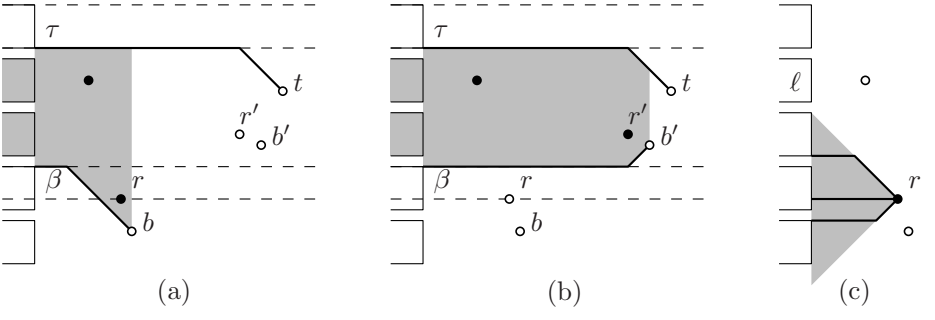


Fig. 4. (a) The subproblem defined by β, τ, b and t. (b) The subproblem defined by β, τ, b' and t. (c) Because leaders have limited slope, no leader from r can reach ℓ.

An additional complication is that as a result of the limited slope of leaders, not every subproblem with the right number of labels and points can be solved—see Figure 4c. The details are easily filled in and we get:

Theorem 3. *Assume we are given a set of points P, a set of labels as described in Section 1, a bend angle α, and a badness function bad() such that we can determine, in $O(n)$ time, the badness and the location of an optimal do-leader to a given point with its arm in a given height interval (independent of the location of other leaders). We can now compute a crossing-free labeling with do-leaders with bend angle α and minimum total badness for P, if such a labeling exists, in $O(n^5)$ time and $O(n^4)$ space.*

3.2 Minimizing the Total Leader Length

Like with *po*-leaders, we can use a plane sweep algorithm instead of dynamic programming to improve the running time for the special case of minimizing the total leader length. In the description of our algorithm we distinguish *downward diagonals* (lines of negative slope that make an angle of α with the x-axis) and *upward diagonals* (lines of positive slope that make an angle of α with the x-axis). For each label ℓ we can define three regions in the plane:

- $A(\ell)$ is the relatively open half plane *above* the *upward* diagonal through the *upper* right corner of ℓ;
- $B(\ell)$ is the relatively open half plane *below* the *downward* diagonal through the *lower* right corner of ℓ;
- $R(\ell)$ is the complement of $A(\ell) \cup B(\ell)$.

Note that a *do*-leader from a point p to ℓ is possible if and only if $p \in R(\ell)$.

The core of our approach is a recursive sweep-and-divide algorithm that takes as input a list of labels $\mathcal{L}$ and points P sorted in the order in which they would be (first) hit by a downward diagonal sweep line that sweeps the plane bottom-up and from left to right. For any line d, let $\mathcal{L}(d)$ be the set of labels whose lower right corners lie below or on d, and let $P(d)$ be the set of points that lie below or on d. The algorithm sweeps the plane with a downward diagonal d up to the first point where we have $|P(d)| = |\mathcal{L}(d)|$. Observe that we will have to find a one-to-one matching between $P(d)$ and $\mathcal{L}(d)$, since no leaders are possible between points below d and labels above d. We find such a matching as follows.

If $P(d) \neq P$, we make a recursive call on $P(d)$ and $\mathcal{L}(d)$, and a recursive call on the remaining input ($P \setminus P(d)$ and $\mathcal{L} \setminus \mathcal{L}(d)$), see Figure 5a.

If $P(d) = P$, we find the lowest label $\ell \in \mathcal{L}$. If no point of P lies in $R(\ell)$, we report that no labeling can be found and terminate the algorithm. Otherwise we make a leader from ℓ to the lowest point p in $P \cap R(\ell)$ (see Figure 5b and 5c); then, if $P \setminus \{p\}$ is not empty, we make a recursive call on $P \setminus \{p\}$ and $\mathcal{L} \setminus \{\ell\}$.

The full algorithm is now as follows. We first sort $\mathcal{L}$ and P into the order as described above. We then run the recursive sweep-and-divide algorithm described above. If the algorithm does not fail, the computed set of leaders has minimum total length (as we will prove below), but it may contain crossings. We eliminate these intersections with the algorithm described in the proof of Lemma 1.

Theorem 4. *Assume we are given a set of points P, a set of labels as described in Section 1, and a bend angle α. If there is a labeling for P with do-leaders with*

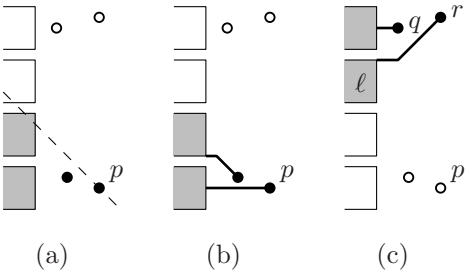


Fig. 5. Illustration of the length-minimization algorithm for *do*-leaders. (a) When the sweep line hits p, we make recursive calls on the input under the sweep line and the input above the sweep line. (b) The result of the recursive call under the sweep line. (c) The result of the recursive call above the sweep line. Although q is the lowest point, ℓ is attached to r, since q cannot reach ℓ.

bend angle α, we can compute a crossing-free labeling of minimum total leader length in $O(n^2)$ time and $O(n)$ space in the worst case. If such a labeling does not exist, we can report infeasibility within the same time and space bounds.

The proof of the correctness of our algorithm is based on the idea to show that any (not necessarily crossing-free) labeling can be transformed into the labeling constructed by our recursive algorithm without increasing the total leader length. Then Lemma 1 can be applied to eliminate the crossings of our solution. The proof will be available in a full version of the paper.

4 Experimental Evaluation

We implemented three variants of our algorithms: length minimization, bend minimization and a hybrid method combining both objectives. The corresponding badness functions bad_{len}, bad_{bend}, and bad_{hyb} are defined as follows.

$$bad_{\text{len}}(l) = |l|, \tag{1}$$

$$bad_{\text{bend}}(l) = \begin{cases} 0 & \text{if } l \text{ is direct} \\ 1 & \text{otherwise} \end{cases}, \tag{2}$$

$$bad_{\text{hyb}}(l) = \frac{|\text{hand}(l)|}{|\text{arm}(l)|} + \lambda_{\text{bend}}\, bad_{\text{bend}}(l), \tag{3}$$

where $|\cdot|$ denotes the Euclidean length. Note that in bad_{hyb} we do not simply reuse bad_{len} but rather include the length ratio of the hand and the arm of a leader which is motivated by the observation that a long hand on a short arm looks worse than on a long arm. The parameter λ_{bend} is used to adjust the weight of bad_{bend}.

Furthermore, we implemented another badness term bad_{cls} that measures how close points in P lie to a leader l within a neighborhood strip $N_\gamma(l)$ of width

γ around l. This term can be added to the previous badness functions to avoid that leaders pass by points with too little clearance. It is defined as

$$bad_{\text{cls}}(l) = \lambda_{\text{cls}} \sum_{p \in N_\gamma(l)} \left(1 - \frac{d(p,l)}{\gamma}\right)^2, \tag{4}$$

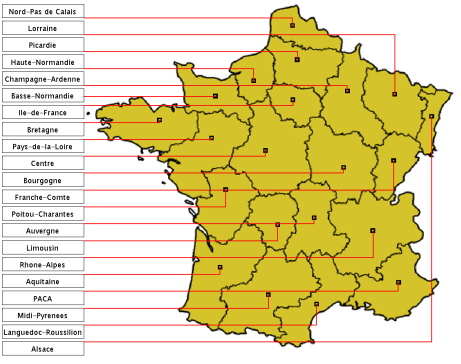
where λ_{cls} is a weight parameter and $d(p,l)$ is the distance between p and l. Adding bad_{cls} helps to reduce confusion when understanding the assignment of points and labels, see Figure 6a generated using bad_{len} and Figure 6b generated using $bad_{\text{len}} + bad_{\text{cls}}$.

We implemented our algorithms as a Java applet[2] and tested them on a map showing the 21 mainland regions of France, see Figure 6. The labelings were computed on an AMD Sempron 2200+ with 1GB main memory, which took between 1 and 5 ms for the *po*-leaders and 12 ms for the *do*-leaders with bend angle $\alpha = 135°$. Running the dynamic programs in a top-down fashion, for *po*-leaders 39% of $O(n^2)$ table entries were computed, while for the *do*-leaders only 0.21% of $O(n^4)$ entries were computed. We also ran the algorithms on artificially generated instances of 100 points uniformly distributed in a unit square. Here the computation of the *po*-leaders took 234 ms averaged over 30 instances and on average 22% of the table entries were computed. The average running time for the *do*-leaders on the same instances was 3328 ms and on average 0.01% of the table entries were computed.

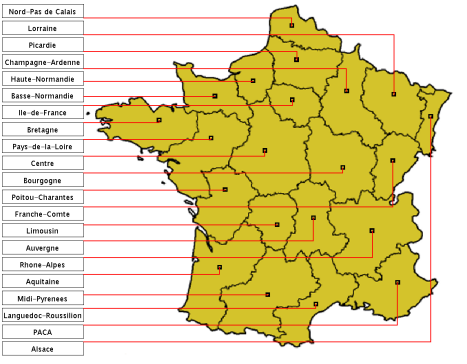
po-leaders vs. do-leaders. Both *po*-leaders and *do*-leaders in Figure 6 have advantages and disadvantages. Obviously, it is not possible to judge whether *po*-leaders or *do*-leaders are generally superior based on our single example map. The answer depends both on the labeled image and on personal taste. Still, an advantage of the *do*-leaders is that due to the smoother angle their shape is easier to follow visually, which simplifies finding the correct label for a point and vice versa.

Optimizing for length vs. bends. Minimizing the total leader length seems to give more comprehensible and visually more pleasing results than minimizing the total number of bends. One reason for this is that minimizing the length favors having each label close to the point being labeled. This results in a label assignment where the vertical order of the labels tends to reflect the vertical order of the points in the figure fairly well. In contrast, when minimizing the number of bends this correspondence is more easily lost, which can be confusing, compare Figures 6b and 6c. In addition, the longer the hand segments are, the harder they are to follow and this is not considered in bad_{bend}. Nevertheless, although direct leaders are easy to read, their number should not be maximized without considering the shape and length of the non-direct leaders. Therefore the hybrid badness function applied in Figures 6d and 6f is designed to find a good compromise between both optimization goals.

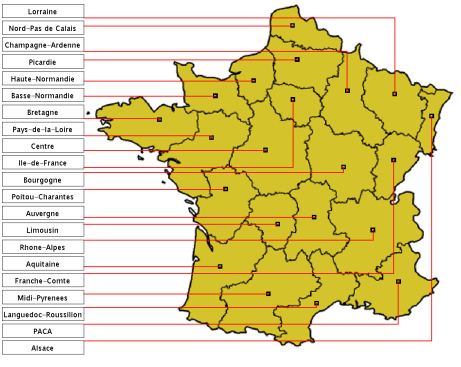
[2] The applet is available at `http://i11www.iti.uni-karlsruhe.de/labeling`



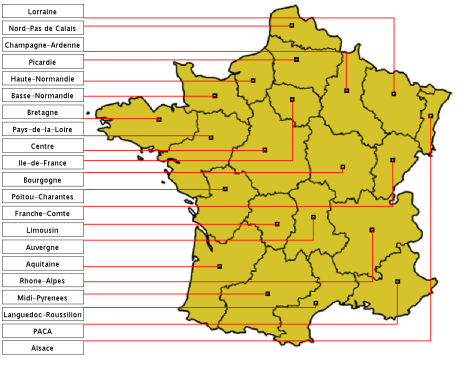
(a) *po*-leaders and badness bad_{len}.



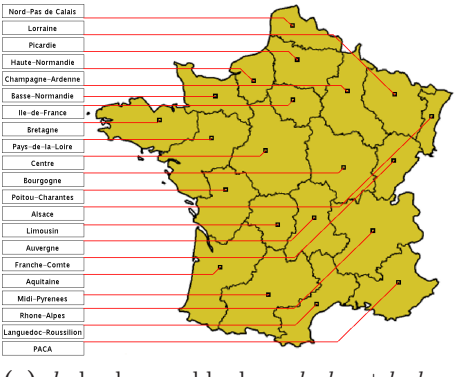
(b) *po*-leaders and badness $bad_{\text{len}} + bad_{\text{cls}}$.



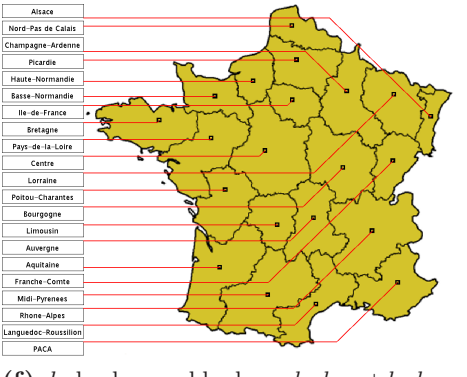
(c) *po*-leaders and badness $bad_{\text{bend}} + bad_{\text{cls}}$.



(d) *po*-leaders and badness $bad_{\text{hyb}} + bad_{\text{cls}}$.



(e) *do*-leaders and badness $bad_{\text{len}} + bad_{\text{cls}}$.



(f) *do*-leaders and badness $bad_{\text{hyb}} + bad_{\text{cls}}$.

Fig. 6. One-sided labelings for the mainland regions of France

Conclusion. We find that minimizing the length is more important for the aesthetic quality of a labeling than minimizing the bends. Combining both aspects in a hybrid badness function leads to a good compromise between the two objectives. Furthermore the closeness term bad_{cls} turned out to be of great importance for good labelings.

5 Concluding Remarks

An interesting future task is to reflect the interference of a leader and the background image in the badness function.

We also looked at the case where the labels are placed on two opposite sides of the point-containing rectangle. Using dynamic programming and similar ideas as for the one-sided case (a split line that splits a subproblem into two two-sided subproblems), we could establish an $O(n^8)$- and $O(n^{14})$-time algorithm for the *po*- and *do*-leaders, respectively. Unfortunately, not only the asymptotical running times of these algorithms were bad, it also turned out that these algorithms are useless in practice since they do not compute a result in acceptable time.

Hence, for producing two-sided labelings in practice we suggest to use the $O(n^2)$-time *po*-leader length-minimization algorithm of Bekos et al. [4] or to split the instance in the middle and solve the resulting one-sided problems. We leave it as an open problem to find efficient algorithms for dividing points between the left and the right side in an appropriate fashion to find good two-sided *po*- and *do*-labelings. Note that splitting in the middle does in general not yield aesthetically good results. For the *do*-leaders a feasible instance can even become infeasible by splitting in the middle.

References

1. Ali, K., Hartmann, K., Strothotte, T.: Label layout for interactive 3D illustrations. J. of WSCG 13, 1–8 (2005)
2. Bekos, M.A., Kaufmann, M., Potika, K., Symvonis, A.: Multi-stack boundary labeling problems. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 81–92. Springer, Heidelberg (2006)
3. Bekos, M.A., Kaufmann, M., Potika, K., Symvonis, A.: Polygon labelling of minimum leader length. In: Misue, K., Sugiyama, K., Tanaka, J. (eds.) APVIS 2006. Proc. Asia Pacific Symp. on Inform. Visualisation, CRPIT, vol. 60, pp. 15–21 (2006)
4. Bekos, M.A., Kaufmann, M., Symvonis, A., Wolff, A.: Boundary labeling: Models and efficient algorithms for rectangular maps. Computational Geometry: Theory & Applications 36, 215–236 (2007)
5. Fekete, J.-D., Plaisant, C.: Excentric labeling: Dynamic neighborhood labeling for data visualization. In: CHI 1999. Proc. of the SIGCHI conference on Human factors in computing systems, pp. 512–519 (1999)
6. Wolff, A., Strijk, T.: The map-labeling bibliography (2006), http://i11www.iti.uni-karlsruhe.de/~awolff/map-labeling/bibliography/

Multi-circular Layout of Micro/Macro Graphs*

Michael Baur[1] and Ulrik Brandes[2]

[1] Department of Computer Science, Universität Karlsruhe (TH), Germany
baur@informatik.uni-karlsruhe.de
[2] Department of Computer & Information Science, University of Konstanz, Germany
Ulrik.Brandes@uni-konstanz.de

Abstract. We propose a layout algorithm for micro/macro graphs, i.e. relational structures with two levels of detail. While the micro-level graph is given, the macro-level graph is induced by a given partition of the micro-level vertices. A typical example is a social network of employees organized into different departments. We do not impose restrictions on the macro-level layout other than sufficient thickness of edges and vertices, so that the micro-level graph can be placed on top of the macro-level graph. For the micro-level graph we define a combinatorial multi-circular embedding and present corresponding layout algorithms based on edge crossing reduction strategies.

1 Introduction

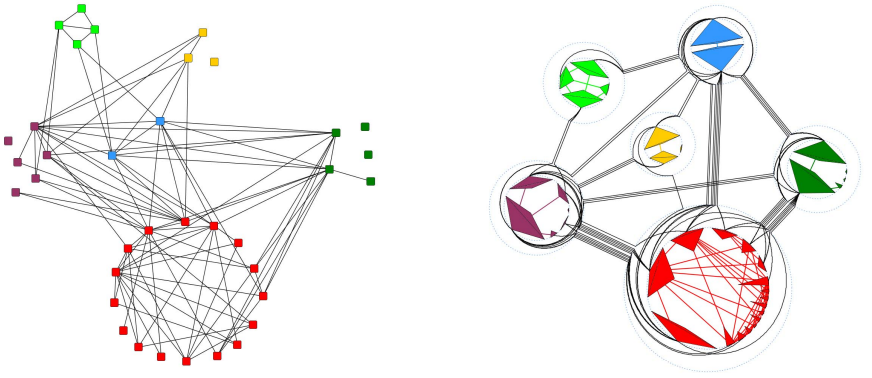
An important aspect in the visualization of many types of networks is the interplay between fine- and coarse-grained structures. Think, for instance, of low-level interaction giving rise to emergent features at a larger scale, or people implementing organizational relations. Assuming that the structure on the micro level is a graph, a macro-level graph may originate from a group-level network analysis such as clustering or role analysis (e.g., [5]), from an attribute-based partitioning of the vertices, or may just be given in advance.

Depending on the particular application domain and other contexts, different layout methods will be appropriate for the macro graph. Since we only require large nodes and thick edges, we assume it is given. Either the macro-level layout algorithm can handle varying vertex size (e.g., [12,21]) and edge thickness (e.g., [7]), or some post-processing is applied (e.g., [11]).

Given a drawing of the macro-level graph with large nodes and thick edges, each vertex of the micro-level graph is drawn in the area defined by the macro vertex it belongs to, and each micro edge is routed through its corresponding macro edge. We propose a multi-circular layout model for the micro graph. Each micro vertex is placed on a circle inside of the area of its corresponding macro vertex and micro edges whose end vertices belong to the same macro vertex are drawn inside of these circles. All other micro edges are then drawn inside of their corresponding macro edges and at constant but different distances from the border of the macro edge, i.e. in straight-line macro edges they are

* Research partially supported by DFG, grants Wa 654/13-2 and Br 2158/2-3.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 255–267, 2007.



(a) geometric grouping and straight-line edges

(b) multi-circular layout

Fig. 1. (a) Example organizational network with geometric grouping and straight-line edges (redrawn from [15]). In our multi-circular layout (b), all details are still present and the macro structure induced by the grouping becomes visible. The height and width of the vertices reflects the number of connections within and between groups.

drawn as parallel lines. These edges must also be routed inside the area of macro vertices to connect to their endpoints, but are not allowed to cross the circles. In principle, an arbitrary layout strategy can be used as long as it complies with these requirements. Figure 1 shows a concrete example of this model. Micro edges connecting vertices in the same macro vertex are drawn as straight lines. Inside of macro vertices, the other edges spiral around the circle of micro vertices until they reach the area of the macro edge. We give a combinatorial description of the above model and then focus on the algorithmically most challenging aspect of these layouts, namely crossing reduction by cyclic ordering of micro vertices and choosing edge winding within macro vertices. Finally, we apply the multi-circular layout to an email communication network to exemplify its use case.

While the drawing convention consists of proven components (geometric grouping is used, e.g., in [15,20], and edge routing to indicate coarse-grained structure is proposed in, e.g., [13,3]), our approach is novel in the way we organize micro vertices to let the macro structure dominate the visual impression without cluttering the micro-level details too much. Note also that the setting is very different from layout algorithms operating on structure-induced clusterings (e.g., [14,1]), since we cannot make any assumptions on the structure of clusters (they may even consist of isolates). Therefore, we neither want to utilize the clustering for better layout, nor do we want to display the segregation into dense subregions or small cuts. Our aim is to represent the interplay between a (micro-level) graph and a (most likely extrinsic) grouping of its vertices.

After defining some basic terminology in Sect. 2, we state required properties for macro-graph layout in Sect. 3. Multi-circular micro-graph layout is discussed in more detail in Sect. 4 and crossing reduction algorithms for it are given in Sect. 5. We conclude with an application in Sect. 6.

2 Preliminaries

Throughout this paper, let $G = (V, E)$ be a simple undirected graph with $n = |V|$ vertices and $m = |E|$ edges. Furthermore, let $E(v) = \{\{u, v\} \in E : u \in V\}$ denote the incident edges of a vertex $v \in V$, let $N(v) = \{u \in V : \{u, v\} \in E\}$ denote its neighbors, and let $sgn : \mathbb{R} \to \{-1, 0, 1\}$ be the signum function.

Since each micro-vertex is required to belong to exactly one macro-vertex, the macro structure defines a clustering, or partitioning, of the micro-vertices. Contrary to this top-down approach, we can also start from the bottom. A *partition assignment* $\phi : V \to \{0, \ldots, k-1\}$ for G subdivides the (micro-)vertex set V into k pairwise disjoint subsets $V = V_0 \dot\cup \ldots \dot\cup V_{k-1}$, where $V_i = \{v \in V : \phi(v) = i\} = \phi^{-1}(i)$. An edge $e = \{u, v\} \in V_i \times \in V_j$ is called an *intra-partition edge* iff $i = j$, otherwise it is called an *inter-partition edge*. The set of intra-partition edges of a partition V_i is denoted by E_i, the set of inter-partition edges of two partitions V_i, V_j by $E_{i,j}$. We use $G = (V, E, \phi)$ to denote a graph $G = (V, E)$ and a related partition assignment ϕ.

A *circular order* $\pi = \{\pi_0, \ldots, \pi_{k-1}\}$ defines for each partition V_i a vertex order π_i as a bijective function $\pi_i : V_i \to \{0, \ldots, |V_i| - 1\}$ with $u \prec v \Leftrightarrow \pi_i(u) < \pi_i(v)$ for any two vertices $u, v \in V_i$. An order π_i can be interpreted as a counter-clockwise sequence of distinct positions on the circumference of a circle.

3 Macro Layout

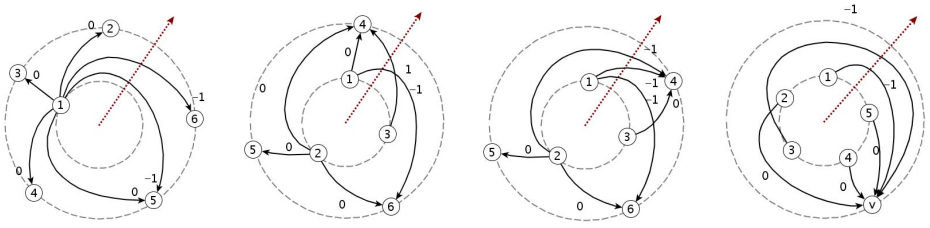
A prototypical macro graph, the *quotient graph*, is defined by a partition assignment. Given a partition assignment $\phi : V \to \{0, \ldots, k-1\}$, the corresponding quotient graph $Q(G, \phi) = (V_Q, E_Q)$ contains a vertex for each partition of G and two vertices $V_i, V_j \in V_Q$ are connected iff E contains at least one edge between a vertex in V_i and a vertex in V_j.

We do not require a specific layout strategy for the macro graph as long as its elements are rendered with sufficient thickness to draw the underlying micro graph on top of them. To achieve this, post-processing can be applied to any given layout [11] or methods which consider vertex size (e.g., [12,21]) and edge thickness (e.g., [7]) have to be used.

From a macro layout we get *partition orders* $\Pi_i : V_Q \setminus V_i \to \{0, .., \deg(V_i) - 1\}$ for each partition V_i, defined by the sequence of its incident edges in $Q(G, \phi)$, and a partition order $\Pi = \{\Pi_0, \ldots, \Pi_{k-1}\}$ for G. For each macro vertex this can be seen as a counter-clockwise sequence of distinct docking positions for its incident (macro) edges on its border.

4 Micro Layout

Before we discuss the multi-circular layout model for the micro graph, let us recall the related concepts of (single) circular and radial embeddings. In *(single) circular layouts* all vertices are placed on a single circle and edges are drawn as



(a) some incident edges (b) node 4 is at position 0 (c) node 4 rotated to position 2 (d) without parting

Fig. 2. Radial layouts. Edges are labeled with their winding value.

straight lines. Therefore, a *(single) circular embedding* ε of a graph $G = (V, E)$ is fully defined by a vertex order π, i.e. $\varepsilon = \pi$ [4]. Two edges $e_1, e_2 \in E$ *cross* in ε iff the end vertices of e_1, e_2 are encountered alternately in a cyclic traversal.

4.1 Radial Layout

In *radial layouts* the partitions are placed on nested concentric circles (*levels*) and edges are drawn as curves between consecutive partitions. Therefore, only graphs $G = (V, E)$ with a *proper* partition assignment $\phi : V \to \{0, \ldots, k-1\}$ are allowed, i.e. $|\phi(u) - \phi(v)| = 1$ for all edges $\{u, v\} \in E$. For technical reasons, edges are considered to be directed from lower to higher levels.

Recently, Bachmaier [2] investigated such layouts. They introduced a *ray* from the center to infinity to mark the start and end of the circular vertex orders. Using this ray it is also possible to count how often and in which direction an edge is wound around the common center of the circles. We call this the *winding* $\psi : E \to \mathbb{Z}$ of an edge (*offset* in [2]). $|\psi(e)|$ counts the number of crossings of the edge with the ray and the sign reflects the mathematical direction of rotation. See Figure 2 for some illustrations. Finally, a *radial embedding* ε of a graph $G = (V, E, \phi)$ is defined to consist of a vertex order π and an edge winding ψ, i.e. $\varepsilon = (\pi, \psi)$. Note that the rotation of a partition without permuting the vertices changes the positions and winding values but not the number of crossings.

Crossings between edges in radial embeddings depend on their winding and on the order of the end vertices. There can be more than one crossing between two edges if they have very different winding. We denote the number of crossings between two edges $e_1, e_2 \in E$ in an radial embedding ε by $\chi_\varepsilon(e_1, e_2)$. The (radial) crossing number of an embedding ε and a level graph $G = (V, E, \phi)$ is then naturally defined as $\chi(\varepsilon) = \sum_{\{e_1,e_2\} \in E, e_1 \neq e_2} \chi_\varepsilon(e_1, e_2)$ and $\chi(G) = \min\{\chi(\varepsilon) : \varepsilon$ is a radial embedding of $G\}$ is called the *radial crossing number* of G.

Theorem 1 ([2]). *Let $\varepsilon = (\pi, \psi)$ be a radial embedding of a 2-level graph $G = (V_1 \dot\cup V_2, E, \phi)$. The number of crossings $\chi_\varepsilon(e_1, e_2)$ between two edges $e_1 = (u_1, v_1) \in E$ and $e_2 = (u_2, v_2) \in E$ is*

$$\chi_\varepsilon(e_1,e_2) = \max\left\{0, \left|\psi(e_2) - \psi(e_1) + \frac{b-a}{2}\right| + \frac{|a|+|b|}{2} - 1\right\} ,$$
$$\textit{where } a = sgn(\pi_1(u_2) - \pi_1(u_1)) \textit{ and } b = sgn(\pi_2(v_2) - \pi_2(v_1)) \ .$$

Bachmaier also states that in crossing minimal radial embeddings every pair of edges crosses at most once and incident edges do not cross at all. As a consequence, only embeddings need to be considered where there is a clear *parting* between all edges incident to the same vertex u. The parting is the position of the edge list of u that separates the two subsequences with different winding values. See Figure 2 for layouts with and without proper parting.

4.2 Multi-circular Layouts

Unless otherwise noted, vertices and edges belong to the micro-level in the following. In the micro layout model each vertex is placed on a circle inside of its corresponding macro vertex. Intra-partition edges are drawn within these circles as straight lines. Inter-partition edges are drawn inside their corresponding macro edges and at constant but different distances from the border of the macro edge. To connect to their incident vertices, this edges must also be routed inside of macro vertices. Since they are not allowed to cross the circles, they are drawn as curves around them. We call such a drawing a *(multi-)circular layout*.

Since intra- and inter-partition edges can not cross, all crossings of intra-partition edges are completely defined by the vertex order π_i of each partition V_i. Intuitively speaking, a vertex order defines a circular layout for the intra-partition edges. In the following we thus concentrate on inter-partition edges.

The layout inside each macro vertex V_i can be seen as a 2-level radial layout. The orders can be derived from the vertex order π_i and the partition order Π_i. Similar to radial layouts we introduce a *ray* for each partition and define the beginning of the orders and the edge winding according to these rays. Note that for each edge $e = \{u, v\} \in E$, $u \in V_i$, $v \in V_j$, two winding values are needed, one for the winding around partition V_i denoted by $\psi_i(e) = \psi_u(e)$, and one for the winding around partition V_j denoted by $\psi_j(e) = \psi_v(e)$. If the context implies an implicit direction of the edges we call windings either source or target windings respectively. Since radial layouts can be rotated without changing the embedding, rays of different partitions are independent and can be arbitrary directed. Finally, a *multi-circular embedding* ε is defined by a vertex order π, a partition order Π, and the winding of the edges ψ, i.e. $\varepsilon = (\pi, \Pi, \psi)$.

Observation 2. *For each partition V_i in a multi-circular embedding $\varepsilon = (\pi, \Pi, \psi)$ a 2-level radial embedding $\varepsilon_i = ((\pi_i, \pi'), \psi_i)$ is defined by the vertex order π_i, the partition order Π_i, and the edge winding ψ_i, where $\pi'(v) = \Pi_i(\phi(v)), v \in V \setminus V_i$.*

There is another connection between radial and multi-circular layouts. A 2-level radial layout can easily be transformed in a 2-partition circular layout and vice versa. Given a graph $G = (V_1 \dot\cup V_2, E, \phi)$ and a radial embedding $\varepsilon = (\pi, \psi)$ of G, the 2-partition circular embedding $\varepsilon^* = (\pi^*, \Pi^*, \psi^*)$ defined by $\pi_1^* = \pi_1$, $\pi_2^* = -\pi_2$,

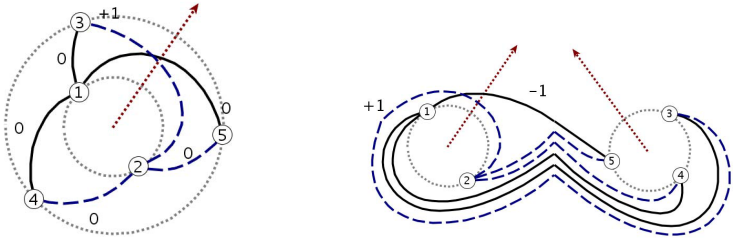


Fig. 3. A 2-level radial layout and its corresponding 2-circular layout

$\Pi_1^* = 0$, $\Pi_2^* = 0$, and $\psi_1^*(e) = \psi(e)$, $\psi_2^*(e) = 0$ realizes exactly the same crossings. See Figure 3 for an example. Intuitively speaking, the topology of the given radial embedding is not changed if we drag the two circles apart and reverse one of the vertex orders. If a 2-partition circular embedding $\varepsilon^* = (\pi^*, \Pi^*, \psi^*)$ is given, a related radial embedding $\varepsilon = (\pi, \psi)$ is defined by $\pi_1 = \pi_1^*$, $\pi_2 = -\pi_2^*$, and $\psi(e) = \psi_1(e) - \psi_2(e)$.

Observation 3. *There is a one-to-one correspondence between a 2-level radial embedding and a 2-circular embedding.*

Crossings in the micro layout are due to either the circular embedding or crossing macro edges. Since crossings of the second type can not be avoided by changing the micro layout, we do not consider them in the micro layout model. Obviously, pairs of edges which are not incident to a common macro vertex can only cause crossings of this type. For pairs of edges which are incident to at least one common macro vertex we can define corresponding 2-level radial layouts using Observations 2 and 3 and compute the number of crossings by modifications of Theorem 1.

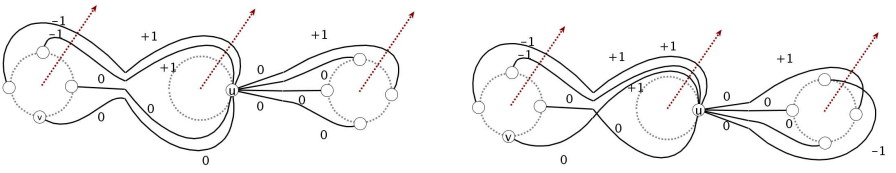
Theorem 4. *Let $\varepsilon = (\pi, \Pi, \psi)$ be a multi-circular embedding of a graph $G = (V, E, \phi)$ and let $e_1 = \{u_1, v_1\}$, $e_2 = \{u_2, v_2\} \in E$ be two inter-partition edges.*

If e_1 and e_2 share exactly one common incident macro vertex, e.g., $V_i = \phi(u_1) = \phi(u_2)$, $\phi(v_1) \neq \phi(v_2)$, then the number of crossings of e_1 and e_2 is

$$\chi_\varepsilon(e_1, e_2) = \max\left\{0, \left|\psi_i(e_2) - \psi_i(e_1) + \frac{b-a}{2}\right| + \frac{|a|+|b|}{2} - 1\right\},$$
$$\text{where } a = sgn(\pi_i(u_2) - \pi_i(u_1)) \text{ and } b = sgn(\Pi(\phi(v_2)) - \Pi(\phi(v_1))).$$

If e_1 and e_2 belong to the same macro edge, e.g., $V_i = \phi(u_1) = \phi(u_2)$, $V_j = \phi(v_1) = \phi(v_2)$, then the number of crossings of e_1 and e_2 is

$$\chi_\varepsilon(e_1, e_2) = \max\left\{0, \left|\psi'(e_2) - \psi'(e_1) + \frac{b-a}{2}\right| + \frac{|a|+|b|}{2} - 1\right\},$$
$$\text{where } a = sgn(\pi_i(u_2) - \pi_i(u_1)),\ b = sgn(\pi_j(v_1) - \pi_j(v_2)), \text{ and}$$
$$\psi'(e) = \psi_i(e) + \psi_j(e).$$



(a) parting (b) edge $\{u, v\}$ violates parting

Fig. 4. Not all winding combinations for the incident edges of u result in a good layout

Similar to radial layouts, in a crossing minimal multi-circular embedding incident edges do not cross and there is at most one crossing between every pair of edges. Therefore, only embeddings need to be considered where there is a clear *parting* between all edges incident to the same vertex $u \in V_i$. Since in multi-circular layouts winding in different macro vertices can be defined independently, we split the edge list $E(u)$ of u by target partitions and get edge lists $E(u)_j = \{\{u, v\} \in E(u) : v \in V_j\}$. For each list $E(u)_j$, we get a position ℓ_j that separates the two subsequences with different values of winding ψ_j and defines the parting for this partition. Furthermore, there is also a parting for V_i defined on the edge list $E(u)$. The order of $E(u)$ for this parting depends on the partings ℓ_j in the target partitions V_j. Edges are sorted by the partition order, and for edges to the same partition V_j, ties are broken by the reverse vertex order started not at the ray but at the parting position ℓ_j. Then, the parting for V_i is the position ℓ_i which separates different values of winding ψ_i in the so ordered list. See Figure 4 for a layout with parting and a layout where the edge $\{u, v\}$ violates the parting.

Corollary 1. *Multi-circular crossing minimization is $\mathcal{NP}$-hard.*

Proof. Single circular and radial crossing minimization [2,17] are $\mathcal{NP}$-hard. As we have already seen, these two crossing minimization problems are subproblems of the multi-circular crossing minimization problem, proofing the corollary. □

As a consequence, we do not present exact algorithms for crossing minimization in multi-circular layouts. Instead, we propose extensions of some well known crossing reduction heuristics for horizontal and radial crossing reduction.

5 Layout Algorithms

Since the drawing of inter-partition edges inside a macro vertex can be seen as a radial drawing, a multi-circular layout can be composed of separate radial layouts for each macro vertex (for instance using the techniques of [20,10,2]. Such a decomposition approach, however, is inappropriate since intra-partition edges are not considered at all and inter-partition edges are not handled adequately due to the lack of information about the layout at the other macro vertices. E.g., choosing a path with more crossings in one macro vertex can allow a routing with much less crossings on the other side.

Nevertheless, we initially present in this section adaptations of radial layout techniques because they are quite intuitive, fast, and simple, and can be used for the evaluation of more advanced algorithms.

5.1 Barycenter and Median Layouts

The basic idea of both the barycenter and the median layout heuristic is the following: each vertex is placed in a central location computed from the positions of its neighbors - in either the barycenter or the median position - to reduce edge lengths and hence the number of crossings. For a 2-level radial layout, the *Cartesian Barycenter* heuristic gets the two levels and a fixed order for one of them. All vertices of the fixed level are set to equidistant positions on a circle and the component-wise barycenter for all vertices of the second level is computed. The cyclic order around the center defines the order of the vertices and the edges are routed along the geometrically shortest-path. The *Cartesian Median* heuristic is defined similar. Running time for both heuristics is in $\mathcal{O}(|E| + |V| \log |V|)$.

Both heuristics are easily extended for multi-circular layouts. The layout in each macro vertex V_i is regarded as a separate 2-level radial layout as described in Observation 3 and the partition orders Π_i are used to define the orders of the fixed levels. Because of the shortest-path routing, no two edges cross more than once and incident edges do not cross at all in the final layout. On the other hand are crossings avoided by the used placement and winding strategies only indirectly by edge length reduction.

5.2 Multi-circular Sifting

To overcome the drawbacks of the radial layout algorithms described before, we propose an extension of the sifting heuristic which computes a complete multi-circular layout and considers edge crossings for optimizing both vertex order and edge winding, and thus is expected to generate better layouts.

Sifting was originally introduced as a heuristic for vertex minimization in ordered binary decision diagrams [19] and later adapted for the layered one-sided, the circular, and the radial crossing minimization problems [18,4,2]. The idea is to keep track of the objective function while moving a vertex along a fixed order of all other vertices. The vertex is then placed in its (locally) optimal position. The method is thus an extension of the greedy-switch heuristic [8]. For crossing reduction the objective function is the number of crossings between the edges incident to the vertex under consideration and all other edges. In multi-circular layouts this function depends on both the vertex order and the edge winding. Therefore, we have to find for each position of a vertex the winding values for its incident edges which result in the minimal crossing number.

The efficient computation of crossing numbers in sifting for layered and single circular layouts is based on the locality of crossing changes, i.e. swapping consecutive vertices $u \hookrightarrow v$ only affects crossings between edges incident to u with edges incident to v. In multi-circular layouts this property clearly holds for intra-partition edges since they form (single-)circular layouts. For inter-partition edges

the best routing path may require an update of the windings. Such a change can affect crossings with all edges incident to the involved partitions.

Since swapping the positions of two consecutive vertices (and keeping the winding values) only affects incident edges, the resulting change in the number of crossings can be efficiently computed. Therefore, we need an efficient update strategy for edge windings while $u \in V_i$ moves along the circle. We do not consider each possible combination of windings for each position of u. but keep track of the parting of the edges. Note that we have to alter simultaneously the parting for the source partition and all the partings for the target partitions because for an edge, a changed winding in the source partition may allow a better routing with changed winding in the target partition. Intuitively speaking, the parting in the source partition should move around the circle in the same direction as u, but on the opposite side of the circle, while the parting in the target partitions should move in the opposite direction. Otherwise, edge lengths increase and with it the likelihood of crossings. Thus, we start with winding values $\psi_u(e) = 1$ and $\psi_v(e) = 1$ for all $e = \{u, v\} \in E(v)$ and iteratively move parting counters around the circles and mostly decrease this values in the following way:

1. First try to improve the parting at V_i, i.e. the value of ψ_u for the current parting edge is decreased and the parting moved counter-clockwise to the next edge, until this parting can no longer be improved.
2. For edges whose source winding were changed in step one, there may be better target windings which can not be found in step three, because the value of ψ_j has to be increased, i.e. for each affected edge, the value of ψ_j for the edge is increased until no improvement is made.
3. Finally try to improve the parting for each target partition V_j separately, i.e. for each V_j the value of ψ_j for the current parting edge is decreased and the parting moved clockwise to the next edge, until this parting can no longer be improved.

After each update, we ensure that all counters are valid and that winding values are never increased above 1 and below -1.

Based on the above, the locally optimal position of a single vertex can be found by iteratively swapping the vertex with its neighbor and updating the edge winding while keeping track of the change in crossing number. After the vertex has past each position, it is placed where the intermediary crossing counts reached their minimum. Repositioning each vertex once in this way is called a *round of sifting.*

Theorem 5. *The running time of multi-circular sifting is in* $\mathcal{O}(|V| \cdot |E|^2)$.

Proof. Computing the difference in cross count after swapping two vertices requires $\mathcal{O}(|E|^2)$ running time for one round of sifting. For each edge the winding changes only a constant number of times because values are bounded, source winding and target winding are decreased in steps one and three resp., and the target winding is only increased for edges whose source winding decreased before. Counting the crossings of an edge after changing its winding takes time

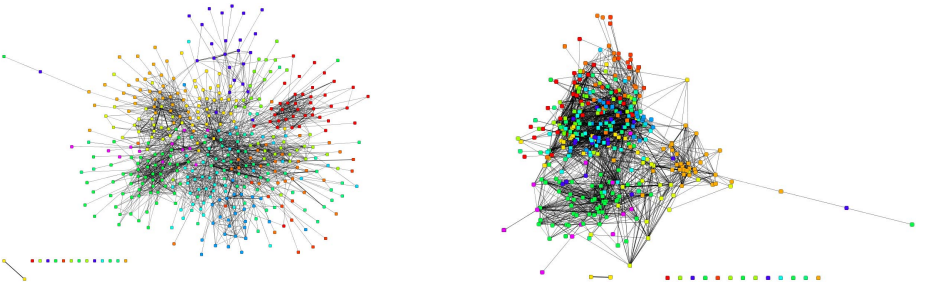


Fig. 5. Drawings of the email network generated by a force-directed method (left) and by multi-dimensional scaling (MDS, right)

$\mathcal{O}(|E|)$. For each vertex $u \in V$ the windings are updated $\mathcal{O}(|V| \cdot \deg(u))$ times, once per position and once per shifted parting. For one round, this results in $\mathcal{O}(|V||E|)$ winding changes taking time $\mathcal{O}(|V| \cdot |E|^2)$. □

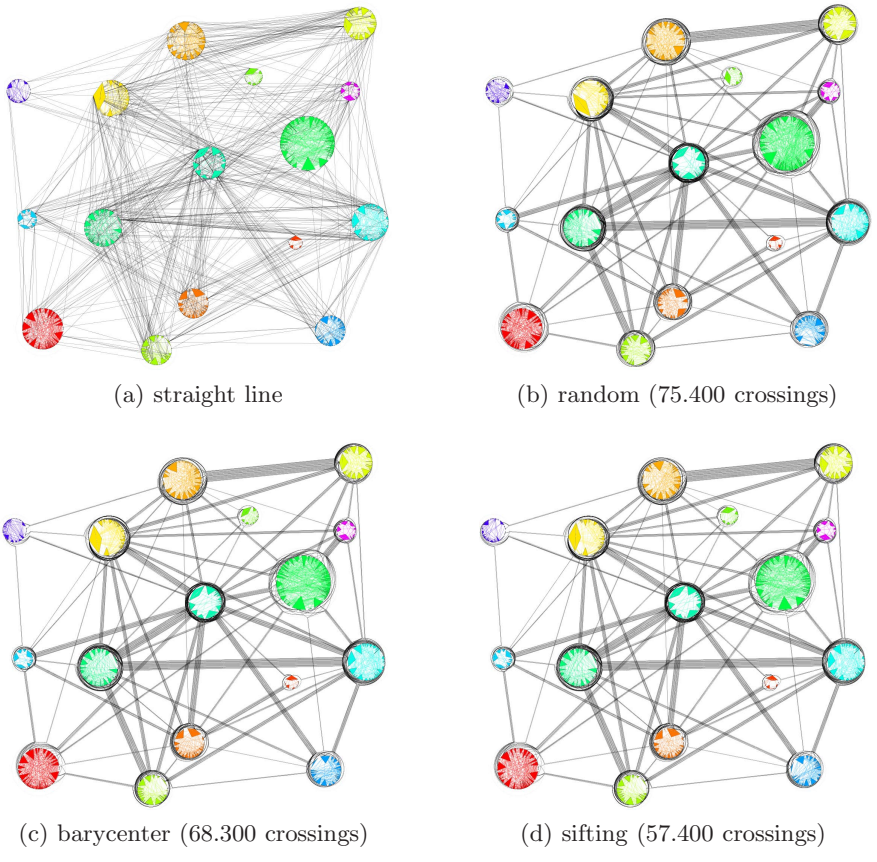
6 Application: Email Communication Network

The strength of a multi-circular layout is the coherent drawing of vertices and edges at the two levels of detail. It reveals structural properties of the macro graph and allows identification of micro level connections at the same time. The showcase for the benefits of our micro/macro layout is a email communication network of a department of the Universität Karlsruhe. The micro graph consists of 442 anonymized department members and 2,201 edges representing at least one email communication in the considered time frame of five weeks. At the macro level, a grouping into 16 institutes is given, resulting in 66 macro edges.

We start by inspecting drawings generated by a general force-directed approach similar to [9] and by multi-dimensional scaling (MDS) [6], see Figure 5. Both methods tend to place adjacent vertices near each other but ignore the additional grouping information. Therefore, it is not surprising that the drawings do not show a geometric clustering and the macro structure can not be identified. Moreover, it is difficult or even impossible to follow edges since they overlap each other.

More tailored for the drawing of graphs with additional vertex grouping are the layout used by Krebs [15], and the force-directed attempts to assign vertex positions by Six and Tollis [20] and Krempel [16]. All three methods place the vertices of each group on circles inside of separated geometric areas. While some efforts are made to find good vertex positions on the circles, edges are simply drawn as straight lines. Figure 6 (a) gives a prototypical example of this layout style. Although these methods feature a substantial progress compared to general layouts and macro vertices are clearly visible, there is no representation of macro edges and so the overall macro structure is still not identifiable.

Finally, we layouted the email network according to the micro/macro drawing convention. Its combinatorial descriptions allows for an enrichment with an



(a) straight line (b) random (75.400 crossings)

(c) barycenter (68.300 crossings) (d) sifting (57.400 crossings)

Fig. 6. Multi-circular layouts of the email network

analytical visualization of the vertices. In the Figures 1 and 6 the length of the circular arc a vertex covers is proportional to its share of the total inter-partition edges of this group. The height from its chord to the center of the circle reflects the fraction of present to possible intra-edges.

To investigate the effect of improved vertex orders and appropriate edge windings, we compare two variations of multi-circular layouts: shortest-path edge winding combined with random vertex placement and with barycenter vertex placement, see Figure 6. The macro structure of the graph is apparent at first sight. Since the placement of the vertex circles is the same as in Figure 6 (a), this improvement clearly follows from the grouping of micro edges. A closer look reveals the drawback of random placement: edges between different groups have to cover a long distance around the vertex circles and are hard to follow. Also a lot of edge crossings are generated both inside of the groups and in the area around the vertex placement circles. Assigning vertex positions according to the barycenter heuristic results in a clearly visible improvement and allows the differentiation of some of the micro edges. Using sifting improves the layout even

further, resulting from a decrease of the number of crossings from more than 75.000 to 57.400 in the considered email network. The time for computing the layout of this quiet large graph is below half a minute.

7 Conclusion

We proposed a drawing convention for micro/macro graphs where micro-level elements are drawn on top of the elements of the coarse macro graph, so that the contribution of micro-level elements to macro-level structure becomes apparent. Since there is no need to place restrictions on the layout of the macro graph, we assumed it is given and focused on layouts of the micro graph. We presented a multi-circular layout model and investigated layout strategies based on crossing reduction techniques for it.

Backed by the visualizations of the email communication network computed by an initial implementation of our algorithms we claim that the grouping of micro-edges into macro-edges according to the micro/macro drawing convention exhibits benefits over layouts which group the vertices. Furthermore, since vertex orders and edge windings have a large effect on the readability of multi-circular layouts, it is justified to spend a larger effort to improve them.

A major benefit of the multi-circular layout is it combinatorial description since it allows the combination with other visualization techniques to highlight some graph properties or to further improve the visual appearance. A very interesting aspect would be the combination with Holten's [13] edge bundling technique.

References

1. Archambault, D., Munzner, T., Auber, D.: Topolayout: Multi-level graph layout by topological features. IEEE Trans. Visual. and Comp. Graphics 13(2), 305–317 (2007)
2. Bachmaier, C.: A radial adaptation of the sugiyama framework for visualizing hierarchical information. IEEE Trans. Visual. and Comp. Graphics 13(3), 585–594 (2007)
3. Balzer, M., Deussen, O.: Level-of-detail visualization of clustered graph layouts. In: APVIS 2007. Asia-Pacific Symposium on Visualisation 2007 (2007)
4. Baur, M., Brandes, U.: Crossing reduction in circular layouts. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 332–343. Springer, Heidelberg (2004)
5. Brandes, U., Erlebach, T. (eds.): Network Analysis. LNCS, vol. 3418. Springer, Heidelberg (2005)
6. Cox, T.F., Cox, M.A.A.: Multidimensional Scaling. In: Monographs on Statistics and Applied Probability, 2nd edn., Chapman & Hall/CRC (2001)
7. Duncan, C.A., Efrat, A., Kobourov, S.G., Wenk, C.: Drawing with fat edges. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 162–177. Springer, Heidelberg (2002)
8. Eades, P., Kelly, D.: Heuristics for reducing crossings in 2-layered networks. Ars Combinatoria 21(A), 89–98 (1986)

9. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Software - Practice and Experience 21(11), 1129–1164 (1991)
10. Gansner, E.R., Koren, Y.: Improved circular layouts. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 386–398. Springer, Heidelberg (2007)
11. Gansner, E.R., North, S.C.: Improved force-directed layouts. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 364–373. Springer, Heidelberg (1999)
12. Harel, D., Koren, Y.: Drawing graphs with non-uniform vertices. In: AVI 2002. Proc. Work. Conf. on Advanced Visual Interfaces, pp. 157–166. ACM Press, New York (2002)
13. Holten, D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. IEEE Trans. Visual. and Comp. Graphics 12(5), 741–748 (2006)
14. Kaufmann, M., Wiese, R.: Maintaining the mental map for circular drawings. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 12–22. Springer, Heidelberg (2002)
15. Krebs, V.E.: Visualizing human networks. Release 1.0, pp. 1–25 (February 1996)
16. Krempel, L.: Visualisierung komplexer Strukturen. Grundlagen der Darstellung mehrdimensionaler Netzwerke. Campus (2005)
17. Masuda, S., Kashiwabara, T., Nakajima, K., Fujisawa, T.: On the $\mathcal{NP}$-completeness of a computer network layout problem. In: Proc. 20th IEEE Int. Symposium on Circuits and Systems 1987, pp. 292–295 (1987)
18. Matuszewski, C., Schönfeld, R., Molitor, P.: Using sifting for k-layer straightline crossing minimization. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 217–224. Springer, Heidelberg (1999)
19. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: Proc. IEEE/ACM Conf. Computer-Aided Design, pp. 42–47. IEEE Society, Los Alamitos (1993)
20. Six, J.M., Tollis, I.G.: A framework for user-grouped circular drawings. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 135–146. Springer, Heidelberg (2004)
21. Wang, X., Miyamoto, I.: Generating customized layouts. In: Brandenburg, F.J. (ed.) GD 1995. LNCS, vol. 1027, pp. 504–515. Springer, Heidelberg (1996)

Constrained Simultaneous and Near-Simultaneous Embeddings

Fabrizio Frati[1], Michael Kaufmann[2], and Stephen G. Kobourov[3]

[1] Dipartimento di Informatica e Automazione – Università Roma Tre, Italy
[2] Wilhelm-Schickard-Institut für Informatik – Universität Tübingen, Germany
[3] Department of Computer Science – University of Arizona, U.S.A.

Abstract. A *geometric simultaneous embedding* of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with a bijective mapping of their vertex sets $\gamma : V_1 \to V_2$ is a pair of planar straight-line drawings Γ_1 of G_1 and Γ_2 of G_2, such that each vertex $v_2 = \gamma(v_1)$ is mapped in Γ_2 to the same point where v_1 is mapped in Γ_1, where $v_1 \in V_1$ and $v_2 \in V_2$.

In this paper we examine several constrained versions and a relaxed version of the geometric simultaneous embedding problem. We show that if the input graphs are assumed to share no common edges this does not seem to yield large classes of graphs that can be simultaneously embedded. Further, if a prescribed combinatorial embedding for each input graph must be preserved, then we can answer some of the problems that are still open for geometric simultaneous embedding. Finally, we present some positive and negative results on the near-simultaneous embedding problem, in which vertices are not mapped exactly to the same but to "near" points in the different drawings.

1 Introduction

Graph drawing techniques are commonly used to visualize relationships between objects, where the objects are the vertices of the graph and the relationships are captured by the edges in the graph. Simultaneous embedding is a problem that arises when visualizing two or more relationships defined on the same set of objects. If the graphs corresponding to these relationships are planar, the aim of simultaneous embedding is to find point locations in the plane for the vertices of the graphs, so that each of the graphs can be realized on the same point-set without edge crossings. To ensure good readability of the drawings, it is preferable if the edges are drawn as straight-line segments. This problem is known as *geometric simultaneous embedding*. It has been shown that only a few classes of graphs can be embedded simultaneously with straight-line segments. Brass *et al.* [1], Erten and Kobourov [5], and Geyer *et al.* [8] showed that three paths, a planar graph and a path, and two trees do not admit geometric simultaneous embeddings. On the positive side, an algorithm for geometric simultaneous embedding of two caterpillars [1] is the strongest known result.

As geometric simultaneous embedding turns out to be very restrictive, it is natural to relax some of the constraints of the problem. Not insisting on straight-line edges led to positive results such as a linear-time algorithm by Erten and

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 268–279, 2007.

Table 1. Known results and our contribution on geometric simultaneous embedding (Geometric), geometric simultaneous embedding with no common edges (Disj. Edges), geometric simultaneous drawing with fixed embedding (Fixed Embedding), geometric simultaneous drawing with fixed embedding and no common edges (Disj. Edges, Fixed Embedding).

	Geometric	*Disj. Edges*	*Fixed Emb.*	*Disj. Edges, Fixed Emb.*
path + path	YES [1]	YES [1]	YES [1]	YES [1]
star + path	YES [1]	YES [1]	YES *Sec.* 4.1	YES *Sec.* 4.1
double-star + path	YES [1]	YES [1]	?	YES *Sec.* 4.1
caterpillar + path	YES [1]	YES [1]	?	?
caterpillar + caterpillar	YES [1]	YES [1]	NO *Sec.* 4.2	NO *Sec.* 4.2
3 paths	NO [1]	?	NO [1]	?
tree + path	?	?	?	?
tree + cycle	?	?	?	?
tree + caterpillar	?	?	NO *Sec.* 4.2	NO *Sec.* 4.2
outerplanar + path	?	?	NO *Sec.* 4.3	NO *Sec.* 4.3
outerplanar + caterpillar	?	?	NO *Sec.* 4.2	NO *Sec.* 4.2
outerplanar + cycle	?	?	NO *Sec.* 4.3	NO *Sec.* 4.3
tree + tree	NO [8]	?	NO [8]	NO *Sec.* 4.2
outerplanar + tree	NO [8]	?	NO [8]	NO *Sec.* 4.2
outerplanar + outerplanar	NO [1]	?	NO [1]	NO *Sec.* 4.2
planar + path	NO [5]	NO *Sec.* 3	NO [5]	NO *Sec.* 3
planar + tree	NO [5]	NO *Sec.* 3	NO [5]	NO *Sec.* 3
planar + planar	NO [5]	NO *Sec.* 3	NO [5]	NO *Sec.* 3

Kobourov for embedding any pair of planar graphs with at most three bends per edge, or any pair of trees with at most two bends per edge [5]. In such results it is allowed for an edge connecting a pair of vertices to be represented by different Jordan curves in different drawings. As this can be detrimental to the readability of the drawings, several papers considered a slightly more constrained version of this problem, namely, *simultaneous embedding with fixed edges*, in which bends are allowed, however, an edge connecting the same pair of vertices must be drawn in exactly the same way in all drawings. Di Giacomo and Liotta [4] showed that outerplanar graphs can be simultaneously embedded with fixed edges with paths or cycles using at most one bend per edge. Frati [6] showed that a planar graph and a tree can also be simultaneously embedded with fixed edges.

Studying the existing variants of simultaneous embedding led to practical embedding algorithms for some graph classes and techniques for simultaneous embedding have been used in visualizing evolving and dynamic graphs [2]. However, many problems remain theoretically open and in practice algorithms applying these ideas to evolving and dynamic graphs do not provide any guarantees on the quality of the resulting layouts. With this in mind, we consider three further variants of the geometric simultaneous embedding problem.

Most of the proofs about the non-existence of simultaneous embeddings exploit the presence of common edges between the input graphs. Hence, it is natural ask if larger classes of graphs have geometric simultaneous embeddings when no edges are shared. In Section 3 we answer in the negative for planar graph-path pairs, generalizing the result in [5], where it is shown that a planar graph and a path that share edges do not admit a geometric simultaneous embedding.

In Section 4 we consider the problem of geometric simultaneous embedding in which the embeddings for the graphs are fixed. We call this setting *geometric simultaneous embedding with fixed embeddings*. Clearly, negative results known

for geometric simultaneous embedding remain valid here. We show that some classes of graphs that have geometric simultaneous embeddings do not admit one with individually fixed embeddings. In particular, we prove such a negative result for caterpillar-caterpillar pairs. Moreover, in the fixed embedding setting we are able to solve problems that are still open for geometric simultaneous embedding. Namely, we provide an outerplanar-path pair that has no geometric simultaneous drawing with fixed embedding. All the negative results claimed are still valid if the input graphs are assumed to not share edges. On the other hand, we partially cover the known positive results for geometric simultaneous embedding, by showing that a star and a path can always be realized and that a double-star and a path can always be realized if they do not share edges.

In the quest for more practical setting where we can guarantee some properties of the layouts, in Section 5 we study a variant we call *geometric near-simultaneous embedding*. In this setting edges are straight lines and vertices representing the same entity in different graphs can be placed not exactly in the same point but just in "near" points. Assuming vertices are placed on the grid, we show that there exist pairs of n-vertex planar graphs in which vertices that represent the same entity in different graphs must be placed at distance $\Omega(n)$. We then consider graphs "similar" in their combinatorial structure, describing algorithms which guarantee that vertices representing the same entity have only constant displacement from one drawing to the next. Such algorithms can be used to guarantee limited displacement in dynamic graph drawings.

Due to space limitations, we leave out some proofs, that can be found in [7].

2 Preliminaries

We summarize basic terminology used in this paper; for more details see [3,11]. A *straight-line drawing* of a graph is a mapping of each vertex to a unique point in the plane and of each edge to a segment between the endpoints of the edge. A *planar drawing* is one in which no two edges intersect. A *planar graph* is a graph that admits a planar drawing. A *grid drawing* is one in which every vertex is placed at a point with integer coordinates in the plane. An *embedding* of a graph is a circular ordering of the edges incident on each vertex of G. An embedding of a graph specifies the faces in any drawing respecting such an embedding, even though the embedding does not determine which one is the *external face*. A graph is *triconnected* if for every pair of distinct vertices there exist three vertex-disjoint paths connecting them. A triconnected graph has an unique embedding, up to a reversal of its adjacency lists.

An *outerplanar graph* is a graph that admits a drawing in which all the vertices are incident to the same face. A *caterpillar* is a tree in which the removal of all the leaves and their incident edges yields a path. A *star* (*double-star*) is a caterpillar with only one vertex (two vertices) of degree greater than one.

A *geometric simultaneous embedding* of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with a bijective mapping γ of their vertex sets is a pair of planar

straight-line drawings Γ_1 of G_1 and Γ_2 of G_2, such that each vertex $v_2 = \gamma(v_1)$ is mapped in Γ_2 to the same point where v_1 is mapped in Γ_1, where $v_1 \in V_1$ and $v_2 \in V_2$.

3 Simultaneous Embedding without Common Edges

We consider the geometric simultaneous embedding of graphs not sharing common edges, exhibiting a planar graph and a path that cannot be drawn simultaneously. We revisit the problem of embedding simultaneously graphs not sharing edges in the conclusions (Section 6).

Let G^* be the planar graph on vertices v_1, v_2, ..., v_9 shown in Fig. 1(a). Since G^* is triconnected, it has the same faces in any planar embedding. Let F^* denote the triangular face $\Delta v_1 v_3 v_9$ and P^* be the path $(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9)$.

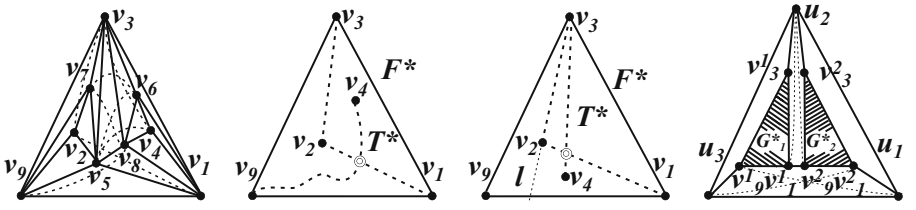


Fig. 1. (a) Planar graph G^* drawn with solid edges and path P^* drawn with dashed edges; (b)–(c) Illustrations for the proof of Lemma 1; (d) Planar graph G drawn with solid edges and path P drawn with dashed edges.

Lemma 1. *There does not exist a geometric simultaneous embedding of G^* and P^* in which the external face of G^* is F^*.*

Proof: All vertices of G^*, other than v_1, v_3 and v_9, are inside F^* as F^* is the external face of G^*. Consider the triangle T^* formed by edges (v_1, v_2), (v_2, v_3) of P^*, and by edge (v_1, v_3) of G^*. Since v_9 is incident to F^*, it must lie outside T^*. Let l be the line passing through v_2 and v_3; l separates the plane in two open half-planes, one containing v_9, called the *exterior part* of l, and one not containing v_9, called the *interior part* of l. Consider the possible placements of v_4. If v_4 is placed inside T^* then the subpath of P^* composed of edges (v_1, v_2) and (v_2, v_3) crosses the subpath of P^* connecting v_4, that lies inside T^*, and v_9, that lies outside T^*; see Fig. 1(b). Suppose v_4 is placed outside T^*. Since vertex v_4 (vertex v_2) must lie inside triangle $\Delta v_1 v_3 v_5$ (inside triangle $\Delta v_3 v_5 v_9$), the clockwise order of edges $(v_3, v_1), (v_3, v_5), (v_3, v_9)$ of G^* and edges $(v_3, v_4), (v_3, v_2)$ of P^* around v_3 must be $(v_3, v_1), (v_3, v_4), (v_3, v_5), (v_3, v_2), (v_3, v_9)$. Therefore v_4 is in the *interior part* of l and hence edge (v_1, v_2) crosses edge (v_3, v_4) in P^*; see Fig. 1(c). □

Theorem 1. *There exist a planar graph G, a path P, and a mapping between their vertices such that: (i) G and P do not share edges, and (ii) G and P have no geometric simultaneous embedding.*

Proof: We construct G and P out of two copies of G^* and P^* described above. Let G_1^* and G_2^* be two copies of G^*. Denote by v_i^j the vertex of G_j^* that corresponds to the vertex v_i in G^*, where $j = 1, 2$ and $i = 1, \ldots, 9$. Let G be the graph composed of G_1^* and G_2^* together with three additional vertices u_1, u_2, and u_3 and eight additional edges (u_1, u_2), (u_1, u_3), (u_2, u_3), (u_1, v_1^2), (u_2, v_3^1), (u_2, v_3^2), (u_3, v_9^1), and (v_1^1, v_9^2); see Fig. 1(d). Graph G is triconnected and therefore it has exactly one planar embedding and it has the same faces in any plane drawing. Let P be the path $(u_1, v_9^1, v_8^1, v_7^1, v_6^1, v_5^1, v_4^1, v_3^1, v_2^1, v_1^1, u_2, v_9^2, v_8^2, v_7^2, v_6^2, v_5^2, v_4^2, v_3^2, v_2^2, v_1^2, u_3)$. It is easy to verify that G and P do not share edges. Note that the subpaths of P induced by the vertices of G_1^* and by the vertices of G_2^* play the same role that path P^* plays for graph G^* in Lemma 1.

Let F_1^* and F_2^* denote cycles (v_1^1, v_3^1, v_9^1) and (v_1^2, v_3^2, v_9^2); these cycles are faces of G_1^* and G_2^*. We now show that every plane drawing Γ of G determines a non-planar drawing of P. Consider the embedding $\mathcal{E}_G$ of G obtained by choosing $\Delta u_1 u_2 u_3$ as external face; see Fig. 1(d). Choosing any face external to F_1^* (F_2^*) in $\mathcal{E}_G$ as external face of Γ leaves G_1^* (G_2^*) embedded with external face F_1^* (F_2^*). Hence, we can apply Lemma 1 and conclude that there does not exist a simultaneous embedding of G and P. □

4 Simultaneous Drawing with Fixed Embedding

Next, we examine the possibility of embedding graphs simultaneously with straight-line edges and with fixed embeddings.

4.1 Simultaneous Drawing of Stars, Double-Stars and Paths with Fixed Embedding

Let P be an n-vertex path and let S be an n-vertex star with *center* c and embedding $\mathcal{E}$. Let $P = (a_1, a_2, \ldots, a_l, c, b_1, b_2, \ldots, b_m)$, where one among sequences $(a_1, a_2, \ldots, a_l)$ and $(b_1, b_2, \ldots, b_m)$ could be empty. Draw S with c as leftmost point and with all edges in an order around c consistent with $\mathcal{E}$, so that edge (c, b_1), if it exists, is the *uppermost* edge of S. This can be done so that the x-coordinate of a vertex b_i is greater than the one of a vertex a_j, with $1 \leq i \leq m$ and $1 \leq j \leq l$, the x-coordinate of a vertex b_i is greater than the one of a vertex b_j, with $1 \leq j < i \leq m$, and the x-coordinate of a vertex a_i is greater than the one of a vertex a_j, with $1 \leq i < j \leq l$; see Fig. 2(a). The resulting drawing of S is clearly planar. Further, P is not self-intersecting as it is realized by two x-monotone curves joined by an edge that is higher than every other edge of P. This yields the following result:

Theorem 2. *An n-vertex star and an n-vertex path admit a geometric simultaneous embedding in which the star has a fixed prescribed embedding.*

Now let P be an n-vertex path and let D be an n-vertex double-star with *centers* c_1 and c_2 and with embedding $\mathcal{E}$. Suppose D and P do not share edges. Let $P = (a_1, a_2, \ldots, a_l, c_1, b_1, b_2, \ldots, b_m, c_2, d_1, d_2, \ldots, d_p)$. Sequences $(a_1, a_2, \ldots, a_l)$

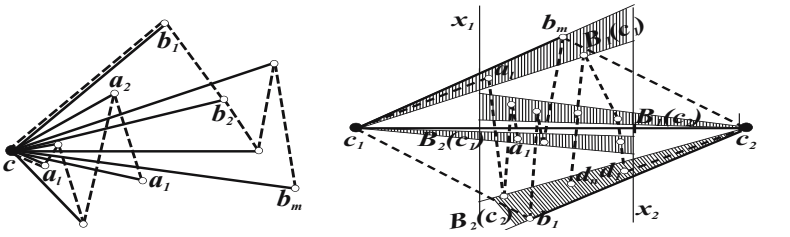


Fig. 2. (a) Simultaneous embedding of a star and a path; (b) Simultaneous embedding of a double-star and a path not sharing edges

and $(d_1, d_2, \ldots, d_p)$ could be empty, while $m \geq 2$. Further, b_1 is neighbor of c_2 and b_m is neighbor of c_1 in D; see Fig. 2(b). Group the edges incident to c_1 (incident to c_2), except for (c_1, c_2), in two *bundles* $B_1(c_1)$ and $B_2(c_1)$ (resp. $B_1(c_2)$ and $B_2(c_2)$). $B_1(c_1)$ is made up of the edges starting from (c_1, b_m) until, but not including, (c_1, c_2) in the clockwise order of the edges incident to c_1. $B_2(c_1)$ is made up of the edges starting from (c_1, c_2) until, but not including, (c_1, b_m) in the clockwise order of the edges incident to c_1. The other two bundles $B_1(c_2)$ and $B_2(c_2)$ are defined analogously. P is divided into three subpaths, $P_1 = (c_1, a_l, a_{l-1}, \ldots, a_2, a_1)$, $P_2 = (c_1, b_1, b_2, \ldots, b_m, c_2)$, and $P_3 = (c_2, d_1, d_2, \ldots, d_p)$.

Draw (c_1, c_2) as an horizontal segment, with c_1 on the left. $B_1(c_1)$ and $B_2(c_1)$ ($B_1(c_2)$ and $B_2(c_2)$) are drawn inside *wedges* centered at c_1 (at c_2) and directed rightward (leftward), with $B_1(c_1)$ above (c_1, c_2) and $B_2(c_1)$ below (c_1, c_2) (with $B_1(c_2)$ above (c_2, c_1) and $B_2(c_2)$ below (c_2, c_1)). Such wedges are disjoint and they share an interval $[x_1, x_2]$ of the x-axis, where $[x_1, x_2]$ is a sub-interval of the x-extension of the edge (c_1, c_2). Draw each edge inside the wedge of its bundle, respecting $\mathcal{E}$ and so that the following rules are observed: the x-coordinate of a vertex b_i is greater than the one of a vertex a_j, with $1 \leq i \leq m$ and $1 \leq j \leq l$; the x-coordinate of a vertex d_k is greater than the one of a vertex b_i, with $1 \leq k \leq n$ and $1 \leq i \leq m$; the vertices of P_1, of P_2, and of P_3 have increasing, increasing, and decreasing x-coordinates, respectively. Each vertex has an x-coordinate in the open interval (x_1, x_2). Edge (c_1, b_m) ((c_2, b_1)) of D is drawn so high (so low) that edge (c_2, b_m) ((c_1, b_1)) of P does not create crossings with other edges of the path. The drawing of D is planar since the edges of D are drawn inside disjoint regions of the plane. The absence of crossings in the drawing of P follows from (1) the planarity of the drawings of its subpaths, which in turn follows from the strictly increasing or decreasing x-coordinate of its vertices; and (2) from the fact that the subpaths occupy disjoint regions, except for edges (c_1, b_1) and (c_2, b_m) which do not create crossings, as already discussed. Thus, we have:

Theorem 3. *An n-vertex double-star and an n-vertex path not sharing edges admit a geometric simultaneous embedding in which the double-star has a fixed prescribed embedding.*

4.2 Simultaneous Drawing of Two Caterpillars with Fixed Embedding

Insisting on a fixed embedding when simultaneously embedding planar graphs is a very restrictive requirement as shown by the following theorem:

Theorem 4. *It is not always possible to find a geometric simultaneous embedding for two caterpillars with fixed embeddings.*

Proof: Let C_1 and C_2 be the two caterpillars with fixed embeddings $\mathcal{E}_1$ and $\mathcal{E}_2$ and a bijective mapping $\gamma(x) = x$ between their vertices; see Fig. 3(a-b). We now show that there does not exist a geometric simultaneous embedding of C_1 and C_2 in which C_1 and C_2 respect $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively.

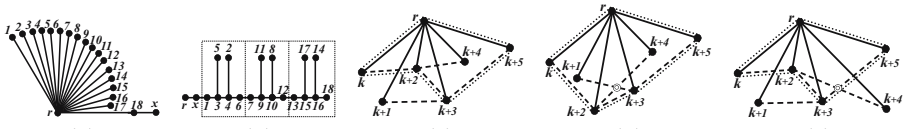


Fig. 3. (a)–(b) Caterpillars C_1 and C_2; (c)–(e) Illustrations for the proof of Theorem 4

Construct a straight-line drawing Γ_1 of C_1. The embedding $\mathcal{E}_1$ of C_1 forces the vertices $1, 2, \ldots, 18$ to appear in this order around r in Γ_1. Consider the subtrees of C_1 induced by the vertices $r, 1, 2, \ldots, 6$, by the vertices $r, 7, 8, \ldots, 12$, and by the vertices $r, 13, 14, \ldots, 18$. Since such subtrees appear consecutively around r, then at least one of them must be drawn in a wedge rooted at r and with angle less than π. Let C_S be such a subtree and let $k, k+1, \ldots, k+5$ be the vertices of C_S, with $k = 1, 7$ or 13. Without loss of generality, let r be the uppermost point of this wedge. It follows that C_S must be drawn *downward*. Denote by P the polygon composed of the edges (r, k) and $(r, k+5)$ of C_1 and of the edges $(k, k+2)$, $(k+2, k+3)$, and $(k+3, k+5)$ of C_2. Note that vertices $k+1$ and $k+4$ must be either both inside or both outside P. In fact, placing one of these vertices inside and the other outside P is not consistent with the embedding constraints of $\mathcal{E}_2$; see Fig. 3(c). If both vertices $k+1$ and $k+4$ are placed inside P, then the embedding constraints of $\mathcal{E}_1$ and $\mathcal{E}_2$ and the upwardness of C_S imply that edge $(k+2, k+4)$ must cut edge $(r, k+3)$ and that edge $(k+1, k+3)$ must cut edge $(r, k+2)$. It follows that there is an intersection between edges $(k+2, k+4)$ and $(k+1, k+3)$, both belonging to C_S; see Fig. 3(d). Similarly, if both vertices $k+1$ and $k+4$ are placed outside P, then by the embedding constraints of $\mathcal{E}_1$ and $\mathcal{E}_2$ vertex $k+2$ is placed inside the polygon formed by the edges $(r, k+1)$, $(r, k+5)$ of C_1 and by the edges $(k+1, k+3)$, $(k+3, k+5)$ of C_2. Hence, edge $(k+2, k+4)$ cuts such a polygon either in edge $(k+1, k+3)$ or in edge $(k+3, k+5)$; see Fig. 3(e) and this concludes the proof. □

4.3 Simultaneous Drawing of Outerplanar Graphs and Paths with Fixed Embedding

Let O^* be the outerplanar graph on vertices v_1, v_2, ..., v_7 shown in Fig. 4(a) and $\mathcal{E}^*$ be the embedding of O^* shown in Fig. 4(b). Let F^* be the face of $\mathcal{E}^*$ with incident vertices v_1, v_3, and v_7 and let P^* be the path $(v_1, v_2, v_3, v_4, v_5, v_6, v_7)$.

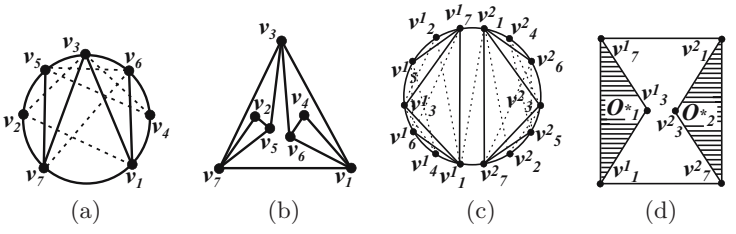


Fig. 4. (a) Outerplanar graph O^*, drawn with solid edges, and path P^*, drawn with dashed edges. (b) Embedding $\mathcal{E}^*$ of O^*. (c) Outerplanar graph O, drawn with solid edges, and path P, drawn with dashed edges. (d) Embedding $\mathcal{E}$ of O.

Lemma 2. *There does not exist a geometric simultaneous embedding of O^* and P^* in which the embedding of O^* is $\mathcal{E}^*$ and the external face of O^* is F^*.*

Theorem 5. *There exist an outerplanar graph O, an embedding $\mathcal{E}$ of O, a path P, and a mapping between their vertices such that: (i) O and P do not share edges, and (ii) O and P have no geometric simultaneous embedding.*

Proof: Let O_1^* and O_2^* be two copies of the outerplanar graph O^* defined above. Denote by v_i^j, with $j = 1, 2$ and $i = 1, \ldots, 7$, the vertex of O_j^* that corresponds to vertex v_i of O^* in O. Let $\mathcal{E}_1^*$ and $\mathcal{E}_2^*$ be the embeddings of O_1^* and O_2^* corresponding to the embedding $\mathcal{E}^*$ of O^*. Let O be the graph composed of O_1^*, of O_2^*, and of edges (v_7^1, v_1^2), (v_1^1, v_7^2); see Fig. 4(c). Let the embedding $\mathcal{E}$ for O be defined as follows: (i) each vertex of O_1^* (of O_2^*) but for v_1^1 and v_7^1 (but for v_1^2 and v_7^2) has the same adjacency list as in $\mathcal{E}_1^*$ (in $\mathcal{E}_2^*$); (ii) the adjacency lists of the remaining vertices are as follows: $v_1^1 \rightarrow (v_7^1, v_6^1, v_4^1, v_3^1, v_7^2)$, $v_7^1 \rightarrow (v_1^2, v_3^1, v_2^1, v_5^1, v_1^1)$, $v_1^2 \rightarrow (v_7^2, v_6^2, v_4^2, v_3^2, v_7^1)$, $v_7^2 \rightarrow (v_1^1, v_3^2, v_2^2, v_5^2, v_1^2)$. Let P be the path $(v_7^1, v_6^1, v_5^1, v_4^1, v_3^1, v_2^1, v_1^1, v_1^2, v_2^2, v_3^2, v_4^2, v_5^2, v_6^2, v_7^2)$. O and P do not share edges, and the subpaths of P induced by the vertices of O_1^* (O_2^*) play for O_1^* (O_2^*) the same role that path P^* plays for graph O^* in Lemma 2.

Let F_1^* and F_2^* denote cycles (v_1^1, v_3^1, v_7^1) and (v_1^2, v_3^2, v_7^2), respectively. These cycles are faces of O_1^* and O_2^*. We now show that every plane drawing $\Gamma_{\mathcal{E}}$ of O with embedding $\mathcal{E}$ determines a non-planar drawing of P. Consider the embedding $\mathcal{E}_O$ of O obtained by choosing $(v_1^1, v_7^1, v_1^2, v_7^2)$ as external face; see Fig. 4(d). Choosing any face external to F_1^* (F_2^*) in $\mathcal{E}_O$ as external face of $\Gamma_{\mathcal{E}}$ leaves O_1^* (O_2^*) embedded with external face F_1^* (F_2^*). Hence, we can apply Lemma 2 and conclude that there is no simultaneous embedding of O and P. □

5 Near-Simultaneous Embedding

In this section we study the variation of geometric simultaneous embedding in which vertices representing the same entity in different graphs can be placed in different points in different drawings. However, in order to preserve the viewer's "mental map" corresponding vertices should be placed as close as possible. This turns out to be impossible for general planar graphs, as the first lemma of this section shows. First, define the *displacement* of a vertex v between two drawings Γ_1 and Γ_2 as the distance between the location of v in Γ_1 and the location of v in Γ_2. Second, we show that there exist two n-vertex planar graphs G_1 and G_2 with a bijection γ between their vertices such that for any two planar straight-line grid drawings Γ_1 and Γ_2 of G_1 and G_2, respectively, there exists a vertex v that has a displacement $\Omega(n)$ between Γ_1 and Γ_2.

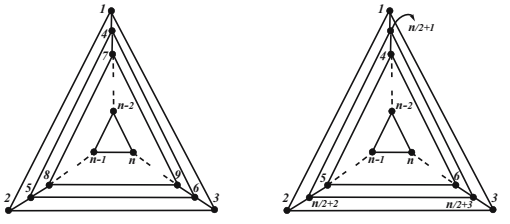


Fig. 5. (a) Nested triangle graph G_1; (b) Nested triangle graph G_2

Let G_1 and G_2 be two n-vertex *nested triangle* graphs; see Fig. 5. A nested triangle graph G is a triconnected planar graph with a triangular face $F(G)$ such that removing the vertices of $F(G)$ and their incident edges leaves a smaller nested triangle graph or an empty vertex set. Suppose the mapping $\gamma(v_1) = v_2$ between vertices $v_1 \in V(G_1)$ and vertices $v_2 \in V(G_2)$ is the one shown in Fig. 5 and defined by the following procedure: embed G_1 and G_2 with external faces $F(G_1)$ and $F(G_2)$, respectively. Starting from G_1 (G_2), for $i = 1, \ldots, n/3$, remove from the current graph the three vertices of the external face and label them $3i-2, 3i-1$, and $3i$ ($3(i+1)/2-2, 3(i+1)/2-1$, and $3(i+1)/2$ if i is odd, or $(n+3i)/2-2, (n+3i)/2-1$, and $(n+3i)/2$ if i is even). Then, for any two planar straight-line grid drawings Γ_1 of G_1 and Γ_2 of G_2 and G_2, we have:

Lemma 3. *There exists a vertex representing the same entity in G_1 and G_2 that has displacement $\Omega(n)$ between Γ_1 and Γ_2.*

The lower bound in Lemma 3 is easily matched by an upper bound obtained by independently drawing each planar graph in $O(n) \times O(n)$ area: Each vertex is displaced by at most the length of the diagonal of the drawing's bounding box. Clearly, such a diagonal has length $O(n)$.

The above result shows that we cannot hope to guarantee near-simultaneous embeddings for arbitrary pairs of planar graphs. It is possible, however, that for graphs that are "similar", near-simultaneous embeddings might exist. Similarity

between graphs could be defined and regarded in several different ways, by minding both the combinatorial structure of the graphs and the mapping between the vertices of the graphs. With this in mind, in the following we look for near-simultaneous embeddings of similar paths and similar trees.

5.1 Near-Simultaneous Drawings of Similar Paths

Recall that two paths always have a geometric simultaneous embedding, while three of them might not have one [1]. Therefore, in order to represent a sequence of paths using a sequence of planar drawings, vertices that are in correspondence under the mapping must be displaced from one drawing to the next.

Observing that a path induces an ordering of the vertices, call two n-vertex paths P_1 and P_2 with orderings π_1 and π_2 of their vertices and with a bijective mapping γ between their vertices *k-similar* if for each vertex $v_1 \in P_1$ the position of v_1 in π_1 differs by at most k positions from the one of $v_2 = \gamma(v_1)$ in π_2. Drawing the paths as horizontal polygonal lines with uniform horizontal distances between adjacent vertices gives a near-simultaneous drawing. As any vertex v_i if P_1 occurs within k positions in P_2 (compared with its position in P_1) then the extent of the displacement of the vertex from one drawing to the next is limited by exactly k units. More generally, this idea can be summarized as follows:

Theorem 6. *A sequence of n-vertex paths $P_0, P_1, ..., P_m$, where each two consecutive paths are k-similar, can be drawn so that the displacement of any vertex in a pair of paths that are consecutive in the sequence is at most k.*

5.2 Near-Simultaneous Drawings of Similar Trees

Generalizing the idea of k-similarity to trees, call two rooted arbitrarily ordered trees T_1 and T_2 with vertex sets V_1 and V_2 and with bijective mapping γ between their vertices, k-similar if: (i) The depths of any vertex $v_1 \in V_1$ and of its corresponding vertex $\gamma(v_1) \in V_2$ differ by at most k; (ii) The positions of any two corresponding vertices in any pre-established traversal of the tree among pre-, in-, post-order, or breadth-first-search traversal differ by at most k.

Given two trees T_1 and T_2 that are k-similar with respect to a pre-established traversal order π, we can draw each of T_1 and T_2 as follows: (1) Assign to each vertex v_i its position $\pi(v_i)$ as an x-coordinate; (2) Assign to each vertex v_i its depth as a y-coordinate.

Such an algorithm produces layouts that are planar and *layered*. A drawing is layered if (i) each vertex is assigned to a *layer*, (ii) for each layer an order of its vertices is specified, and (iii) there are only edges joining vertices on consecutive layers. Since subsequent trees are k-similar, the depth of any vertex and its position in a tree traversal changes only by k in two consecutive trees; hence, we have that the displacement of a vertex representing the same entity in different drawings is at most $\sqrt{k^2 + k^2} = k\sqrt{2}$. More generally, we have the following:

Theorem 7. *A sequence of n-vertex trees $T_0, T_1, ..., T_m$, where each two consecutive trees are k-similar, can be drawn such that the displacement of any vertex in a pair of trees that are consecutive in the sequence is at most $k\sqrt{2}$.*

Observe that an analogous definition of similarity between two graphs and the same layout algorithm work more generally for *level planar graphs* [9,10] (and hence for *outerplanar graphs*). Finally, the area requirement of the drawings produced by the described algorithm is worst-case quadratic in the number of vertices of a tree (or of a level planar graph).

6 Conclusions

In this paper we have considered some variations of the well-known problem of embedding graphs simultaneously.

Concerning the geometric simultaneous embedding without common edges, we provided a negative result that seems to show that the geometric simultaneous embedding is not more powerful by assuming the edge sets of the input graphs to be disjoint. Further, we believe that there exist two trees not sharing common edges that do not admit a geometric simultaneous embedding. This would extend the result in [8] where two trees that do not admit a simultaneous embedding and that do share edges are shown. Consider two isomorphic rooted trees $T_1(h,k)$ and $T_2(h,k)$ a mapping γ between their vertices defined as follows (see Fig. 6): (i) the root of $T_1(h,k)$ (of $T_2(h,k)$) has k children; (ii) each vertex of $T_1(h,k)$ (of $T_2(h,k)$) at distance i from the root, with $1 \leq i < h$, has a number of children one less than the number of vertices at distance i from the root in $T_1(h,k)$ (in $T_2(h,k)$); (iii) one vertex of $T_1(h,k)$ (of $T_2(h,k)$) at distance h from the root has one child; (iv) each child of the root of $T_1(h,k)$ is mapped to a distinct child of the root of $T_2(h,k)$; (v) for each pair of vertices v_1 of $T_1(h,k)$ and v_2 of $T_2(h,k)$ that are at distance i from the root of their own tree and that are such that $v_2 \neq \gamma(v_1)$, there exists a child of v_1 that is mapped to a child of v_2; (vi) the only vertex of $T_1(h,k)$ (of $T_2(h,k)$) that is at distance $h+1$ from the root is mapped to the root of $T_2(h,k)$ (to the root of $T_1(h,k)$).

Conjecture 1. For sufficiently large h and k, $T_1(h,k)$ and $T_2(h,k)$ do not admit a geometric simultaneous embedding with mapping γ between their vertices.

For the problem of drawing graphs simultaneously with fixed embedding, we provided more negative results than in the usual setting for geometric simultaneous

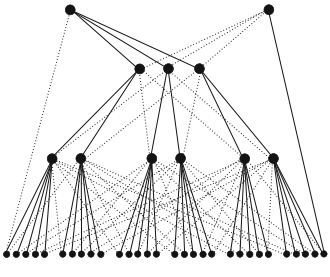


Fig. 6. Trees $T_1(3,3)$ and $T_2(3,3)$ with the mapping γ between their vertices. $T_1(3,3)$ has solid edges and $T_2(3,3)$ has dashed edges.

embedding, while providing only two positive results partially covering the ones already known for geometric simultaneous embedding. We believe that understanding the possibility of obtaining a simultaneous embedding of a tree and a path in which the tree has a fixed embedding could be useful for the same problem in the non-fixed embedding setting.

Even in the more relaxed near-simultaneous setting, we have shown that without assuming a similarity in the sequence of graphs to be drawn, it is difficult to limit the displacement of a vertex from a drawing to the next. We have shown that for paths, for trees, and for level planar graphs there exist reasonable similarity measures that allow us to obtain near-simultaneous drawings. However, in the case of general planar graphs it is not yet clear what kind of similarity metric can be defined and how well can such graphs be drawn.

Acknowledgments

Thanks to Markus Geyer for useful discussions.

References

1. Braß, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous planar graph embeddings. Comput. Geom. 36(2), 117–130 (2007)
2. Collberg, C., Kobourov, S.G., Nagra, J., Pitts, J., Wampler, K.: A system for graph-based visualization of the evolution of software. In: SoftVis, pp. 377–386 (2003)
3. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing. Prentice Hall, Upper Saddle River, NJ (1999)
4. Di Giacomo, E., Liotta, G.: A note on simultaneous embedding of planar graphs. In: EWCG, pp. 207–210 (2005)
5. Erten, C., Kobourov, S.G.: Simultaneous embedding of planar graphs with few bends. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 195–205. Springer, Heidelberg (2005)
6. Frati, F.: Embedding graphs simultaneously with fixed edges. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 108–113. Springer, Heidelberg (2007)
7. Frati, F., Kaufmann, M., Kobourov, S.: Constrained simultaneous and near-simultaneous embeddings. Tech. Report RT-DIA-120-2007, University of Roma Tre (2007), http://dipartimento.dia.uniroma3.it/ricerca/rapporti/rt/2007-120.pdf
8. Geyer, M., Kaufmann, M., Vrto, I.: Two trees which are self-intersecting when drawn simultaneously. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 201–210. Springer, Heidelberg (2006)
9. Healy, P., Kuusik, A., Leipert, S.: A characterization of level planar graphs. Discr. Math. 280(1-3), 51–63 (2004)
10. Jünger, M., Leipert, S.: Level planar embedding in linear time. Jour. of Graph Alg. and Appl. 6(1), 67–113 (2002)
11. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)

Simultaneous Geometric Graph Embeddings*

Alejandro Estrella-Balderrama[1], Elisabeth Gassner[3], Michael Jünger[4], Merijam Percan[4], Marcus Schaefer[2], and Michael Schulz[4]

[1] Department of Computer Science, University of Arizona
aestrell@cs.arizona.edu
[2] School of CTI, DePaul University
mschaefer@cs.depaul.edu
[3] Institut für Mathematik B, Technische Universität Graz
gassner@opt.math.tu-graz.ac.at
[4] Institut für Informatik, Universität zu Köln
{mjuenger,percan,schulz}@informatik.uni-koeln.de

Abstract. We consider the following problem known as simultaneous geometric graph embedding (SGE). Given a set of planar graphs on a shared vertex set, decide whether the vertices can be placed in the plane in such a way that for each graph the straight-line drawing is planar. We partially settle an open problem of Erten and Kobourov [5] by showing that even for two graphs the problem is **NP**-hard.

We also show that the problem of computing the rectilinear crossing number of a graph can be reduced to a simultaneous geometric graph embedding problem; this implies that placing SGE in **NP** will be hard, since the corresponding question for rectilinear crossing number is a long-standing open problem. However, rather like rectilinear crossing number, SGE can be decided in **PSPACE**.

1 Introduction

Simultaneous drawing deals with the problem of drawing two or more graphs at the same time such that all drawings satisfy specific requirements. When two planar graphs are given, the natural question arises whether a combined drawing leads to two planar drawings [2,5,6,8,9,10]. This problem has been studied in different variations. While most work has been spent on deciding whether different kinds of graphs allow such drawings, this paper focuses on the complexity question. We study the *geometric* version which restricts the problem to straight-line drawings.

Problem:	*Simultaneous Geometric Embedding Problem (SGE)*
Instance:	A set of planar graphs $G_i = (V, E_i)$ on the same vertex set V.
Question:	Are there plane straight-line drawings D_i of G_i such that each vertex is mapped to the same point in the plane in all such D_i?

* Partially supported by Marie-Curie Research Training Network (ADONET) and by the German Science Foundation (JU204/10-1).

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 280–290, 2007.

The complexity of the SGE problem for two graphs is mentioned as an open problem in [5]. We settle part of the problem by showing that it is **NP**-hard. It remains open whether the problem lies in **NP**, but we show by a comparison to the rectilinear crossing number and the existential theory of the real numbers that settling the complexity of SGE will be hard, since determing the complexity of calculating the rectilinear crossing number is a long-standing open problem. Our result is related to an earlier paper, in which we showed that deciding the simultaneous embeddability with fixed edges is **NP**-complete for *three* graphs (Gassner et al. [8]).

It is easy to see that SGE is non-trivial; that is, there are two planar graphs without a simultaneous geometric embedding. More surprisingly, there are even two trees that cannot be simultaneously embedded geometrically [9].

2 NP-Hardness Proof

Theorem 1. *Deciding whether two graphs have a simultaneous geometric embedding is* **NP***-hard.*

Proof. We show that there exists a polynomial transformation from 3SAT, which is well-known to be **NP**-complete, to SGE for two planar graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$.

Problem:	*3-Satisfiability Problem (3SAT)*
Instance:	A CNF-system with a set U of boolean variables and a set C of clauses over U such that each clause in C has exactly three literals.
Question:	Is there a satisfying truth assignment for U?

Given an instance of 3SAT, we construct an instance (G_1, G_2) of SGE. Then we prove that the instance of 3SAT is satisfiable if and only if there exists a simultaneous geometric embedding of (G_1, G_2).

Construction: Let $U = \{u_1, u_2, \ldots, u_n\}$ be the variable set and $C = \{c_1, c_2, \ldots, c_m\}$ be the clause set where $c_j = (l_1^j \vee l_2^j \vee l_3^j)$ for literals $l_i^j = u_h$ or $l_i^j = \bar{u}_h$ for some variable u_h ($j \in \{1, 2, \ldots, m\}$, $h \in \{1, 2, \ldots, n\}$, $i \in \{1, 2, 3\}$). The 3SAT formula f can then be written $f = c_1 \wedge c_2 \wedge \ldots \wedge c_m$.

For our construction we assume an ordering of the clauses, say $(c_1, c_2, \ldots, c_m)$. Furthermore we choose an order of the three literals in each clause c_j and hence get an order of all literals in the following way $(l_1^1, l_2^1, l_3^1, l_1^2, \ldots, l_3^m)$.

For each clause c_j we define a *clause box* by introducing vertices $r_1^j, \ldots, r_7^j$, $y^{1,j}$, $y^{2,j}$, $y^{3,j}$. These vertices are connected by edges of E_1 (solid) and E_2 (dashed) such as shown in Figure 1.

Next, we introduce two global vertices R_1 and R_2. We add an edge (R_1, R_2) to both graphs G_1 and G_2. Furthermore, R_1 is connected to the clause box of each clause c_j by edges (R_1, r_i^j) in $E_1 \cap E_2$ with $i = 2, \ldots, 6$. We also connect R_2 to the clause box by edges (R_2, r_1^j) and (R_2, r_7^j) in E_1.

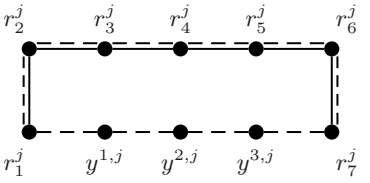


Fig. 1. The clause box of clause c_j. Edges of G_1 are solid and edges of G_2 are dashed.

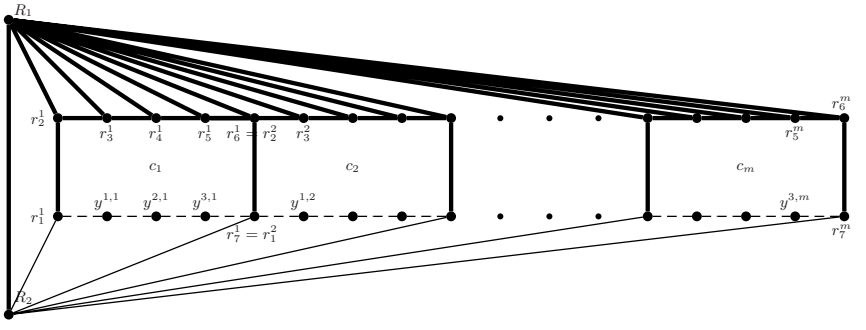


Fig. 2. The figure shows all vertices and all edges constructed so far. Edges which belong to both E_1 and E_2 are drawn bold and solid, edges of $E_1 \backslash E_2$ are thin and solid while edges of $E_2 \backslash E_1$ are dashed.

To make the construction more rigid we glue together neighboring clause boxes. This is done by identifying r_2^{j+1} with r_6^j and r_1^{j+1} with r_7^j for $j = 1, 2, \ldots, m-1$.

Figure 2 gives an idea of the construction so far. Notice that the graph given by the edges in E_1 is a subdivision of a triconnected graph which will be used later in the proof. Its planar embedding is unique up to a homomorphism of the plane.

For every literal l_i^j with $i = 1, 2, 3$, $j = 1, 2, \ldots, m$, we define a *literal gadget* that consists of thirteen vertices and eighteen edges in E_1 and fifteen edges in E_2 as shown in Figure 3. Notice that the edges in E_1 of each literal gadget are a subdivision of a triconnected graph. The only two possible embeddings are shown in Figure 3.

From now on in all figures the edges in E_1 are represented by solid lines while the edges in E_2 are drawn dashed.

Furthermore, we define edge sets that link all literal gadgets that belong to the same variable u_h. Let $l_{i_1}^{j_1}$, $l_{i_2}^{j_2}$, ..., $l_{i_{\omega_h}}^{j_{\omega_h}}$ be the set of all literals that belong to variable u_h, that is either $l_{i_\alpha}^{j_\alpha} = u_h$ or $l_{i_\alpha}^{j_\alpha} = \bar{u}_h$. Assume that these literals are given in the order defined above. Then we will link the gadgets of each pair of literals neighbored in this ordered list by edges in E_2 in the following way:

Let $l_{i_k}^{j_k}$ and $l_{i_{k+1}}^{j_{k+1}}$ with $k \in \{1, 2, \ldots, m-1\}$ be two literals neighbored in the ordered list. We add three edges in E_2. Their endpoints depend on the fact whether

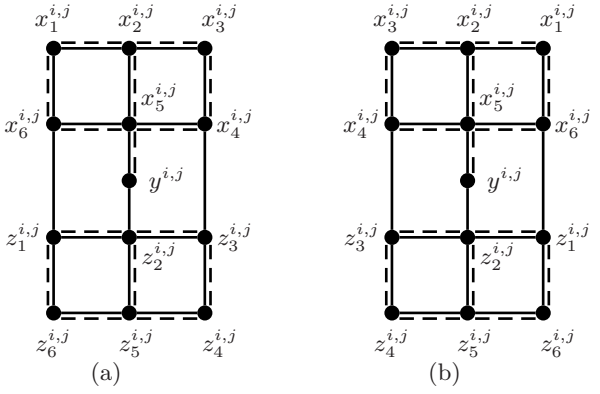


Fig. 3. Literal gadget for l_i^j with corresponding variable u_h. The edges in E_1 are solid and those in E_2 are dashed. The two different drawings (a) and (b) will become important later.

the two literals are negated or unnegated. If both literals are negated or both are unnegated, then we add the three edges $(z_1^{i_k,j_k}, z_6^{i_{k+1},j_{k+1}})$, $(z_2^{i_k,j_k}, z_5^{i_{k+1},j_{k+1}})$, $(z_3^{i_k,j_k}, z_4^{i_{k+1},j_{k+1}})$. If one of the literals is negated and one is unnegated, we add the three edges $(z_1^{i_k,j_k}, z_4^{i_{k+1},j_{k+1}})$, $(z_2^{i_k,j_k}, z_5^{i_{k+1},j_{k+1}})$, $(z_3^{i_k,j_k}, z_6^{i_{k+1},j_{k+1}})$ to graph G_2. For an example with three literals ($\omega_h = 3$) the linking edges are visualized in Figure 4.

For each clause we define a *clause gadget* consisting of three literal gadgets, the clause box and some additional vertices and edges. Let c_j be a clause with literals l_1^j, l_2^j and l_3^j. Notice that the three literal gadgets are already connected to the clause box using the vertices $y^{i,j}$ with $i = 1, 2, 3$. Further connections are given by the additional edges $(r_3^j, x_2^{1,j})$, $(r_4^j, x_2^{2,j})$ and $(r_5^j, x_2^{3,j})$ in E_2. We also add two vertices s^j, t^j and connect them to the literal gadgets via the new edges $(x_3^{1,j}, s^j) \in E_2$, $(s^j, x_1^{2,j})$, $(x_3^{2,j}, t^j) \in E_1$ and $(t^j, x_1^{3,j}) \in E_2$. A possible simultaneous embedding of a clause gadget is shown in Figure 5.

In order to connect the clause gadget to the global vertex R_2 we add vertices w^j, $w^{1,j}$, $w^{2,j}$ and $w^{3,j}$ and connect them to vertices R_2, $z_5^{1,j}$, $z_5^{2,j}$ and $z_5^{3,j}$ and to each other as shown in Figure 5.

This completes the construction.

1. Assume that the 3SAT-instance is satisfiable. Thus we can fix a true/false-assignment of the variables that satisfies the given formula and we construct an instance of SGE as explained above. We prove that there exists a simultaneous geometric embedding of the constructed instance. We say that a variable u makes a clause c `true` if either u is a literal in c and $u =$ `true` or if $\bar{u}$ is a literal in c and $u =$ `false`. Since the instance of 3SAT is satisfiable there exists at least one variable u in each clause c that makes c `true`. If variable u makes its clause `true` we draw the corresponding literal gadget as shown in Figure 3 (a). Otherwise we draw the gadget as shown in Figure 3 (b). The clause gadgets are drawn side by side in their specific ordering with the global vertices R_1 and R_2 being

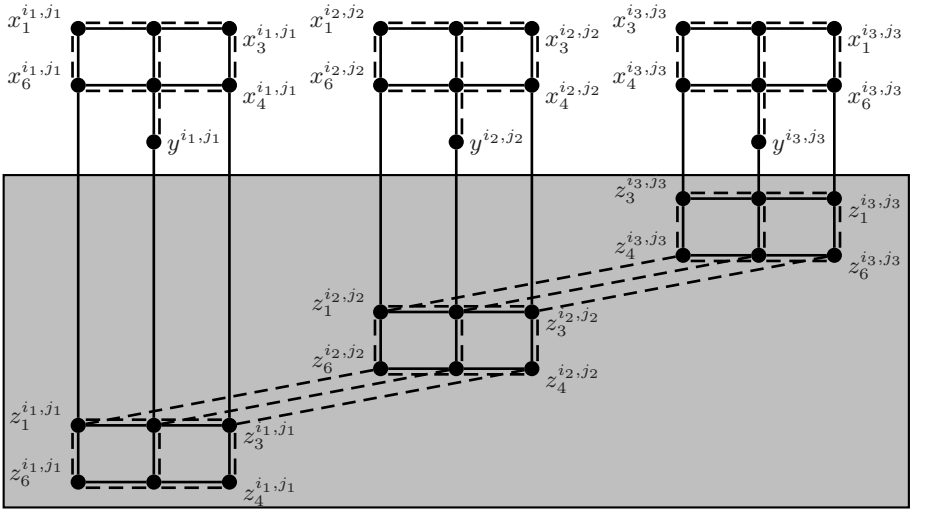


Fig. 4. All literal gadgets that belong to the same variable u_h are linked with edges in E_2. Here, the first two gadgets belong to an unnegated literal u_h whereas the third belongs to a negated literal $\bar{u}_h$.

positioned at the outer face as shown in Figure 2. Furthermore, the x-vertices of each literal gadget lie inside the clause box of its corresponding clause and the z-vertices lie outside. Moreover, every variable u gets its own horizontal region for the z-vertices to avoid crossings of linking edges of different variables. In Figure 4 the horizontal level is marked gray. Linking edges belonging to a different variable are either positioned above or below this region.

Consider now different literal gadgets corresponding to one variable u. Either all the unnegated or all the negated literals (if there exist such literals) make their clauses true but not both. But that is sufficient for the linking edges to be drawn without crossings (not counting crossings between an edge of G_1 and an edge of G_2) as shown in Figure 4.

It remains to show that we can draw the edges inside the clause gadgets without crossings of edges of the same graph.

Consider clause c_j with literals l_1^j, l_2^j and l_3^j and corresponding variables u_l, u_m, u_r. If u_l makes c_j `true`, there exists a simultaneous geometric embedding. See Figure 6 for the case where u_l is the only variable that makes c_j `true`. Simple modifications yield a simultaneous embedding for the case where u_l is not the only variable that makes c_j `true`. Due to symmetry an analogous drawing can be found for the case where u_r makes c_j `true`.

Finally, if u_m makes c_j `true`, we can find a simultaneous embedding as shown in Figure 5. Hence, we have found a simultaneous geometric embedding of the constructed instance.

2. Now assume that we are given a 3SAT-formula and the constructed SGE instance allows a simultaneous geometric embedding. We show that we can find a satisfying truth assignment for the 3SAT-instance.

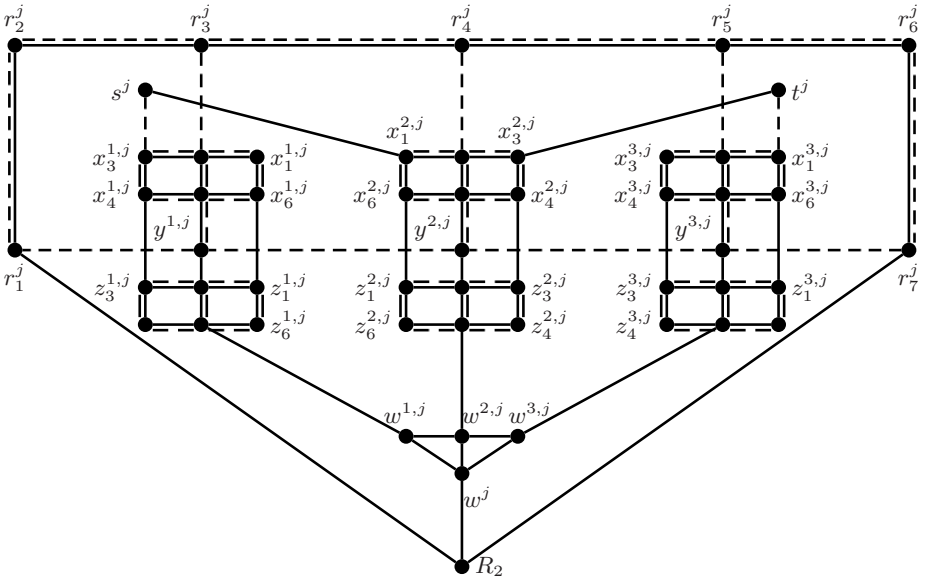


Fig. 5. Clause gadget for clause c_j plus global vertex R_2

Notice that the subgraph of G_1 shown in Figure 2 is a triconnected subdivision. Consequently, it has a unique combinatorial embedding up to homomorphisms of the plane. We choose the planar embedding with the edge (R_1, R_2) on the boundary of the outer face such that the cycle $(R_1, r_2^1, r_1^1, R_2, r_7^m, r_6^m)$ has the same order as visualized in Figure 2.

Observe that each literal gadget in the construction has one of exactly two possible planar embeddings shown in Figure 3. Let $l_{i_1}^{j_1}$, $l_{i_2}^{j_2}$, ..., $l_{i_{\omega_h}}^{j_{\omega_h}}$ be the set of all literals that belong to variable u_h. Then due to the edges in E_2 shown in Figure 4 all unnegated literals of u_h have the same embedding and all negated literals have just the opposite embedding. We assign the value `true` to variable u_h if the ordering for unnegated literals is the same as in Figure 3 (a) and `false` otherwise.

For each literal l_i^j in each clause c_j the vertex $y^{i,j}$ lies on the boundary of the clause box. The edge $(r_3^j, x_2^{1,j})$ is not allowed to cross any of the edges incident to global vertex R_1 (which is positioned outside the clause box). Hence $x_2^{1,j}$ and thus all vertices $x_i^{1,j}$, with $i = 1, \ldots, 6$, have to lie within the clause box. With similar arguments the x-vertices of l_2^j and l_3^j lie within the clause box. But now the vertices s^j and t^j must lie within the clause box which is surrounded by edges in E_2.

As soon as a literal gadget l_i^j is connected to a literal gadget of the same variable (see Figure 4) the vertices $z_k^{i,j}$, with $k = 1, \ldots, 6$, lie outside the corresponding clause box. This is particularly the case for all literal gadgets that belong to a clause which is not `true`.

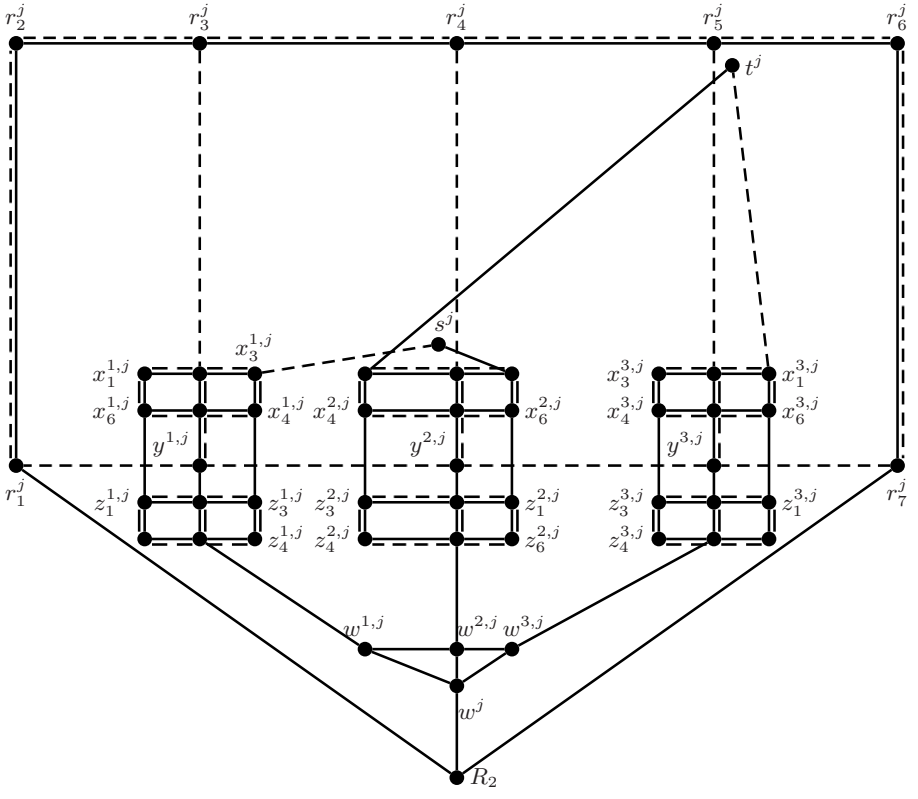


Fig. 6. SGE of the clause gadget when u_l is the only variable that makes c_j `true`

Assume that there exists a clause c_j that is not `true`. Since no variable makes c_j `true` all gadgets are of the form in Figure 3 (b). This case is shown in Figure 7.

Notice that in Figure 7 vertex s_j must be placed in the light gray area as vertex $x_3^{1,j}$ lies in this area. Otherwise the edge $(x_3^{1,j}, s_j) \in E_2$ crosses one edge of the cycle that surrounds the gray area, which is a contradiction. With similar arguments t_j lies inside the dark gray area on the right of this figure. Hence the edge pair $(r_5^j, x_2^{3,j})$ and $(s_j, x_1^{2,j})$ or the edge pair $(r_3^j, x_2^{1,j})$ and $(x_3^{2,j}, t_j)$ must cross twice in order to avoid a crossing of two edges of the same graph. But this is not possible in a straight-line drawing and leads to a contradiction to the assumption that clause c_j is not `true`. Thus all clauses are `true` and hence we have found a satisfying truth assignment. □

3 Simultaneous Straight-Line Drawings and the Rectilinear Crossing Number

In this section we discuss the relationship between simultaneous geometric embeddings and two famous problems, the rectilinear crossing number and

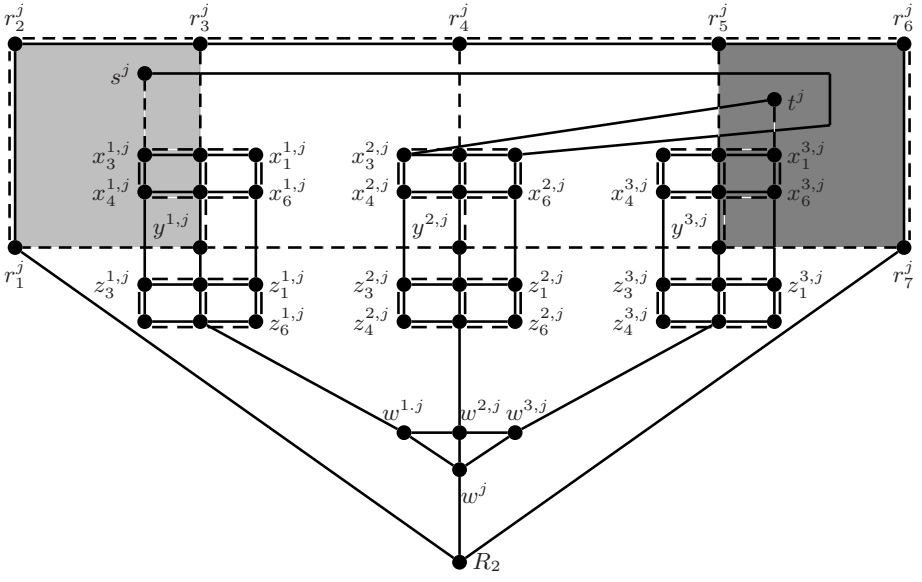


Fig. 7. If a clause is `false` then there exist two edges in the corresponding clause gadget that cross twice

existential theory of the reals. We show, that the complexity of SGE can be placed in between these two problems.

Problem:	*Rectilinear Crossing Number Problem (RCR)*
Instance:	A graph G.
Question:	What is the minimum number of crossings in a straight-line drawing of G?

RCR is well-known to be **NP**-hard [7,1]. We will show that RCR reduces to SGE via **NP**-many-one reductions, which are many-one reductions computed by an **NP**-machine rather than a polynomial time machine:

Theorem 2. *RCR* **NP***-many-one reduces to SGE for an unbounded number of graphs.*

Proof. Let $G = (V, E)$ be a graph. Guess k pairs of edges that are the potential crossing pairs and let M be the set of these edge pairs.

We define graphs $G_{e,f} = (V, E_{e,f})$ with $E_{e,f} = \{e, f\}$ for each pair of edges e and f which is not in M. If there exist an edge d which is not part of any of the new graphs $G_{e,f}$ we define a graph $G_d = (V, E_d)$ with $E_d = \{d\}$.

Notice, that each edge (and each vertex) has been added to at least one graph $G_{e,f}$ or G_d. Furthermore, if one of the graphs $G_{e,f}$ contains two edges e and f they are not allowed to cross in a straight-line drawing as this pair is not one of the k guessed pairs. Thus the decision problem whether G can be drawn straight-line with only edge-crossings in M is equivalent to the problem of finding a simultaneous geometric embedding of the graphs $G_{e,f}$ and G_d. □

Since **NP** is closed under **NP**-many-one reductions, placing SGE in **NP** has immediate consequence for RCR:

Corollary 1. *If SGE lies in* **NP** *then RCR lies in* **NP**.

Since placing RCR in **NP** is a long-standing open problem, we should not expect any easy resolution of the complexity of SGE [3, pg. 389].

Next, we will show that SGE can be expressed in the language of the existential theory of the reals. More, formally, SGE reduces to $\mathbb{R}_\exists$, the set of existential first-order sentences true over the real numbers.

Problem: *Existential Theory of the Real Numbers ($\mathbb{R}_\exists$)*
Instance: An expression of the form

$$(\exists x_1 \in \mathbb{R}) \ldots (\exists x_n \in \mathbb{R})\ P(x_1, \ldots, x_n)$$

where P is a quantifier-free Boolean formula with atomic predicates of the form $g(x_1, \ldots, x_n) \Delta 0$ where g is a real polynomial and $\Delta \in \{>, =\}$. Atomic predicates can be combined using $\vee$, $\wedge$ and $\neg$.
Question: Is the given formula true?

Theorem 3. *There exists a polynomial transformation from SGE to* $\mathbb{R}_\exists$.

Proof. Let $G_1 = (V, E_1), \ldots, G_k = (V, E_k)$ be an instance of SGE. Edge pairs $\{e, f\}$ belonging to the same graph G_i are not allowed to cross; we call such a pair a *forbidden* pair. We define the graph $G = (V, E)$ by $E := \bigcup_{i=1,\ldots,k} E_i$.

We construct an instance of $\mathbb{R}_\exists$ in the following way. For each vertex $v \in V$ we let two variables $x_v, y_v \in \mathbb{R}$ represent the coordinates of the vertex in the final drawing (which leads to the embedding that we are looking for). An edge $(u, v) \in E$ is then represented by the set of points $(x_u + t(x_v - x_u), y_u + t(y_v - y_u))$ where $t \in [0, 1]$.

We need to write constraints ensuring that the resulting drawing of G is good. In particular, we have to guarantee that no two vertices coincide, that no edge contains a vertex other than its endpoints, and that no two forbidden edges intersect.

The constraints are all of the same form: two geometric objects are apart from each other; we express this by requiring there to be a line separating them. For example, for an edge e between points $u = (x_u, y_u)$ and $w = (x_w, y_w)$ and a vertex v at (x_v, y_v) we can use the formula $A(v, e)$:

$$\begin{array}{ll} (& y_u > a_{v,e} x_u + b_{v,e} \quad \wedge \\ & y_w > a_{v,e} x_w + b_{v,e} \quad \wedge \\ & y_v < a_{v,e} x_v + b_{v,e} \ ) \quad \vee \\ (& y_u < a_{v,e} x_u + b_{v,e} \quad \wedge \\ & y_w < a_{v,e} x_w + b_{v,e} \quad \wedge \\ & y_v > a_{v,e} x_v + b_{v,e} \ ). \end{array}$$

Then $A(v, e)$ is true if and only if v and e lie on opposite sides of the line $y = a_{v,e}x + b_{v,e}$, that is, if v does not lie on e. Similarly, we can write formulas $B(e, f)$ that express that e and f do not intersect and $C(u, v)$ expressing that u and v are distinct.

Define

$$A := \bigwedge_{v \in V, e \in E, v \notin e} (\exists a_{v,e}, b_{v,e} \in \mathbb{R})\ A_{v,e},$$

$$B := \bigwedge_{(e,f) \in X} (\exists a_{e,f}, b_{e,f} \in \mathbb{R})\ B_{e,f},$$

$$C := \bigwedge_{u,v \in V} (\exists a_{u,v}, b_{u,v} \in \mathbb{R})\ C_{u,v},$$

where we let X be the set of forbidden edge pairs.

Let $V = \{v_1, \ldots, v_n\}$ and let (x_i, y_i) be the coordinates of vertex v_i for $i = 1, \ldots, n$, then

$$(\exists x_1, y_1, \ldots, x_n, y_n \in \mathbb{R})\ A \wedge B \wedge C$$

expresses that there exists a good straight-line drawing of G in which no forbidden pair of edges crosses. The drawing of G gives rise to a set of drawings for each graph G_i (by deleting all other edges) and thus to a simultaneous geometric embedding. As the forbidden edge pairs do not cross, each graph G_i has a planar drawing.

Finally, note that the formula can easily be brought into the normal form required for $\mathbb{R}_\exists$. □

Since it is known that $\mathbb{R}_\exists$ can be decided in **PSPACE** [4], we can draw the following conclusion about the complexity of SGE:

Corollary 2. *SGE, for an arbitrary number of graphs, is* **NP***-hard and lies in* **PSPACE**.

References

1. Bienstock, D.: Some provably hard crossing number problems. Discrete Comput. Geom. 6(5), 443–459 (1991)
2. Brass, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous planar graph embeddings. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 243–255. Springer, Heidelberg (2003)
3. Brass, P., Moser, W., Pach, J.: Research problems in discrete geometry. Springer, New York (2005)
4. Canny, J.: Some algebraic and geometric computations in pspace. In: STOC 1988. Proceedings of the twentieth annual ACM symposium on Theory of computing, pp. 460–469 (1988)
5. Erten, C., Kobourov, S.G.: Simultaneous embedding of planar graphs with few bends. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 195–205. Springer, Heidelberg (2005)

6. Frati, F.: Embedding graphs simultaneously with fixed edges. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 108–113. Springer, Heidelberg (2007)
7. Garey, M.R., Johnson, D.S.: Crossing number is NP-complete. SIAM Journal on Algebraic and Discrete Methods 4(3), 312–316 (1983)
8. Gassner, E., Jünger, M., Percan, M., Schaefer, M., Schulz, M.: Simultaneous graph embeddings with fixed edges. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 325–335. Springer, Heidelberg (2006)
9. Geyer, M., Kaufmann, M., Vrt'o, I.: Two trees which are self-intersecting when drawn simultaneously. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 201–210. Springer, Heidelberg (2006)
10. Di Giacomo, E., Liotta, G.: A note on simultaneous embedding of planar graphs. In: 21st European Workshop on Comp.Geometry, pp. 207–210 (2005)

Efficient C-Planarity Testing for Embedded Flat Clustered Graphs with Small Faces*

Giuseppe Di Battista and Fabrizio Frati

Dipartimento di Informatica e Automazione – Università di Roma Tre

Abstract. Let C be a clustered graph and suppose that the planar embedding of its underlying graph is fixed. Is testing the c-planarity of C easier than in the variable embedding setting? In this paper we give a first contribution towards answering the above question. Namely, we characterize c-planar embedded flat clustered graphs with at most five vertices per face and give an efficient testing algorithm for such graphs. The results are based on a more general methodology that shades new light on the c-planarity testing problem.

1 Introduction

Determining the computational complexity of the c-planarity testing for clustered graphs is one of the main Graph Drawing challenges. However, despite all the research efforts spent, only for restricted families of clustered graphs polynomial-time testing algorithms have been found, and the general problem is open.

A brief survey on the problem of testing the c-planarity of clustered graphs can be found in [2]. The classes of clustered graphs for which the problem is known to be polynomial-time solvable are the following. *c-Connected clustered graphs*, in which each cluster induces a connected subgraph of the underlying graph; the first algorithm for this class has been presented in [7]. *Completely connected clustered graphs*, that are c-connected clustered graphs such that the complement of the subgraph induced by each cluster is connected; an elegant characterization for this class is shown in [1]. *Almost connected clustered graphs*, in which either all nodes corresponding to non-connected clusters are in the same path in the cluster hierarchy, or for each non-connected cluster its parent and all its siblings are connected [9]. *Extrovert clustered graphs*, a generalization of c-connected clustered graphs with special restrictions on the cluster hierarchy [8]. *Cycles of clusters*, in which the hierarchy is *flat*, the underlying graph is a simple cycle, and the clusters are arranged in a cycle [3]. The clustering hierarchy is *flat* if all clusters, but for the root, are at the same level. *Clustered cycles*, that are clustered graphs in which the hierarchy is flat, the underlying graph is a simple cycle, and the clusters are arranged into an embedded plane graph [4].

Let C be a clustered graph. Suppose that a planar embedding of its underlying graph is fixed. Is testing the c-planarity of C easier than in the variable embedding setting? This question is motivated by the existence of many NP-hard Graph Drawing problems

* Work partially supported by MUR under Project MAINSTREAM Algorithms for Massive Information Structures and Data Streams.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 291–302, 2007.

on planar graphs that become polynomial-time solvable if the embedding is fixed. Testing if a graph admits an orthogonal planar drawing with at most k bends or if a graph admits an upward planar drawing are examples of such problems.

In this paper we give a first contribution towards answering the above question. Namely, we characterize c-planar embedded flat clustered graphs with at most five vertices per face and give an efficient testing algorithm for such graphs. Our approach is to look for an augmentation of the embedded underlying graph with extra edges such that the resulting graph is c-connected and c-planar. We call *candidate saturating edges* those edges that are candidates for the augmentation. Two of such edges have a *conflict* if using both of them in the augmentation causes a crossing. We present a characterization and an efficient c-planarity testing algorithm for *single-conflict embedded flat clustered graphs*, that are embedded clustered graphs such that (i) the cluster hierarchy is flat and (ii) each candidate saturating edge has a conflict with at most one other candidate saturating edge. Characterization and algorithm for clustered graphs with at most five vertices per face are a consequence of such a more general result.

The paper is organized as follows: In Section 2 we give preliminaries. In Section 3 we characterize c-planar single-conflict embedded flat clustered graphs and c-planar embedded flat clustered graphs with at most five vertices per face. In Section 4 we present a linear time and space c-planarity test. Section 5 contains conclusions and open problems. Because of space limits some proofs are in the full version of the paper [6].

2 Preliminaries

A graph G is *vertex (edge) k-connected* if the removal of any $k-1$ vertices (edges) leaves G connected. A *separating edge* is an edge whose removal disconnects G.

A *drawing* of a graph is a mapping of each vertex to a distinct point of the plane and of each edge to a Jordan curve between the endpoints of the edge. A *planar drawing* is such that no two edges intersect except, possibly, at common endpoints. A planar drawing of a graph determines a circular ordering of the edges incident to each vertex. Two drawings of the same graph are *equivalent* if they determine the same circular orderings around each vertex. A *planar embedding* is an equivalence class of planar drawings. A planar drawing partitions the plane into topologically connected regions, called *faces*. The unbounded face is the *outer face*. Two planar drawings with the same planar embedding have the same faces. However, such drawings could still differ for their outer face. The *dual graph* D of a planar embedded graph G is the graph with a vertex for each face of G and with an edge $e(D)$ between two vertices if the corresponding faces share an edge $e(G)$; edge $e(D)$ is *dual* to edge $e(G)$. In the following we will deal both with biconnected (that is vertex 2-connected) and with simply connected (that is vertex 1-connected) embedded planar graphs. In the former case, the "*number of vertices in a face*" is trivially defined as the number of vertices incident to the face, while in the latter one is meant to be the number of occurrences of vertices in the border of the face.

A *clustered graph* is a pair $C(G,T)$, where G is a planar graph and T is a rooted tree whose leaves are the vertices of G. Graph G and tree T are called *underlying graph* and *inclusion tree*, respectively. Each internal node μ of T corresponds to the subset $V(\mu)$ (called *cluster*) of the vertices of G that are leaves of the subtree of T rooted at μ; $G(\mu)$

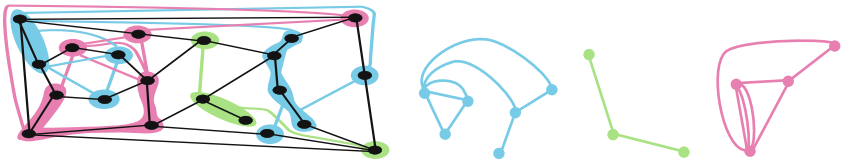


Fig. 1. (a) An embedded flat clustered graph C and its candidate saturating edges. Different clusters have different colors. (b)–(c)–(d) Multigraphs $\mathcal{G}_i$ for C.

denotes the subgraph of G induced by the vertices in $V(\mu)$. If each cluster induces a connected subgraph of G, then C is *c-connected*, otherwise C is non-c-connected. An *embedded clustered graph* is a clustered graph such that G is connected and the planar embedding of the underlying graph of C is fixed. A *flat clustered graph* is such that the number of nodes in any path from the root to a leaf of T is three. When referring to a flat clustered graph, given a vertex v of the underlying graph we say that the *cluster of* v is its parent in T. Also, we call *clusters* only the children of the root.

A drawing of a clustered graph $C(G,T)$ consists of a drawing of G and of a representation of each node μ of T as a simple closed region $R(\mu)$ such that: (i) $R(\mu)$ contains the drawing of $G(\mu)$; (ii) $R(\mu)$ contains a region $R(\nu)$ iff ν is a descendant of μ in T; and (iii) the borders of any two regions don't intersect. Consider an edge e and a node μ of T. If e crosses the boundary of $R(\mu)$ more than once, we say that edge e and region $R(\mu)$ have an *edge-region crossing*. A drawing of a clustered graph is *c-planar* if it does not have edge crossings or edge-region crossings. A clustered graph is c-planar if it admits a c-planar drawing. An embedded clustered graph is c-planar if it admits a c-planar drawing in which the embedding of G is preserved.

Consider an embedded flat clustered graph $C(G,T)$. For each face f of G a set of *candidate saturating edges* is defined as follows: Let O be the clockwise circular order of the vertices on the border of f. Subdivide such vertices into subsets such that each subset V_i contains a maximal sequence of consecutive vertices in O belonging to the same cluster. Introduce a candidate saturating edge for each $V_i \neq V_j$ such that (i) V_i and V_j contain vertices of the same cluster μ_k and (ii) V_i and V_j are in different connected components of $G(\mu_k)$. Candidate saturating edges are edges that can be added to the clustered graph to make it c-connected (see Fig. 1). For a cluster μ_i of T we define $\mathcal{G}_i$ as the embedded multigraph whose vertices are the connected components of $G(\mu_i)$ and whose edges are the candidate saturating edges. The embedding of $\mathcal{G}_i$ is given by the order of the faces around the vertices of G. Observe that $\mathcal{G}_i$ does not have self-loops and is, in general, non-planar. However, possible crossings are only between edges introduced in the same face of G. Two candidate saturating edges e_1 and e_2, joining connected components $G_1(\mu_i)$ and $G_2(\mu_i)$ of $G(\mu_i)$, and $G_1(\mu_j)$ and $G_2(\mu_j)$ of $G(\mu_j)$, respectively, with $\mu_i \neq \mu_j$ and with e_1 and e_2 in the same face f of G, have a *conflict* if $G_1(\mu_i)$, $G_1(\mu_j)$, $G_2(\mu_i)$, and $G_2(\mu_j)$ appear in this order around the border of f. Informally, two candidate saturating edges have a conflict if adding both of them to the clustered graph causes a crossing. The following theorem shows the role of candidate saturating edges for the c-planarity of a flat embedded clustered graph. Even if not explicitly stated, Theorem 1 has been used in [3].

Theorem 1. *An embedded flat clustered graph $C(G,T)$ is c-planar if and only if: (1) G is planar; (2) there exists a face f in G such that when f is chosen as outer face for G no cycle composed by vertices of the same cluster encloses a vertex of a different cluster; (3) it is possible to augment G to a graph G' by adding a subset of the candidate saturating edges of C so that no two added edges have a conflict and each cluster induces in G' exactly one connected component.*

Hence, given an embedded flat clustered graph $C(G,T)$, if Conditions 1 and 2 are satisfied by G, the problem of testing the c-planarity of C can be restated as the problem of testing if it is possible to select from multigraphs $\mathcal{G}_i$ a set of candidate saturating edges to enforce Condition 3. If such a set exists, we call it a *saturator* of C.

Lemma 1. *An embedded flat clustered graph $C(G,T)$ admits a saturator if and only if it admits an acyclic saturator.*

Hence, testing the c-planarity of an embedded flat clustered graph satisfying Conditions 1 and 2 of Theorem 1 is the same of testing if there exists a spanning tree of each $\mathcal{G}_i$ where no two edges in different spanning trees have a conflict.

3 A Characterization

We restrict ourselves to embedded flat clustered graphs in which each candidate saturating edge has a conflict with at most one other candidate saturating edge. We call *single-conflict* an embedded flat clustered graph satisfying such a property. Consider a single-conflict embedded flat clustered graph $C(G,T)$ and the multigraph $\mathcal{G}_i$ for each cluster μ_i in T. We have the following structural lemma.

Lemma 2. *If a graph $\mathcal{G}_i$ contains two crossing edges e_1 and e_2, then e_1 and e_2 have no conflict with edges of other multigraphs.*

By Lemma 3, we can assume that in the interesting cases the $\mathcal{G}_i$'s are connected.

Lemma 3. *If there exists a $\mathcal{G}_i$ that is not connected, then C is not c-planar.*

There are edges in the $\mathcal{G}_i$'s that must be used in any saturator of C and edges that are not used in any saturator. Further, there are edges that can be supposed to belong to a saturator without altering the possibility to have one. Roughly speaking, such edges do not belong to the "core" of the problem. Hence, in the following we simplify the $\mathcal{G}_i$'s with an algorithm that either returns that C is not c-planar or returns a structure where there are no trivial choices. For this purpose, we define two operations on $\mathcal{G}_i$.

The operation of *removing* an edge e from $\mathcal{G}_i$ corresponds to the choice of not using e in the saturator of C. Notice that, when an edge e is removed from $\mathcal{G}_i$, an edge of $\mathcal{G}_j$, with $i \neq j$, that possibly had a conflict with e does not have a conflict any longer.

The operation of *collapsing* an edge e with end-vertices u and v in $\mathcal{G}_i$ corresponds to the choice of using e in the saturator of C. It consists of: (i) deleting vertices u and v, (ii) removing from $\mathcal{G}_i$ all edges between u and v, and (iii) inserting in $\mathcal{G}_i$ a new vertex whose incident edges are those of u and v. The embedding of $\mathcal{G}_i$ is preserved. The

collapsing operation "preserves" the conflicts. Namely, let e_i be an edge of $\mathcal{G}_i$ incident to u or in v but not in both. Suppose that e_i has a conflict (has not a conflict) with an edge e_j of $\mathcal{G}_j$, with $i \neq j$. After collapsing edge e in a new vertex w the edge incident to w corresponding to e_i has a conflict (resp. has not a conflict) with e_j.

The algorithm is as follows. Repeatedly modify the $\mathcal{G}_i$'s by applying one of the following simplifications. From now on, $\mathcal{G}_i$ denotes the multigraph obtained from the starting $\mathcal{G}_i$ after some simplifications have been performed. **Simplification 1:** If there exists an edge e of a $\mathcal{G}_i$ that has no conflict, then collapse e in $\mathcal{G}_i$. **Simplification 2:** If there exist a separating edge e_i and a non-separating edge e_j that are in $\mathcal{G}_i$ and $\mathcal{G}_j$, respectively, and that conflict each other, then collapse e_i in $\mathcal{G}_i$ and remove e_j from $\mathcal{G}_j$. **Simplification 3:** If there exist two separating edges e_i and e_j that are in $\mathcal{G}_i$ and $\mathcal{G}_j$, respectively, and that conflict each other, then stop because C is not c-planar.

If the algorithm does not stop for non-c-planarity, we call the final $\mathcal{G}_i$ *candidate saturating graph* for cluster μ_i and we denote it by $\mathcal{G}_i^*$. Also, we say that μ_i *admits a candidate saturating graph*. The following properties can be easily proved.

Property 1. None of Simplifications 1, 2, and 3 could disconnect any $\mathcal{G}_i$.

Property 2. The subgraphs induced by the collapsed edges are acyclic.

Property 3. Candidate saturating graphs are planar embedded and edge 2-connected.

Property 4. Any edge of a candidate saturating graph has exactly one conflict with an edge of a different candidate saturating graph.

We now prove that each simplification performed by the algorithm preserves the possibility of finding a saturator of C. Consider simplification s_m, that is performed at a certain step of the simplification phase. s_m can be one of Simplification 1, 2, or 3. Denote by $s_0, s_1, \ldots, s_{m-1}$ the simplifications performed before s_m and denote by E the set of edges collapsed while applying $s_0, s_1, \ldots, s_{m-1}$. Inductively, suppose that if an acyclic saturator of C exists, there exists an acyclic saturator composed by the edges of E plus some of the edges remaining in the $\mathcal{G}_i$'s after simplifications $s_0, s_1, \ldots, s_{m-1}$. This is indeed the case when no simplification has been performed yet.

Lemma 4. *Consider an edge e of $\mathcal{G}_i$ with no conflict. We have that C admits a saturator only if it admits an acyclic saturator containing e and containing the edges of E.*

Proof: Suppose C admits a saturator. Then, by Lemma 1 and by inductive hypothesis, it admits an acyclic saturator S such that $E \subseteq S$. If S contains e the statement follows. Otherwise, observe that since S is a saturator, there exists a set $S' \subseteq S$ of edges forming a path between the end-vertices u and v of e. Hence, the edges of $S' \cup \{e\}$ form a cycle. Not all the edges of S' belong to E, otherwise u and v would not have been distinct vertices in $\mathcal{G}_i$ after simplifications $s_0, s_1, \ldots, s_{m-1}$. Hence, the set S^* of edges obtained from S by inserting e and by removing any edge of S' not in E is an acyclic saturator of C containing E and e. Namely, all the connected components of C are connected by a path of edges in S^* and since e has no conflict and S is a saturator, then no two edges in S^* have a conflict. □

Lemma 5. *Consider two edges e_i and e_j of two distinct multigraphs $\mathcal{G}_i$ for cluster μ_i and $\mathcal{G}_j$ for cluster μ_j, respectively. Suppose that e_i and e_j conflict each other. Also, suppose that e_i is a separating edge, while e_j is not. Then C admits a saturator only if it admits an acyclic saturator containing e_i, containing E, and not containing e_j.*

Proof: Suppose C admits a saturator. Then, by Lemma 1 and by inductive hypothesis, it admits an acyclic saturator S such that $E \subseteq S$. Since at step s_m end-vertices u and v of e_i are in $\mathcal{G}_i$, then no path composed by edges of E connects u and v. Since e_i is a separating edge, then if e_i is not in S adding the edges of S to G would not connect $G(\mu_i)$. Hence $e_i \in S$. Since no two conflicting edges can be in S, then $e_j \notin S$. □

Lemma 6. *Consider two separating edges e_i and e_j of two distinct multigraphs $\mathcal{G}_i$ for cluster μ_i and $\mathcal{G}_j$ for cluster μ_j, respectively. Suppose that e_i and e_j conflict each other. We have that C is not c-planar.*

Proof: Suppose that C admits a saturator. Then, by inductive hypothesis, it admits an acyclic saturator S such that $E \subseteq S$. Since at step s_m the end-vertices u and v of e_i (the end-vertices w and x of e_j) are in $\mathcal{G}_i$ (are in $\mathcal{G}_j$), then no path composed by edges of E connects u and v (connects w and x). Since e_i and e_j are separating edges, then if e_i (e_j) is not in S, adding the edges of S to G would not connect $G(\mu_i)$ ($G(\mu_j)$). However, S cannot contain both e_i and e_j, that conflict each other. □

Let μ_i and μ_j be two distinct clusters admitting candidate saturating graphs $\mathcal{G}_i^*$ and $\mathcal{G}_j^*$, respectively. We define graph $\mathcal{G}_{i,j}^*$ as the planar embedded subgraph of $\mathcal{G}_i^*$ induced by the edges having a conflict with the edges of $\mathcal{G}_j^*$. We have:

Theorem 2. *A single-conflict embedded flat clustered graph $C(G,T)$ is c-planar iff: (1) G is planar; (2) There exists a face f in G such that when f is chosen as outer face for G no cycle composed by vertices of the same cluster encloses a vertex of a different cluster; (3) Each cluster of C admits a candidate saturating graph; (4) For each pair of distinct clusters μ_i and μ_j, $\mathcal{G}_{i,j}^*$ is edge 2-connected; and (5) For each pair of distinct clusters μ_i and μ_j, $\mathcal{G}_{i,j}^*$ is dual to $\mathcal{G}_{j,i}^*$.*

Proof: Let S be an acyclic saturator of C and let u and v be two vertices of candidate saturating graph $\mathcal{G}_i^*$. Denote by $S(u,v)$ the path of S connecting u and v. If edges e_i and e_j of candidate saturating graphs $\mathcal{G}_i^*$ and $\mathcal{G}_j^*$ conflict each other, we write $e_i \oplus e_j$.

The necessity of Conditions (1) and (2) descends from the one of Conditions 1 and 2 of Theorem 1. We prove the necessity of Condition (3). Suppose that C does not admit candidate saturating graphs. Two cases are possible: Either before the simplification phase one of the $\mathcal{G}_i$'s is not connected, or during the simplification phase two separating conflicting edges are found. In the former case the non-c-planarity of C descends from Lemma 3, in the latter case from Lemma 6.

Now we deal with Condition (4). Suppose that $\mathcal{G}_{i,j}^*$ is not connected. Denote by v_1 and v_2 vertices in different connected components. Suppose, for a contradiction, that an acyclic saturator S of C exists. Consider $S(v_1,v_2)$ (see Fig. 2.a). Since v_1 and v_2 are in different components of $\mathcal{G}_{i,j}^*$, there exists an edge $(u,v) \in S(v_1,v_2)$ s. t. $(u,v) \oplus (w,x)$, where $(w,x) \in \mathcal{G}_k^*$, with $k \neq i,j$. Consider $S(w,x)$. Each edge of $S(w,x)$ cannot have a conflict with any edge of $S(v_1,v_2)$, otherwise S would contain two conflicting edges, and with any edge e of $\mathcal{G}_{i,j}^*$, otherwise e would conflict with

two candidate saturating edges. Hence, $\mathcal{G}^*_{j,i}$ has at least two connected components. Let u_1 and u_2 be two vertices in such components, respectively. Then, $S(u_1, u_2)$ either contains an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in S(v_1, v_2)$, or contains an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in S(w, x)$, implying that S contains two conflicting edges.

Now suppose that $\mathcal{G}^*_{i,j}$ has a separating edge (u, v). By construction $(u, v) \oplus (w, x)$, where $(w, x) \in \mathcal{G}^*_{j,i}$. Suppose, for a contradiction, that a saturator S of C exists.

1. If $(u, v) \notin S$, then consider $S(u, v)$ (see Fig. 2.b). Since (u, v) is a separating edge for $\mathcal{G}^*_{i,j}$, then there exists an edge $(u', v') \in S(u, v)$ s. t. $(u', v') \oplus (w', x')$, where $(w', x') \in \mathcal{G}^*_k$, with $k \neq i, j$. Hence, $S(w', x')$ either contains an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in S(u, v)$, implying that S contains two conflicting edges, or contains an edge e_1 conflicting with (u, v), implying that (u, v) conflicts with two candidate saturating edges.
2. If $(u, v) \in S$, then consider $S(w, x)$.
   - If an edge $(w', x') \in S(w, x)$ is s. t. $(w', x') \oplus (u', v')$, with $(u', v') \notin \mathcal{G}^*_{i,j}$, a contradiction is obtained as in the previous case (see Fig. 2.c).
   - Otherwise, consider any edge $(w', x') \in S(w, x)$ and edge $(u', v') \in \mathcal{G}^*_{i,j}$ s. t. $(u', v') \oplus (w', x')$. Let v (v') be the endpoint of (u, v) (resp. of (u', v')) outside cycle $S(w, x) \cup (w, x)$.
     - If $u = u'$ or if all edges of $S(u, u')$ have conflicts with edges of $\mathcal{G}^*_{j,i}$ (see Fig. 2.d), consider $S(v, v')$. Then there exists an edge $(u'', v'') \in S(v, v')$ s. t. $(u'', v'') \oplus (w'', x'')$, where $(w'', x'') \in \mathcal{G}^*_k$, with $k \neq i, j$, otherwise (u, v) would not be a separating edge. Hence, $S(w'', x'')$ either contains an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in S(v, v')$, implying that S contains two conflicting edges, or an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in S(u, u')$, implying that S contains two conflicting edges, or an edge e_1 s. t. $e_1 \oplus (u', v')$, implying that (u', v') conflicts with two candidate saturating edges, or an edge e_1 s. t. $e_1 \oplus (u, v)$, implying that (u, v) conflicts with two candidate saturating edges.
     - If $u \neq u'$ and $S(u, u')$ contains at least one edge (u'', v'') s. t. $(u'', v'') \oplus (w'', x'')$, where $(w'', x'') \in \mathcal{G}^*_k$, with $k \neq i, j$ (see Fig. 2.e), then $S(w'', x'')$ contains either an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in S(w, x)$, implying that S contains two conflicting edges, or an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in S(u, u')$, or an edge e_1 s. t. $e_1 \oplus (u', v')$, implying that (u', v') conflicts with two candidate saturating edges, or an edge e_1 s. t. $e_1 \oplus (w, x)$, implying that (w, x) conflicts with two candidate saturating edges.

Now we prove the necessity of Condition (5). Each edge of $\mathcal{G}^*_{i,j}$ has a conflict with (and hence is dual to) one edge of $\mathcal{G}^*_{j,i}$ and vice versa. By the necessity of Condition (4), we can assume that both $\mathcal{G}^*_{i,j}$ and $\mathcal{G}^*_{j,i}$ are edge 2-connected. Hence $\mathcal{G}^*_{i,j}$ is not dual to $\mathcal{G}^*_{j,i}$ only if there is a face of $\mathcal{G}^*_{i,j}$ that contains in its interior two vertices of $\mathcal{G}^*_{j,i}$, or vice versa. Suppose w.l.o.g. that a face f of $\mathcal{G}^*_{i,j}$ contains in its interior two vertices u and v of $\mathcal{G}^*_{j,i}$. Suppose, for a contradiction, that a saturator S of C exists. Consider $S(u, v)$.

1. If the vertices of $S(u, v)$ are in part inside f and in part outside f (see Fig. 2.f), consider two vertices v_1 and v_2 in different connected components, disconnected by $S(u, v)$, of f. Consider $S(v_1, v_2)$. There exists an edge $(w, x) \in S(v_1, v_2)$ s. t.

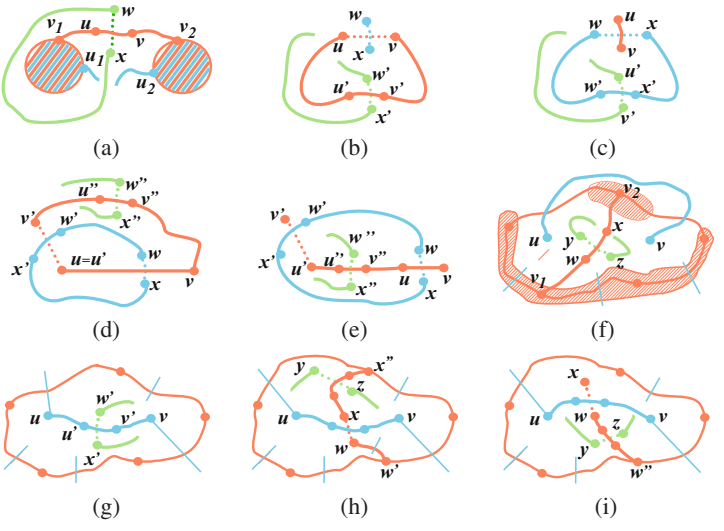


Fig. 2. Illustrations for the necessity of the conditions of Theorem 2. Edges of $\mathcal{G}_i^*$ are red, edges of $\mathcal{G}_j^*$ are light blue, and edges of $\mathcal{G}_k^*$ are green.

$(w,x) \oplus (y,z)$, where $(y,z) \in \mathcal{G}_k^*$, with $k \neq i,j$, otherwise f would not be a face. Hence, $S(y,z)$ either contains an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in S(v_1,v_2)$, implying that S contains two conflicting edges, or an edge e_1 conflicting with an edge $e_2 \in f$, implying that e_2 conflicts with two candidate saturating edges.

2. Otherwise, $S(u,v)$ is composed by vertices all lying inside f.
   - If there is an edge $(u',v') \in S(u,v)$ s. t. $(u',v') \oplus (w',x')$, where $(w',x') \in \mathcal{G}_k^*$, with $k \neq i,j$ (see Fig. 2(g)), then $S(w',x')$ either contains an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in S(u,v)$, implying that S contains two conflicting edges, or an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in f$, implying that e_2 conflicts with two candidate saturating edges, or an edge e_1 s. t. $e_1 \oplus e_2$, with e_2 dual to an edge of f, implying that e_2 conflicts with two candidate saturating edges.
   - Otherwise, any edge of $S(u,v)$ is dual to an edge of $\mathcal{G}_{i,j}^*$. Consider any edge (w,x) dual to an edge of $S(u,v)$.
     - If $w \in f$ or if there exists a vertex $w' \in f$ s. t. any edge of $S(w,w')$ conflicts with an edge of $\mathcal{G}_{j,i}^*$ (see Fig. 2(h)), then $x \notin f$ and there exists no vertex x' in f s. t. all edges of $S(x,x')$ conflict with edges of $\mathcal{G}_{j,i}^*$, otherwise f would not be a face. Consider any vertex $x'' \in f$ and $S(x,x'')$. Then, there exists an edge in $S(x,x'')$ that has a conflict with an edge (y,z) in $\mathcal{G}_k^*$, with $k \neq i,j$. Hence, $S(y,z)$ either contains an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in S(u,v)$, implying that S contains two conflicting edges, or contains an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in S(x,x'')$, implying that S contains two conflicting edges, or contains an edge e_1 s. t. $e_1 \oplus e_2$, with $e_2 \in f$, implying that e_2 conflicts with two candidate saturating edges, or contains an edge e_1 s. t. $e_1 \oplus e_2$, with e_2 dual to an edge in f, implying that e_2 conflicts with two candidate saturating edges.

- If $w \notin f$ and there exists no vertex $w' \in f$ s. t. any edge of $S(w, w')$ conflicts with an edge of $\mathcal{G}^*_{j,i}$ (see Fig. 2i), then there exists a vertex $w'' \in f$ s. t. $S(w, w'')$ contains an edge e_1 s. t. $e_1 \oplus (y, z)$, with $(y, z) \in \mathcal{G}^*_k$, with $k \neq i, j$, and a contradiction is derived as in the previous case.

We prove the sufficiency of Conditions 1, 2, 3, 4, and 5 for the c-planarity of C. Consider any planar drawing of G satisfying Conditions 1 and 2 and hence satisfying Conditions 1 and 2 of Theorem 1. We show how to construct an acyclic saturator S of C satisfying Condition 3 of Theorem 1. Apply the simplification phase, choosing an acyclic set E of edges to be in S and obtaining a candidate saturating graph $\mathcal{G}^*_i$ for each cluster μ_i (this can be done since C satisfies Condition 3). Order the clusters in whichever way $\mu_1, \mu_2, \ldots, \mu_m$. For any pair of clusters μ_i and μ_j, with $i < j$, choose a spanning tree $\mathcal{T}^*_{i,j}$ of $\mathcal{G}^*_{i,j}$ ($\mathcal{T}^*_{i,j}$ can be found since, by Condition 4, $\mathcal{G}^*_{i,j}$ is edge 2-connected). Remove from $\mathcal{G}^*_{j,i}$ all edges dual to edges of $\mathcal{T}^*_{i,j}$, obtaining a graph $\mathcal{T}^*_{j,i}$. We claim that $\mathcal{T}^*_{j,i}$ is a spanning tree of $\mathcal{G}^*_{j,i}$. By Condition 5, $\mathcal{G}^*_{i,j}$ and $\mathcal{G}^*_{j,i}$ are dual graphs, and the edges of a cycle in $\mathcal{G}^*_{i,j}$ are dual to the edges of a cutset in $\mathcal{G}^*_{j,i}$, and vice versa (Lemma 1.4 of [11]). Hence, if $\mathcal{T}^*_{j,i}$ has more than one connected component, the edges removed from $\mathcal{G}^*_{j,i}$ form a cutset for $\mathcal{G}^*_{j,i}$, and those of $\mathcal{T}^*_{i,j}$ form a cycle, contradicting the hypothesis that $\mathcal{T}^*_{i,j}$ is a tree. If a set of edges of $\mathcal{T}^*_{j,i}$ is a cycle, the edges dual to such a cycle form a cutset for $\mathcal{G}^*_{i,j}$, contradicting the hypothesis that $\mathcal{T}^*_{i,j}$ is spanning for $\mathcal{G}^*_{i,j}$. For any pair of clusters μ_i and μ_j, with $i < j$, add the edges of $\mathcal{T}^*_{i,j}$ and of $\mathcal{T}^*_{j,i}$ to S. We claim that S is a saturator of C. Edges chosen in the simplification phase do not conflict each other by construction. Such edges do not conflict with edges of trees $\mathcal{T}^*_{i,j}$. In fact, an edge in $\mathcal{T}^*_{i,j}$ conflicts only with an edge in $\mathcal{G}^*_j$, with $i \neq j$. By construction, edges of the $\mathcal{T}^*_{i,j}$'s do not conflict each other. Hence, S does not have two conflicting edges. It's easy to see that, after G has been augmented to a graph G' by adding the edges of S to it, each cluster μ_i has exactly one connected component. Namely, distinct connected components of $G(\mu_i)$ are represented after the simplification phase by distinct vertices in $\mathcal{G}^*_i$, that is edge 2-connected and that is partitioned in edge 2-connected subgraphs $\mathcal{G}^*_{i,j}$. Since a spanning tree is chosen to be in S for any $\mathcal{G}^*_{i,j}$, then $\bigcup_j \mathcal{T}^*_{i,j}$ is spanning for $\mathcal{G}^*_i$ and $G'(\mu_i)$ has exactly one connected component. Finally, suppose that $G'(\mu_i)$ has a cycle c containing an edge of S. By construction, edges chosen in the simplification phase only join different connected components of $G(\mu_i)$ and no edge of c could belong to some $\mathcal{G}^*_{i,j}$, otherwise $G'(\mu_j)$ would be disconnected. □

Theorem 3. *An embedded flat clustered graph $C(G, T)$ with at most five vertices per face is c-planar if and only if: (1) G is planar; (2) There exists a face f in G such that when f is chosen as outer face for G no cycle composed by vertices of the same cluster encloses a vertex of a different cluster; (3) Each cluster of C admits a candidate saturating graph; and (4) For each pair of distinct clusters μ_i and μ_j, $\mathcal{G}^*_{i,j}$ is edge 2-connected; and (5) For each pair of distinct clusters μ_i and μ_j, $\mathcal{G}^*_{i,j}$ is dual to $\mathcal{G}^*_{j,i}$.*

Proof: Consider any face f of G. Since f has at most five vertices, then it has at most two connected components of each cluster, so it has at most one candidate saturating edge per cluster. Since at least two vertices are necessary for each candidate saturating edge, then f contains candidate saturating edges for at most two clusters. Hence, C is a single-conflict embedded flat clustered graph and Theorem 2 applies. □

4 An Efficient c-Planarity Testing Algorithm

We use Theorem 3 to derive a linear time and space c-planarity testing algorithm for embedded flat clustered graphs with at most five vertices per face. The algorithm can be extended to test the c-planarity of single-conflict embedded flat clustered graphs relying on Theorem 2. The details of the extension are omitted for brevity. Anyway, we will emphasize the steps of the algorithm that have to be modified for this purpose.

Let $C(G,T)$ be an n-vertex embedded flat clustered graph with at most five vertices per face. To test Condition (1) of Theorem 3, it is sufficient to test if G is a planar embedding. This can be done in $O(n)$ time and space with the techniques in [10].

To test Condition (2), we observe that a face exists satisfying such a condition iff the embedded clustered graph is *hole-free*, that is, chosen an arbitrary face as external, a cycle c of G doesn't exist composed by vertices of the same cluster μ such that c has a vertex inside and a vertex outside both belonging to clusters different from μ. A linear-time algorithm for checking if an embedded clustered graph is hole-free has been provided in [5] in the case of c-connected clustered graphs. However, we can use the same algorithm because of the following lemma.

Lemma 7. *Let $C(G,T)$ be a clustered graph. Let $C'(G,T')$ be the c-connected clustered graph obtained from C as follows. Each node ν of T is replaced in T' by nodes $\nu_1,\ldots,\nu_h$, one for each of the $h \geq 1$ connected components of $G(\nu)$. Let $\mu_1,\ldots,\mu_k$ be the nodes replacing the parent μ of ν. The parent of ν_j in T' is the node μ_i such that $G(\nu_j)$ is a subgraph of $G(\mu_i)$. We have that C is hole-free iff C' is hole-free.*

In order to test Condition (3) we create multigraphs $\mathcal{G}_i$. This is done in $O(n)$ time as follows. **Connected Components.** For each node μ of T compute the connected components of $G(\mu)$. This is done in linear time and space. **Candidate saturating edges.** We insert candidate saturating edges inside the faces of G. Consider a face f. Construct maximal sequences of vertices consecutive on the border of f and belonging to the same cluster. For any two sequences S_1 and S_2 that have vertices belonging to the same cluster, take a vertex $v_1 \in S_1$ and a vertex $v_2 \in S_2$. If the connected component associated to v_1 is different from the one associated to v_2 (this can be tested in constant time), then insert a candidate saturating edge between v_1 and v_2. At most two edges are inserted inside f. The described insertion can be performed in constant time and hence in linear time for all faces of G. This step is more tricky when considering single-conflict clustered graphs. In this case, in order to achieve total linear time special care must be put when considering groups of candidate saturating edges between vertices of the same cluster and when determining the conflicts between candidate saturating edges. **Multigraphs $\mathcal{G}_i$.** Consider cluster μ_i. Add a vertex to $\mathcal{G}_i$ for each connected component of $G(\mu_i)$. For each candidate saturating edge e insert an edge between the connected components joined by e. The construction of the $\mathcal{G}_i$'s can be done so that their embeddings are those induced by the adjacencies of the faces of G. Further, such a construction can be done in linear time and space because of the following:

Property 5. $\sum_{\mu_i} |\mathcal{G}_i| = O(n)$, where $|\mathcal{G}_i|$ is the size of the graph.

Property 5 does not hold when considering single-conflict embedded flat clustered graphs, that can generally have faces with a linear number of incident vertices. However,

the arrangement of the candidate saturating edges in the single-conflict setting allows to reduce the size of the construction introducing only an overall linear number of them.

Now we show how to test if Condition (3) of Theorem 3 is satisfied. First, test if the $\mathcal{G}_i$'s are connected. If not, return non-c-planar.

We equip each $\mathcal{G}_i$ with a data structure supporting the following update and query operations: remove an edge, collapse (identify the end-vertices of) an edge and merge the embeddings of its end-vertices, answer if an edge is a separating edge, answer if an edge has a conflict and in case output the conflicting edge. Observe the difference between the above definition of the collapse operation and the one given in Section 3. A data structure exists that can be set-up in linear time and that performs each of the above operations in constant time. In fact, all of them are trivial graph operations, with the exception of answering if an edge e is a separating edge. We equip each edge with two pointers to the two identifiers of the incident faces. When an edge e is removed from $\mathcal{G}_i$ we simply modify the identifier of one of the two faces former incident on e. To answer the query in constant time we check if the two faces around e are the same face. Also, we compute a set $\mathcal{F}$ of candidate saturating edges that have no conflict. For each edge e of $\mathcal{F}$ we compute the set $\mathcal{E}$ of edges parallel to e. Such computations are performed in linear time. We will show how to use $\mathcal{F}$ and sets $\mathcal{E}$ during the simplification steps.

We show how to apply Simplification 1. Construct the set $\mathcal{F}'$ of the edges of any spanning forest of $\mathcal{F}$. Set $\mathcal{F}'' = \emptyset$. Take each edge e_1 of $\mathcal{F}'$. Consider the set $\mathcal{E}$ of edges parallel to e_1. For each edge $e_2 \neq e_1$ in $\mathcal{E}$, if e_2 has a conflict with an edge e_2^*, add e_2^* to $\mathcal{F}''$. After this work has been performed on all the edges of $\mathcal{F}'$, collapse all of such edges, removing self-loops. Set $\mathcal{F}''$ contains all the edges that became conflict-free after the previous step. The edges of $\mathcal{F}''$ do not have multiple edges:

Lemma 8. *The edges of set $\mathcal{F}''$ do not have multiple edges.*

Perform Simplification 1 on the edges of $\mathcal{F}''$. The above lemma guarantees that after this second pass no new conflict-free edge can be originated.

Now, Simplification 2 is applied till the $\mathcal{G}_i$'s are edge 2-connected or the non-c-planarity of C is stated. First, construct a set $\mathcal{H}$ of separating edges as follows. For each edge e in $\mathcal{G}_i$ verify if the faces incident to e are the same. If yes, then add e to $\mathcal{H}$. This computation takes time linear in the number of edges in the $\mathcal{G}_i$'s. Now, for each edge e in $\mathcal{H}$, check if edge e^* conflicting with e belongs to $\mathcal{H}$. If yes, return non-c-planar, otherwise delete e^* and collapse e. After this has been done for all edges in $\mathcal{H}$, other separating edges could have been created in $\mathcal{G}_i$. However, if this happens, then we can conclude that C is not c-planar as stated in the following lemmas:

Lemma 9. *Consider a face f of $\mathcal{G}_i$. Suppose that f contains a separating pair composed by edges (u_1, u_2) and (u_3, u_4). Suppose that (u_1, u_2) has a conflict with edge (v_1, v_2) that is a separating edge, and that (u_3, u_4) has a conflict with edge (v_3, v_4). We have that C is not c-planar.*

Lemma 10. *Suppose that each edge of $\mathcal{H}$ has a conflict with a non-separating edge. Collapse the edges in $\mathcal{H}$, repeatedly applying Simplification 2. Either the resulting multigraphs $\mathcal{G}_i$ are edge 2-connected or C is not c-planar.*

For each pair of distinct clusters μ_i and μ_j, we check if $\mathcal{G}^*_{i,j}$ is edge 2-connected (Condition (4) of Theorem 3) and if $\mathcal{G}^*_{i,j}$ is dual to $\mathcal{G}^*_{j,i}$ (Condition (5) of Theorem 3). This is easily done in linear time because of the following property.

Property 6. $\sum |\mathcal{G}^*_{i,j}| = O(n)$, where $|\mathcal{G}^*_{i,j}|$ is the size of the graph.

Hence, we can conclude the section with the following theorem.

Theorem 4. *The c-planarity of an n-vertex embedded flat clustered graph $C(G,T)$ with at most five vertices per face can be tested in $O(n)$ time and space.*

5 Conclusions

We remark that the simplification phase described in Section 3 is a preprocessing that can be performed on any embedded flat clustered graph. This allows to reduce the problem of testing the c-planarity of such graphs to the one of deciding whether a set of edge 2-connected candidate saturating graphs admits a set of non-conflicting spanning trees. However, it's rather easy to see that the characterization shown in Theorem 2 does not hold for general embedded flat clustered graphs.

We conclude by providing a list of families of embedded clustered graphs for which, in our opinion, determining the time complexity of a c-planarity testing is worth of interest: (i) single-conflict general (non-flat) embedded clustered graphs; (ii) embedded flat clustered graphs where each face of the underlying graph has at most two (or a constant number of) vertices of the same cluster; and (iii) embedded flat clustered graphs.

References

1. Cornelsen, S., Wagner, D.: Completely connected clustered graphs. Journal of Discrete Algorithms 4(2), 313–323 (2006)
2. Cortese, P.F., Di Battista, G.: Clustered planarity. In: ACM SoCG 2005, pp. 32–34 (2005)
3. Cortese, P.F., Di Battista, G., Patrignani, M., Pizzonia, M.: Clustering cycles into cycles of clusters. Journal of Graph Algorithms and Applications 9(3), 391–413 (2005)
4. Cortese, P.F., Di Battista, G., Patrignani, M., Pizzonia, M.: On embedding a cycle in a plane graph. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 49–60. Springer, Heidelberg (2006)
5. Dahlhaus, E.: A linear time algorithm to recognize clustered graphs and its parallelization. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 239–248. Springer, Heidelberg (1998)
6. Di Battista, G., Frati, F.: Efficient c-planarity testing for embedded flat clustered graphs with small faces. Tech. Report RT-DIA-119-2007, Dip. Inf. e Aut., Univ. Roma Tre (2007)
7. Feng, Q., Cohen, R.F., Eades, P.: Planarity for clustered graphs. In: Spirakis, P.G. (ed.) ESA 1995. LNCS, vol. 979, pp. 213–226. Springer, Heidelberg (1995)
8. Goodrich, M.T., Lueker, G.S., Sun, J.Z.: C-planarity of extrovert clustered graphs. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 211–222. Springer, Heidelberg (2006)
9. Gutwenger, C., Jünger, M., Leipert, S., Mutzel, P., Percan, M., Weiskircher, R.: Advances in c-planarity testing of clustered graphs. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 220–235. Springer, Heidelberg (2002)
10. Kirkpatrick, D.G.: Establishing order in planar subdivisions. Discrete & Computational Geometry 3, 267–280 (1988)
11. Nishizeki, T., Chiba, N.: Planar Graphs: Theory and Algorithms. North-Holland (1988)

Clustered Planarity: Small Clusters in Eulerian Graphs

Eva Jelínková[1], Jan Kára[2], Jan Kratochvíl[1,2], Martin Pergel[1,⋆], Ondřej Suchý[1], and Tomáš Vyskočil[1]

[1] Department of Applied Mathematics
[2] Institute for Theoretical Computer Science[⋆⋆]
Charles University
Malostranské nám. 25, 118 00 Praha, Czech Republic
{eva,kara,honza,perm,suchy,whisky}@kam.mff.cuni.cz

Abstract. We present several polynomial-time algorithms for c-planarity testing for clustered graphs with clusters of size at most three. The most general result concerns a special class of Eulerian graphs, namely graphs obtained from a fixed-size 3-connected graph by multiplying and then subdividing edges. We further give algorithms for 3-connected graphs, and for graphs with small faces. The last result applies with no restrictions on the cluster size.

1 Introduction

Clustered planarity (or shortly, c-planarity) has recently become an intensively studied topic in the area of graph and network visualization. In many situations one needs to visualize a complicated inner structure of graphs and networks. Clustered graphs—graphs with recursive clustering structures over the vertices—provide a possible model of such a visualization, and as such they find applications in many practical problems, e.g., management information systems, social networks or VLSI design tools [4]. However, from the theoretical point of view, the computational complexity of deciding c-planarity is still an open problem and it is regarded as one of the challenges of the contemporary graph drawing. Our aim is to add another pebble to the mosaic of known partial results on c-planarity by studying the case of small clusters.

Regarding the graph notations, we follow standard notation on finite loopless graphs. A graph is an ordered pair $G = (V, E)$. By $\overline{G}$ we denote its edge complement (i.e., $(V, \binom{V}{2} \setminus E)$). For a vertex $v \in V$ by $N(v)$ we denote its set of neighbors.

Let $G = (V, E)$ be a graph. A *cluster set* on G is a set $\mathcal{C} \subseteq \mathcal{P}(V(G))$ such that for all $C, D \in \mathcal{C}$, either C and D are disjoint or they are in inclusion. The elements of $\mathcal{C}$ are called *clusters*. A *clustered planar embedding* of $(G, \mathcal{C})$ is a

⋆ Supported by the grant GAUK 154907.
⋆⋆ Supported by grant 1M0021620808 of the Czech Ministry of Education.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 303–314, 2007.

planar embedding emb of G together with a mapping emb_c that assigns to every cluster $C \in \mathcal{C}$ a planar region $emb_c(C)$ whose boundary is a closed Jordan curve and such that

- for each vertex $v \in V$ and every cluster $C \in \mathcal{C}$, it holds that $emb(v) \in emb_c(C)$ if and only if $v \in C$,
- for every two clusters C and D, the regions $emb_c(C)$ and $emb_c(D)$ are disjoint (in inclusion) if and only if C and D are disjoint (in inclusion, respectively), and
- for every edge $e \in E$ and every cluster $C \in \mathcal{C}$, the curve $emb(e)$ crosses the boundary of $emb_c(C)$ at most once.

The pair $(G, \mathcal{C})$ is called *clustered planar* (shortly *c-planar*) if it allows a clustered planar embedding.

It is well known that planar graphs can be recognized in polynomial (even linear) time. For c-planarity determining the time-complexity of the decision problem remains open; only partial results are known. For connected cluster graphs (i.e., when all clusters induce connected subgraphs), the problem can be solved in linear time [3]. This work was extended to "almost" connected clustered graphs in [5,6] by designing an $O(n^2)$-time algorithm. Another important step was achieved by characterization of completely connected clustered graphs (where each cluster and its complement induce connected subgraphs): A completely connected clustered graph is c-planar if and only if the underlying graph is planar [1]. Another polynomially solvable case was identified in [2] (nested triples of clusters).

We propose to study the situation when all clusters are small, which means size at most 3. In Section 2 we first remind the notion of saturators and study its meaning in the case of small clusters. As the first observation we prove that c-planarity of vertex-3-connected graphs is solvable in polynomial time (small clusters assumed). Our most general result is a polynomial time algorithm for c-planarity of Eulerian graphs that can be obtained from vertex-3-connected graphs of fixed size by cloning and subdividing edges. This algorithm is mentioned in Section 2. As can be expected, the cornerstone of this algorithm is understanding the clustered planarity in the case of a single cycle as the underlying graph, since this sheds light on particular faces of the input graph. Though at first sight this case might sound trivial, it turns out far from being obvious. Our algorithm and further useful observations are presented in Section 3. The unexpected complexity of a single cycle was also encountered by di Battista et al. in [2] when solving c-planarity for small *number* (3) of clusters. Our last result concerns the case of small *faces* rather than small clusters. In Section 5 we present a polynomial algorithm for deciding c-planarity of graphs with fixed embeddings with all faces of size at most 4. This result applies to clusters of all sizes, but only of a flat structure (i.e., when all clusters are on the same level, none being in inclusion with any other).

2 Saturators of Small Clusters

Cortese et al. introduced the following notion in [2]. A set F is a *saturator of* $(G, \mathcal{C})$ if $F \subseteq E(\overline{G})$ and for each cluster $C \in \mathcal{C}$, the vertices of C induce a connected subgraph in $G^F = (V(G), E(G) \cup F)$. The saturator F is called *planar* if G^F is planar. Note that the notion of planar saturators is slightly different than in [2]. The role of saturators is described by the following observation (stated in [2] in an equivalent formulation).

Lemma 1. *The pair $(G, \mathcal{C})$ is c-planar if and only if there exists a saturator $F = F(G, \mathcal{C})$ such that $(G^F, \mathcal{C})$ is c-planar.*

For a cluster $A \in \mathcal{C}$, we call every pair of its vertices a *cluster edge*. We say that a cluster edge e is *present in a saturator* F if e is an element of F. When it is clear from the context which saturator is considered, we omit its name and speak about *present* cluster edges only.

We further explore the meaning of saturators in certain special cases of graphs G and cluster sets $\mathcal{C}$. In Section 3, we employ this idea in reducing a special case of c-planarity to the existence of a planar saturator of $(G, \mathcal{C})$, and then further to the bipartiteness and triangle-freeness of certain auxiliary graphs.

The following corollary of Lemma 1 is a first step in reducing c-planarity to the existence of a planar saturator. It states that the existence of a planar saturator is sufficient for clusters that do not induce a cycle.

Corollary 1. *The pair $(G, \mathcal{C})$ is c-planar if and only if there exists a saturator F such that $(G^F, \mathcal{C}')$ is c-planar, where $\mathcal{C}' = \{C \in \mathcal{C} : G^F[C]$ contains a cycle$\}$.*

In this paper we mostly consider clusters of size at most three. We use the fact that if clusters are small, then there are only few possibilities of choosing present cluster edges in a saturator so that each cluster becomes connected.

A highly connected graph imposes other limitations on present cluster edges. Namely, in a fixed planar embedding of a 3-connected graph, each cluster edge can be drawn in at most one way. If we restrict ourselves both to 3-connected graphs and to clusters of size at most three, then it is possible to test c-planarity effectively. The c-planarity instance is transformed to a 2-SAT formula, where a variable represents the presence of a cluster edge, and a clause expresses a crossing.

Proposition 1. *Let G be a 3-connected graph and $\mathcal{C}$ a cluster set containing only clusters of size at most three. Then the c-planarity of $(G, \mathcal{C})$ can be decided in time $O(|\mathcal{C}|^2 \cdot |G|)$.*

3 Three-Clusters on a Cycle

Definition 1. *Let G be a cycle and $\mathcal{C}$ a set of at most three-element clusters on $V(G)$. We say that two cluster edges $\{a_1, a_2\}$ and $\{b_1, b_2\}$* conflict *if the cyclic order of their vertices is abab.*

We say that two three-vertex clusters A and $B \in \mathcal{C}$

- intersect *if the cyclic order of their vertices along G is aabbab,*
- alternate *if the cyclic order of their vertices along G is ababab.*

Given two clusters A and B we say that the vertices $a_i \in A$ and $b_j \in B$ are consecutive *if there exists a path in G from a_i to b_j that uses no other vertices of A or B.*

Lemma 2. *If G is a cycle and $\mathcal{C}$ contains only clusters of size at most three, then $(G, \mathcal{C})$ is c-planar if and only if there exists a planar saturator F.*

The proof is a straightforward case analysis. It is omitted.

3.1 Construction of Auxiliary Graphs G_1, G_1^M, and G_2

We are given a pair $(G, \mathcal{C})$, where G is a cycle and $\mathcal{C}$ contains only clusters of size at most three. According to Lemma 2, deciding the c-planarity of $(G, \mathcal{C})$ amounts to finding a planar saturator F. Thus, we need to pick suitable cluster edges for F so that the graph G^F makes every cluster connected and has a planar embedding.

Since G is a cycle, we only distinguish two ways of drawing a cluster edge in a planar embedding: inside or outside the cycle G.

Conflicts of cluster edges impose restrictions on their embedding. For two-vertex clusters, the situation is evident: each cluster edge must be drawn on one side of the cycle and any conflicting cluster edge must be drawn on the other side.

For three-vertex clusters the situation is more complicated, because we do not know in advance which cluster edges are present in the sought saturator and which are not. However, since F is a saturator, we know that every cluster C is connected in G^F. Hence, out of every pair of cluster edges of C, at least one is present in F. Thus, we consider pairs of three-vertex-cluster edges; these become vertices of an auxiliary graph G_1. Edges of two-vertex clusters will become vertices of G_1 also.

The formal construction of G_1 can be found below. There we also formalize the correspondence of vertices of G_1 and (pairs of) cluster edges. Here, for convenience, we use the notion of correspondence in an intuitive way.

For some vertex pairs x and y of G_1 the following holds: if any cluster edge corresponding to x is present, then any present cluster edge corresponding to y must be drawn on the other side of the cycle. Otherwise, a crossing would occur. Figure 1 illustrates some of those cases. We represent such a case by the edge between x and y in G_1.

We observe that if a vertex x is non-isolated in G_1, then all present cluster edges corresponding to x must be drawn on a common side of the cycle. For adjacent vertices the sides are distinct. Hence, if a bipartition of G_1 exists, it determines the drawing of all present cluster edges corresponding to non-isolated vertices (up to the choice of the inner and outer face of the cycle). Isolated vertices in G_1 are exceptional and we will not consider them to belong to any bipartity of G_1, because their corresponding cluster edges have, in a sense, more freedom.

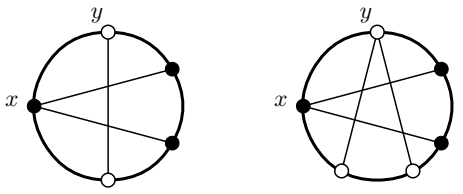


Fig. 1. Some of the situations when any cluster edge corresponding to x must be drawn on the other side of the cycle than any cluster edge corresponding to y

The graph G_1 does not capture well the restrictions caused by alternating clusters—there may be several pairwise-alternating clusters that do not give rise to any edge of G_1. Hence, we define another auxiliary graph G_2. The vertices of G_2 are three-vertex clusters, and edges $\{A, B\}$ of G_2 express that clusters A and B alternate. We later prove that, for c-planarity, there may be no triangle in G_2.

In some cases there are vertices of G_1 whose corresponding cluster edges "behave in the same way" in any planar drawing of a saturator: either all present edges are drawn outside, or all inside, or all may be drawn on both sides. In the bipartition language, such vertices must either belong to a common bipartity of G_1 in any bipartition, or they must be all isolated. We need to "unify" them. Hence, we create the graph G_1^M from G_1 by repeated merging of certain vertex tuples into groups.

The formal construction of the graphs G_1, G_2 and G_1^M follows.

Algorithm: Creation of G_1
Input: $G = (V, E)$, a set $\mathcal{C}$ of clusters
Output: the graph G_1
$V(G_1) := \{x_{A,v} : A \in \mathcal{C}, |A| = 3, v \in A\} \cup \{x_A : A \in \mathcal{C}, |A| = 2\}$
$E(G_1) := \emptyset$

1. For every two clusters $A = \{a_1, a_2\}$ and $B = \{b_1, b_2\}$ whose vertices have the cyclic order a_1, b_1, a_2, b_2, set $E(G_1) = E(G_1) \cup \{\{x_A, x_B\}\}$.
2. For every two clusters $A = \{a_1, a_2\}$ and $B = \{b_1, b_2, b_3\}$ whose vertices have the cyclic order a_1, b_1, a_2, b_2, b_3, set $E(G_1) = E(G_1) \cup \{\{x_A, x_{B,b_1}\}\}$.
3. For every two clusters $A = \{a_1, a_2, a_3\}$ and $B = \{b_1, b_2, b_3\}$ whose vertices have the cyclic order $a_1a_2b_1b_2a_3b_3$, set $E(G_1) := E(G_1) \cup \{\{x_{A,a_3}, x_{B,b_3}\}\}$.
4. For every two alternating clusters A and B such that the vertices y_A and y_B both have degree exactly one in G_2, and for every pair of their vertices $a_i \in A$ and $b_j \in B$ that are consecutive and both non-isolated in G_1, set $E(G_1) := E(G_1) \cup \{\{x_{A,a_i}, x_{B,b_j}\}\}$.

$$V(G_2) := \{y_A : A \in \mathcal{C}, |A| = 3\}$$
$$E(G_2) := \{\{y_A, y_B\} : A \text{ and } B \text{ alternate}\}.$$

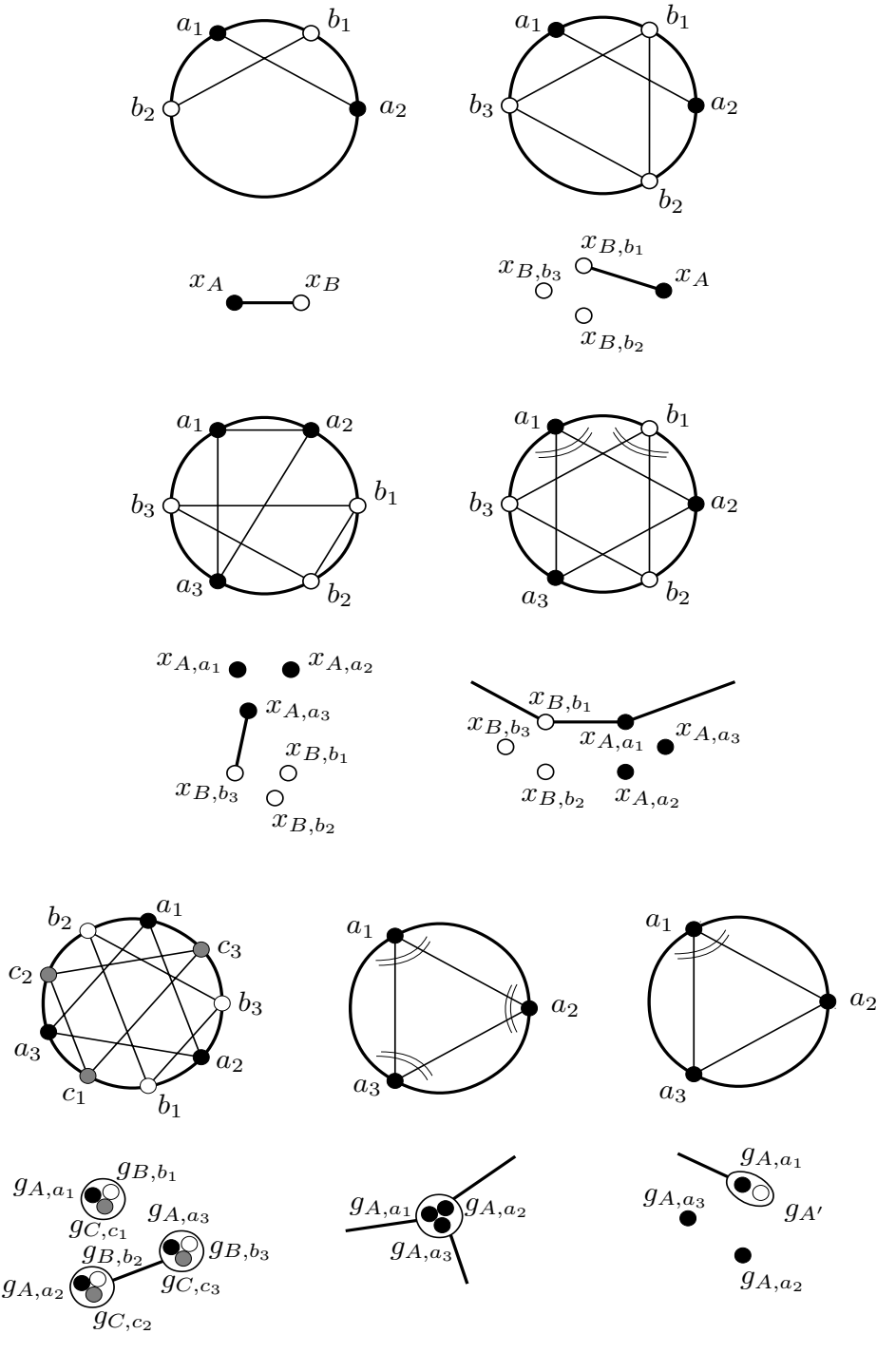


Fig. 2. Illustration of rules 1, 2, 3, 4, 5, 6, and 7

To formalize the correspondence of clusters and cluster edges with vertices of G_1 and G_2, we introduce the following definition.

Definition 2. *We say that a cluster edge* $e = \{a_i, a_j\}$ *of a cluster* A *in* G corresponds *to a vertex* v *of* G_1 *if* $v = x_{A,a_i}$ *or* $v = x_{A,a_j}$, *or if* $v = x_A$.

We say that a cluster B corresponds *to a vertex* u *of* G_2 *if* $u = y_B$.

The following algorithm creates the graph G_1^M from G_1 together with a mapping $g : V(G_1) \to V(G_1^M)$. The vertices of G_1^M will represent groups of vertices of G_1 and the mapping g will assign to each vertex the group to which it belongs. For short, we write g_{A,a_i} or g_A instead of $g(x_{A,a_i})$ or $g(x_A)$, respectively.

The algorithm starts with one-vertex groups equal to vertices of G_1 and then merges certain groups using the following procedure.

Procedure: Merge
Input: vertex groups $g_1, g_2, \ldots, g_k \in V(G_1^M)$
Output: modifies G_1^M

- Replace the groups $g_1, g_2, \ldots, g_k$ with a newly created vertex group w, and set the edges in G_1^M so that

$$N(w) = N(g_1) \cup N(g_2) \cup \cdots \cup N(g_k) \setminus \{g_1, g_2, \ldots, g_k\}.$$

- If there were two indices $1 \le i, j \le k$ (not necessarily distinct) such that g_i and g_j were adjacent then add a loop $\{w, w\}$.
- For all vertices v in $g^{-1}\big(g_1 \cup g_2 \cup \cdots \cup g_k\big)$ set $g(v) := w$.

Algorithm: Creation of G_1^M
Input: the graph G_1
Output: the graph G_1^M, a mapping $g : V(G_1) \to V(G_1^M)$
$G_1^M := G_1$, $g :=$ id

5. For each cluster $A = \{a_1, a_2, a_3\}$ which alternates with at least two other clusters $B = \{b_1, b_2, b_3\}$ and $C = \{c_1, c_2, c_3\}$ in the way a_1, c_3, b_3, a_2, b_1, c_1, a_3, c_2, b_2, do merge$(g_{A,a_1}, g_{B,b_1}, g_{C,c_1})$, merge$(g_{A,a_2}, g_{B,b_2}, g_{C,c_2})$, and merge$(g_{A,a_3}, g_{B,b_3}, g_{C,c_3})$.
6. For every three-vertex cluster A having all corresponding vertices g_{A,a_i} non-isolated in G_1^M, do merge$(g_{A,a_1}, g_{A,a_2}, g_{A,a_3})$.
7. For every two clusters $A' = \{a_1, a_2\}$ and $A = \{a_1, a_2, a_3\}$ such that g_{A,a_1} is not isolated in G_1^M, do merge$(g_{A,a_1}, g_{A'})$.

Having created all the auxiliary graphs G_1, G_1^M and G_2, it is easy to decide if the input pair $(G, \mathcal{C})$ is c-planar, as stated in Theorem 1. Before stating the theorem, we present several auxiliary lemmas.

Lemma 3. *Let* F *be a planar saturator. Then in* G^F, *the following is true:*

1. *if there is an edge between the vertices x and y in G_1, then any present cluster edge corresponding to x is drawn inside the cycle G, and any present cluster edge corresponding to y is drawn outside the cycle G, or vice versa.*
2. *the present cluster edges corresponding to vertices in $g^{-1}(v)$ such that v is non-isolated in G_1^M are drawn either all inside or all outside the cycle G.*
3. *if there is an edge between the vertices x and y in G_1^M then any present cluster edge corresponding to $g^{-1}(x)$ is drawn inside the cycle G, and any present cluster edge corresponding to $g^{-1}(y)$ is drawn outside the cycle G, or vice versa.*

Proof. We follow the creation of G_1 and prove that the first part of the Lemma holds after every step. Before any rule is applied, there are no edges in G_1 and it holds trivially. Then a step according to rule 1, 2, or 3 adds one new edge, say, xy. For all these rules, it is not hard to see that if an edge corresponding to x and an edge corresponding to y are drawn on the same side of the cycle G, then they cross each other. Hence, after a step 1, 2, or 3, the first part remains valid.

Let us consider a step according to rule 4. We use the same notation as in the description of this step, so we have two clusters A and B, and let $x = x_{A,a_1}$ and $y = x_{B,b_1}$. Note that by definition of rule 4, the vertices x and y are already non-isolated. Thus all present cluster edges corresponding to x must be drawn on the same side of the cycle, and the same holds for y.

Assume for contradiction that there are cluster edges corresponding to x and y both inside the cycle G. Then it must be edges a_1a_3 and b_1b_2, because they are the only pair without a crossing. By the above argument, the edge a_1a_2 can only be drawn on the same side as a_1a_3; but that is not possible because of b_1b_2. So a_1a_2 is not present. Then a_2a_3 is present and drawn outside. Similarly, b_1b_3 is not present, and there is no way to draw b_2b_3—a contradiction.

The second part is proved by a straightforward case analysis of the algorithm steps. The third part is an easy consequence of the first two.

Lemma 4. *If $(G, \mathcal{C})$ contains three pairwise alternating clusters, then $(G, \mathcal{C})$ is not c-planar.*

Proof. Let A, B, and C be the three clusters, and assume for contradiction that $(G, \mathcal{C})$ is c-planar. We use Lemma 2 to do a straightforward case analysis of the cluster edges present in a planar drawing. If present cluster edges of A are all drawn on the same side of the cycle, then present cluster edges of B must be drawn on the other side, and there is no way to draw at least two cluster edges of C without crossing. The other possibility is that A has one cluster edge drawn inside and one outside the cycle. Then the same holds for B, and again, there is no way to draw C.

Lemma 5. *If G has a planar saturator, then G_1^M is bipartite and G_2 is triangle-free.*

Proof. First assume that G_1^M is not bipartite. Then it contains an odd cycle C, $V(C) = \{v_1, v_2, v_3 ... v_{2k+1}\}$, $0 \leq k$. Without loss of generality we can assume that

cluster edges corresponding to vertices in $g^{-1}(v_1)$ are drawn inside the cycle. By Lemma 3 we know that $g^{-1}(v_2)$ is outside, by the same argument $g^{-1}(v_3)$ is inside etc. But then both $g^{-1}(v_1)$ and $g^{-1}(v_{2k+1})$ are drawn inside, which is not possible again by Lemma 3, a contradiction.

If G_2 contains a triangle, then there are three pairwise alternating clusters and the instance is not c-planar by Lemma 4.

Lemma 6. *If G_2 is triangle-free and G_1^M is bipartite, then $(G, \mathcal{C})$ has a planar saturator F.*

Proof. Let I be the set of isolated vertices of G_1^M. Let us fix a drawing of the cycle G into the plane and some bipartition of $G_1^M \setminus I$ for the rest of this section. As G is a cycle, any its drawing has well-defined inner and outer face, so drawing an edge of G^F inside or outside of the cycle is well defined. The idea behind our drawing is that edges represented by non-isolated vertices of G_1^M in the first part are drawn inside the cycle and edges in the second part are drawn outside the cycle. Vertices of I do not impose any restriction and therefore the edges represented by them can be drawn both inside or outside the cycle.

The rest of the proof, which is a long and technical case analysis, is omitted.

Theorem 1. *Let G be a cycle, let $\mathcal{C}$ contain only clusters of size at most three, and let G_1^M and G_2 be the graphs constructed for $(G, \mathcal{C})$ using the algorithms above. Then the pair $(G, \mathcal{C})$ is c-planar if and only if G_1^M is bipartite and G_2 is triangle-free.*

Proof. By Lemma 2, a pair $(G, \mathcal{C})$ satisfying the assumptions is c-planar if and only if it has a planar saturator. Lemmas 5 and 6 provide the rest of the proof.

Corollary 2. *Let G be a cycle and let $\mathcal{C}$ contain only clusters of size at most three. Then the c-planarity of $(G, \mathcal{C})$ can be decided in time $O(|V(G)| + |\mathcal{C}|^3)$.*

4 Three-Clusters on Rib-Eulerian Graphs

As a generalization of the previous section, in this section we mention the algorithm for c-planarity of a special subclass of Eulerian graphs. Let k be a constant; we call a graph *k-Rib-Eulerian* if it is Eulerian, and if it can be obtained from a 3-connected graph on k vertices by multiplying some edges, and then subdividing some edges. Figure 3 gives an example of such a graph.

We say that a path whose inner vertices have degree two and the outer vertices have degree larger than two is a *rib*. Thus a k-Rib-Eulerian graph consists of k vertices of degree at least four that are interconnected by ribs. A vertex of degree at least four is called a *branching vertex*. A cluster is called a *branch cluster* if it contains a branching vertex, and *non-branch cluster* otherwise.

We want to decide the c-planarity of a pair $(G, \mathcal{C})$, where G is a Rib-Eulerian graph. First, we deal with the branch clusters. We try all the possibilities of choosing saturator edges to those clusters, we add each chosen edge to G twice (in order not to break its Eulericity) and we run the rest of the algorithm for

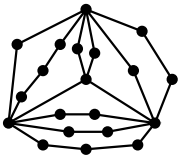


Fig. 3. Example of a Rib-Eulerian graph created from K_4

each of the posibilities. Clearly, the pair $(G, \mathcal{C})$ is c-planar if and only if it is c-planar for at least one of the saturator choices. Moreover there are constantly many choices to check, since there can only be $O(k)$ branch clusters in a k-Rib-Eulerian graph and k is a constant. Hence the rest of the algorithm sketched bellow only deals with non-branch clusters.

As a next step, we seek a suitable planar embedding of G. The planar embedding of the underlying 3-connected graph is unique (up to the choice of the outer face). Hence our main issue is to find the order of ribs originating from a common edge of the underlying graph. This is done with respect to clusters in $\mathcal{C}$, because they force adjacencies of certain ribs.

When a suitable planar embedding of G is obtained, we utilize the algorithm of Section 3 that deals with cycles. In a planar embedding of a Rib-Eulerian graph, the boundary of each face is a cycle. All the restrictions for cluster edges on a cycle apply in this case as well. And more of them appear, because in this case, "the outside" of faces is more complex. Basically, we create the auxiliary graphs G_1 and G_1^M for the whole graph at once, applying the rules from the previous section on (parts of) clusters lying in the same face. The graph G_2 is created for each face separately. We then reduce the c-planarity of $(G, \mathcal{C})$ to the bipartiteness of the graph G_1^M and the triangle-freeness of the graphs G_2 for each face.

Due to space limitations, details of the algorithm and the proof of the correctness and running time are omitted. We just mention the summarizing theorem.

Theorem 2. *The c-planarity of $(G, \mathcal{C})$ can be decided in time $O(3^k \cdot n^3)$ for G being k-Rib-Eulerian with n vertices and $\mathcal{C}$ containing clusters of size at most 3.*

5 Clustered Planarity on Graphs with Small Faces

In this section, we show that the c-planarity problem can be solved in polynomial time for 3-connected graphs with faces of size at most 4 and with cluster sets where every two clusters are disjoint. Thus in this section we always assume the graph G is 3-connected with faces of size at most 4. First, we define the notion of extended graph of a cluster A. Informally, the extended graph contains all cluster edges that can be added to the drawing of G and can help connecting A.

Definition 3. *Let $G = (V, E)$ be a plane graph and $\mathcal{C}$ a cluster set. For a cluster $A \in \mathcal{C}$ we define the* extended graph of A*, called $G_E(A)$, on the set of vertices A. Two vertices u, v in $G_E(A)$ are adjacent if and only if either $\{u, v\} \in E$ or u and v are in the same face of G and in different connected components of $G[A]$.*

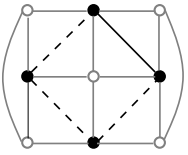


Fig. 4. The extended graph of a cluster A in its unique embedding. Vertices of A are depicted as solid black disks, other vertices of G are depicted as grey circles. Edges of G are drawn as solid (either grey or black) lines, edges of $G_E(A)$ are drawn as black lines (dashed in case the edge is not in G).

As seen in Figure 4, the drawing of G defines drawings of extended graphs:

Lemma 7. *Let $(G, \mathcal{C})$ be as above. Then for each $A \in \mathcal{C}$ the graph $G_E(A)$ is planar and its (unique) planar embedding is defined by the embedding of G.*

The key idea of our approach is to examine the drawing of the extended graph $G_E(A)$ while forgetting about the edges of G itself. We continue by presenting two lemmas about extended graphs.

Lemma 8. *Let G be a 3-connected graph with each face of size at most four, $\mathcal{C}$ be a cluster set, $A \in \mathcal{C}$ and $G_E(A)$ as above. Let f be a face of $G_E(A)$, $B \in \mathcal{C}$ and B' be the vertices of B inside face f. Then for every planar saturator F of $(G, \mathcal{C})$, the graph $G^F[B']$ is connected.*

The following lemma uses the notion of labeled dual of $G_E(A)$. Let $\mathcal{C}$ be a cluster set such that every two clusters are disjoint. A *labeled dual* of $G_E(A)$, called $G_E^d(A)$, is a multigraph whose vertices are faces of $G_E(A)$. Edges of $G_E^d(A)$ are labeled by clusters of $\mathcal{C}$, and $G_E^d(A)$ contains an edge between f and f' labeled by B if f, f' are adjacent and cluster B has vertices inside both these faces. See Figure 5 for an illustration of labeled dual.

Lemma 9. *Let G be as above, and $\mathcal{C}$ a cluster set such that every two clusters in $\mathcal{C}$ are disjoint. If $G_E^d(A)$ has a cycle whose edges have at least two different labels for some $A \in \mathcal{C}$, then $(G, \mathcal{C})$ is not c-planar.*

Now we are ready to show our main theorem.

Theorem 3. *The c-planarity of $(G, \mathcal{C})$, where G is a 3-connected planar graph with faces of size at most 4 and $\mathcal{C}$ is a cluster set such that every two clusters are disjoint, can be decided in time $O(|V(G)|^2)$.*

Proof. (sketch) The idea of the proof is as follows: As every 3-connected graph has a unique embedding into the plane (up to the choice of the outer face), we can assume G has a fixed embedding. We pick arbitrary $A \in \mathcal{C}$. By Lemma 8 in each face f of $G_E(A)$, each cluster has to be connected by any saturator. Thus in each face, we solve a small c-planarity problem. Then we connect parts of clusters in different faces greedily and by planar-duality argument we show that if there was no cycle using two labels in $G_E^d(A)$ (otherwise by Lemma 9 $(G, \mathcal{C})$ is not c-planar), the cluster A can be still connected.

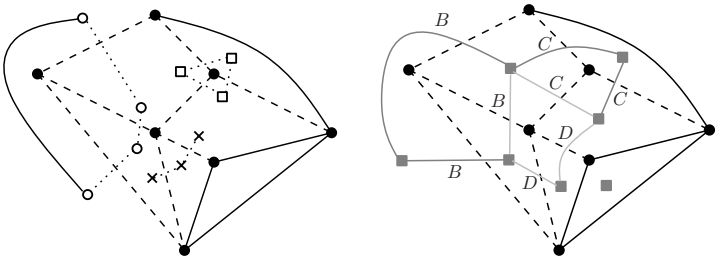


Fig. 5. Left: A graph $G_E(A)$ (discs) together with extended graphs of clusters B, C and D (circles, boxes, crosses). Right: A graph $G_E(A)$ (black) with its labeled dual $G_E^d(A)$ (grey) containing a cycle (light grey).

Acknowledgement

The authors would like to thank Tomáš Chudlarský for valuable discussions.

References

1. Cornelsen, S., Wagner, D.: Completely Connected Clustered Graphs. Journal of Discrete Algorithms 4(2), 313–323 (2006)
2. Cortese, P.F., Di Battista, G., Patrignani, M., Pizzonia, M.: Clustering Cycles into Cycles of Clusters. Journal of Graph Algorithms and Applications, Special Issue on the 2004 Symposium on Graph Drawing, GD 2004 9(3), 391–413 (2005)
3. Dahlhaus, E.: Linear time algorithm to recognize clustered planar graphs and its parallelization. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 239–248. Springer, Heidelberg (1998)
4. Feng, Q.W., Cohen, R.F., Eades, P.: Planarity for clustered graphs. In: Spirakis, P.G. (ed.) ESA 1995. LNCS, vol. 979, pp. 213–226. Springer, Heidelberg (1995)
5. Gutwenger, C., Jünger, M., Leipert, S., Mutzel, P., Percan, M., Weiskircher, R.: C-planarity testing of clustered graphs. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 220–235. Springer, Heidelberg (2002)
6. Gutwenger, C., Jünger, M., Leipert, S., Mutzel, P., Percan, M., Weiskircher, R.: Subgraph induces planar connectivity augmentation. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 261–272. Springer, Heidelberg (2003)

Drawing Colored Graphs with Constrained Vertex Positions and Few Bends per Edge*

Emilio Di Giacomo, Giuseppe Liotta, and Francesco Trotta

Dip. di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia
{digiacomo,liotta,francesco.trotta}@diei.unipg.it

Abstract. Hamiltonicity, book embeddability, and point-set embeddability of planar graphs are strictly related concepts. We exploit the interplay between these notions to describe colored sets of points and to design polynomial-time algorithms to embed k-colored planar graphs on these sets such that the resulting drawings have $\mathcal{O}(k)$ bends per edge.

1 Introduction

Let G be a planar graph with n vertices whose vertex set is partitioned into subsets $V_0, \ldots, V_{k-1}$ for some positive integer $1 \leq k \leq n$ and let S be a set of n distinct points in the plane partitioned into subsets $S_0, \ldots, S_{k-1}$ with $|V_i| = |S_i|$ $(0 \leq i \leq k-1)$. We say that each index i is a color, G is a k-colored planar graph, and S is a k-colored set of points compatible with G. This paper studies the problem of computing a k-colored point-set embedding of G on S, i.e. a crossing-free drawing of G such that each vertex of V_i is mapped to a distinct point of S_i. The problem has received considerable interest in the literature(see, e.g., [1,3,5,6,8,9]), also motivated by the observation that these types of drawings naturally model *semantic constraints* about the placement of the vertices. Particular attention has been devoted to the curve complexity of the computed drawings, i.e. the maximum number of bends along each edge. Namely, reducing the number of bends along the edges is a fundamental optimization goal when computing aesthetically pleasing drawings of graphs (see, e.g., [2,7]).

Two key references about k-colored point-set embeddings are the works by Kaufmann and Wiese [8] and by Pach and Wenger [9]. Kaufmann and Wiese [8] study the monochromatic version of the problem (i.e. the case when $k = 1$) and prove that a planar graph with n vertices always admits a point-set embedding with at most two bends per edge on *any* set of n distinct points in the plane; they also proved that two bends per edge are necessary for some planar graphs and some configurations of points. Pach and Wenger [9] study the n-chromatic version of the problem and show that a linear number of bends is always sufficient to compute an n-colored point-set embedding of an n-colored planar graph G on any n-colored set of points compatible with G; also they show that $\Omega(n)$

* Research partially supported by the MIUR Project "MAINSTREAM: Algorithms for massive information structures and data streams".

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 315–326, 2007.

bends per edge may be necessary even for n-colored paths when the points are in convex position.

The gap between constant curve complexity for $k = 1$ and linear curve complexity for $k = n$ motivates the study of other values of k. In [5] it has been proved that there exists a 2-colored planar graph G and a 2-colored set of points S compatible with G such that any 2-colored points set embedding of G on S has at least one edge with $\Omega(n)$ bends. This result has been extended in [1], where it is proved that for k-colored point set embeddings such that $3 \leq k \leq n$, there may be cases requiring $\Omega(n)$ bends on $\Omega(n)$ edges. The two counterexamples presented in [5] and [1] are either tri-connected or have outerplanarity $\mathcal{O}(n)$, and thus a natural research direction is concerned with the curve complexity of k-colored point set embeddings for (sub)-families of planar graphs that have a simpler structure. In [3] it is proved that the curve complexity of 3-colored point-set embeddings may not be constant even for 3-colored outerplanar graphs.

These negative results suggest two different research directions, both devoted to studying k-colored point-set embeddings with curve complexity that does not depend on the input size. From one side, instead of restricting the classes of graphs to be drawn one can focus on special configurations of k-colored sets of points that make it possible to compute k-colored point-set embeddings with constant curve complexity for *any* k-colored planar graph. On the other side, one can ask what is the size of a universal k-colored set of points that guarantees curve complexity independent of n for any k-colored planar graph. This last question can be asked both in the case that the points have real coordinates or by restricting them to form an integer grid. The main results in this paper can be outlined as follows.

- We study a special type of k-colored point-sets. Namely, let G be any k-colored planar graph with n vertices ($2 \leq k \leq n$) and let S be a k-colored set of n points compatible with G. We show that if S is *ordered*, i.e. for each color all points of that color are consecutive along the x-direction, then there exists an $\mathcal{O}(n \log n + k\ n)$-time algorithm that computes a k-colored point-set embedding of G on S with curve complexity at most $3k + 7$. This result generalizes to all k-colored planar graphs a similar result presented in [3] for k-colored outerplanar graphs and makes it possible to improve a related result of [1].
- We show the existence of k-colored sets of points having linear size and supporting k-colored point-set embeddings of $\mathcal{O}(k)$ curve complexity. Namely, let $\mathcal{F}_k$ be the family of all k-colored planar graphs with n vertices ($1 \leq k \leq n$). For any $G \in \mathcal{F}_k$ and for any k-colored set of points S such that S contains $k\ n - k^2 + 1$ points for each color, there exists an $\mathcal{O}(n \log n + k\ n)$-time algorithm that computes a k-colored point-set embedding of G on S with curve complexity at most $3k + 7$. We recall that, even for 2-colored simple paths, a universal 2-colored set of points that supports straight-line 2-colored point-set embeddings may need a quadratic number of points [6].
- Since the above result implies a total number of $k^2 n - k^3 + k$ points in S, one can ask whether $n + o(n)$ points are sufficient to guarantee a curve complexity that does not depend on n. We give a negative answer to this question for $k = 2$.

Namely, let c be any constant such that $c > 1$. We prove that for $k = 2$ there exists a set S of $n + \frac{n}{c}$ points and a 2-colored planar graph G such that any 2-colored point set embedding of G on S has an edge requiring at least $\Omega(n)$ bends.

• Finally, we show that every k-colored planar graph with n vertices admits a k-colored points-set embedding with curve complexity $6k + 5$ on a k-colored grid whose size is $\mathcal{O}(k\ n^2) \times \mathcal{O}(k\ n^2)$. Such k-colored points-set embedding can be computed in $\mathcal{O}(k\ n)$ time.

The above results are all based on a novel approach to the problem of computing k-colored point-set embeddings of planar graphs. Namely we exploit the notion of *simultaneous k-colored book embedding* of a k-colored planar graph and a k-colored path and show how this notion can be used to compute a suitable Hamiltonian circuit on the graph; in turn, we use the Hamiltonian circuit to compute a point-set embedding with $\mathcal{O}(k)$ curve complexity. For reasons of space some proofs have been sketched or omitted.

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*. In the rest of this section the index i is $0 \leq i \leq k-1$ if not differently specified. For each vertex $v \in V_i$ we denote by $\mathrm{col}(v)$ the color i of v. A graph G with a k-coloring is called a *k-colored graph*. Let S be a set of distinct points in the plane. We always assume that the points of S have distinct x-coordinates (this condition can always be satisfied by means of a suitable rotation of the plane). For any point p in the Euclidean plane 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 S of distinct points in the plane with a k-coloring is called a *k-colored set of points*. For each point $p \in S_i$ $\mathrm{col}(p)$ denotes the color i of p. A k-colored set of points S is *compatible with* a k-colored graph G if $|V_i| = |S_i|$ for every i. Let G be planar. We say that G has a *k-colored 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 $\mathrm{col}(p) = \mathrm{col}(v)$, (ii) each edge e of G is drawn as a polyline; a point shared by any two consecutive segments of the polyline is called a *bend* of e. The *curve complexity* of a drawing is the maximum number of bends per edge. Given a vertex v of G we denote by p_v the point representing v in the drawing. A *k-colored sequence* σ is a linear sequence of (possibly repeated) colors c_0, c_1, …, c_{n-1} such that $0 \leq c_j \leq k-1$ $(0 \leq j \leq n-1)$. We say that σ is *compatible with* a k-colored graph G if, for every i color i occurs $|V_i|$ times in σ. Let S be a k-colored set of points and let $p_0, p_1, \ldots, p_{n-1}$ be the points of S ordered according to their x-coordinates. Let $P = (v_0, v_1, \ldots, v_{n-1})$ be a path with n vertices such that $c(v_i) = c(p_i)$. We say that P is the *path induced by S* and denote it as $\mathsf{path}(S)$. We also say that $\sigma = c(p_0), c(p_1), \ldots, c(p_{n-1})$ is the *k-colored sequence induced by S* and denote it as $\mathsf{seq}(S)$.

A graph G has a *Hamiltonian path* if it has a simple path that contains all the vertices of G. G has a *Hamiltonian cycle* if it has a simple cycle that contains all

the vertices of G. If G is a k-colored graph and $\sigma = c_0, c_1, \ldots, c_{n-1}$ is a k-colored sequence compatible with G, a *k-colored Hamiltonian path of G consistent with σ* is a Hamiltonian path $v_0, v_1, \ldots, v_{n-1}$ such that $col(v_i) = c_i$ $(0 \leq i \leq n-1)$. A *k-colored Hamiltonian cycle of G consistent with σ* is a Hamiltonian cycle $v_0, v_1, \ldots, v_{n-1}$ such that $col(v_i) = c_i$ $(0 \leq i \leq n-1)$. A k-colored planar graph G can always be augmented to a (not necessarily planar) k-colored graph G' by adding to G a suitable number of dummy edges and such that G' has a k-colored Hamiltonian cycle $\mathcal{C}$ consistent with σ and that includes all dummy edges. If G' is not planar, we can apply a planarization algorithm (see, e.g., [2]) to G' with the constraint that only crossings between dummy edges and edges of $G - \mathcal{C}$ are allowed. Such a planarization algorithm constructs an embedded planar graph G'' where each edge crossing is replaced with a dummy vertex, called *division vertex*. By this procedure each edge e of $\mathcal{C}$ can be transformed into a path whose internal vertices are division vertices: let $\mathcal{C}'$ be the resulting cycle. Let e be an edge of $\mathcal{C}'$ (notice that the endvertices of e are either vertices of G or division vertices). The path $\mathcal{H} = \mathcal{C}' \setminus e$ is called an *augmenting k-colored Hamiltonian path of G consistent with σ*. The graph $G'' \setminus e$ is called the *augmented Hamiltonian form of G* and is denoted as $\mathsf{Ham}(G)$. If every edge e of G is crossed at most d times in G' (which implies that e is split by at most d division vertices in $\mathsf{Ham}(G)$), $\mathcal{H}$ is said to be an *augmenting k-colored Hamiltonian path of G consistent with σ and inducing at most d division vertices per edge*. If G' is planar, then $\mathsf{Ham}(G) = G'$ and $\mathcal{H}$ is defined as $\mathcal{C} \setminus e$, where e is any edge of $\mathcal{C}$. Notice that the endvertices of $\mathcal{H}$ are on the same face f of $\mathsf{Ham}(G)$; we may assume that f is the external face (if not we can choose an embedding of $\mathsf{Ham}(G)$ such that f is the external face).

Let v_d be a division vertex for an edge e of G. Since a division vertex corresponds to a crossing between e and an edge of $\mathcal{C}$, there are four edges incident on v_d in G''; two of them are dummy edges that belong to $\mathcal{C}'$, the other two are two "pieces" of edge e obtained by splitting e with v_d. Let (u, v_d) and (v, v_d) be the latter two edges. We say that v_d is a *flat division vertex* if it is encountered after u and before v while walking along $\mathcal{H}$; v_d is a *pointy division vertex* otherwise. Notice that there are exactly four edges incident on v_d in G'', but there can be only three edges incident on v_d in $\mathsf{Ham}(G)$ (this happens if the edge removed from G'' to obtain $\mathsf{Ham}(G)$ has v_d as an endvertex, i.e. if v_d is one of the two endvertices of $\mathcal{H}$). However the edge incident on v_d that is removed is neither (u, v_d), nor (v, v_d) because the removed edge is an edge of $\mathcal{C}'$. It follows that the definition of flat and pointy division vertex apply to v_d also in the case when v_d is an endvertex of $\mathcal{H}$. The following theorem has been proved in [1].

Theorem 1. [1] *Let G be a k-colored planar graph, let σ be a k-colored sequence compatible with G, and let $\mathcal{H}$ be an augmenting k-colored Hamiltonian path of G compatible with σ inducing at most d division vertices per edge, d_p of which are pointy division vertices. Then G admits a k-colored point set embedding Γ on any set of points that induces σ such that the curve complexity is $d + d_p + 1$. Furthermore, there exists an $\mathcal{O}(n \log n)$-time algorithm that computes Γ.*

A *spine* is an horizontal line. Let ℓ be a spine and let p, q be two points of ℓ. An *arc* is a circular arc passing through p and q. We say that the arc is in the *top* (*bottom*) *page* if it belongs to the half plane above (below) the spine. Let $G = (V, E)$ be a planar graph. A *topological book embedding* of G is a planar drawing such that all vertices of G are represented as points of a spine ℓ and each edge can be either above the spine, or below the spine, or it can cross the spine. Each crossing between an edge and the spine is called a *spine crossing*. It is also assumed that in a topological book embedding every edge consists of one or more arcs such that no two consecutive arcs are in the same page. An edge e is said to be in the top (bottom) page of the spine if it consists of exactly one arc and this arc is in the top (bottom) page. A *monotone topological book embedding* is a topological book embedding such that each edge crosses the spine at most once. Also, let $e = (u, v)$ be an edge of a monotone topological book embedding that crosses the spine at a point p; e is such that if u precedes v in the left-to-right order along the spine then p is between u and v, the arc with endpoints u and p is in the bottom page, and the arc with endpoints u and v is in the top page. The edges that do not cross the spine are called *u-shaped* edges, while edges that cross the spine are called *s-shaped* edges. The following theorem has been proved in [4].

Theorem 2. [4] *Every planar graph admits a monotone topological book embedding. Also, a monotone topological book embedding can be computed in $\mathcal{O}(n)$ time, where n is the number of the vertices in the graph.*

Let Γ be a topological book emebedding of a planar graph G. A point p of the spine ℓ of Γ is *accessible from the top (bottom) page* of Γ if the vertical half-line ℓ' starting at p that is in the half-plane above (below) ℓ does not cross any arc of Γ. If ℓ' crosses an arc a, we say that *a covers p*. The *local top (bottom) page width of Γ on p* $\mathsf{lw}_{\mathrm{t}}(\Gamma, p)$ ($\mathsf{lw}_{\mathrm{b}}(\Gamma, p)$) is the number of arcs in the top (bottom) page of Γ that cover p. The *cumulative local page width of Γ on p* is $\mathsf{clw}(\Gamma, p) = \mathsf{lw}_{\mathrm{t}}(\Gamma, p) + \mathsf{lw}_{\mathrm{b}}(\Gamma, p)$. The *cumulative width of Γ* is $\mathsf{cw}(\Gamma) = max_{p \in l}\{\mathsf{clw}(\Gamma, p)\}$. The *top page width of Γ* is $\mathsf{w}_{\mathrm{t}}(\Gamma) = max_{p \in l}\{\mathsf{lw}_{\mathrm{t}}(\Gamma, p)\}$, and analogously the *bottom page width of Γ* is $\mathsf{w}_{\mathrm{b}}(\Gamma) = max_{p \in l}\{\mathsf{lw}_{\mathrm{b}}(\Gamma, p)\}$. Finally the *width of Γ* is $\mathsf{w}(\Gamma) = max\{\mathsf{w}_{\mathrm{t}}(\Gamma), \mathsf{w}_{\mathrm{b}}(\Gamma)\}$. Notice that the following inequality is always satisfied $\mathsf{w}(\Gamma) \leq \mathsf{cw}(\Gamma) \leq \mathsf{w}_{\mathrm{t}}(\Gamma) + \mathsf{w}_{\mathrm{b}}(\Gamma) \leq 2\,\mathsf{w}(\Gamma)$.

3 Overview of the Approach

In this section we give a high-level description of the approach followed throughout the paper. We need some additional definitions. Let Γ be a topological book embedding of a k-colored planar graph G and let $v_0, v_1, \ldots, v_{n-1}$ be the vertices of G in the order they appear along the spine of Γ. The k-colored sequence $c(v_0), c(v_1), \ldots, c(v_{n-1})$ is called the *k-colored sequence induced by Γ*. Let P be a path and let $v_0, v_1, \ldots, v_{n-1}$ be the vertices of P in the order they appear along P; the k-colored sequence $c(v_0), c(v_1), \ldots, c(v_{n-1})$ is called the *k-colored sequence induced by P*. Let Γ_P be a topological book embedding of P. Γ_P is

external if both the endvertices of P are accessible either from the top page or from the bottom page. Let $G_1 = (\bigcup_{i=0}^{k-1} V_{1,i}, E_1)$ and $G_2 = (\bigcup_{i=0}^{k-1} V_{2,i}, E_2)$ be two planar k-colored graphs. We say that G_1 and G_2 are *compatible* if $|V_{1,i}| = |V_{2,i}|$ for every $i = 0, \ldots, k-1$. A *simultaneous k-colored book embedding* of two compatible planar graphs G_1 and G_2 is a pair of drawings $< \Gamma_1, \Gamma_2 >$ such that: (i) Γ_i is a topological book embedding ($i = 1, 2$); (ii) Γ_1 and Γ_2 use the same points to represent the vertices; and (iii) the k-colored sequences induced by Γ_1 and Γ_2 coincide. We are now ready to describe our approach:

Step 1: Let $P = \mathsf{path}(S)$. Compute a simultaneous k-colored book embedding $< \Gamma_G, \Gamma_P >$ of G and P, such that Γ_G is a monotone topological book embedding of G and Γ_P is external;

Step 2: By using $< \Gamma_G, \Gamma_P >$, compute a k-colored point-set embedding of G on S. The curve complexity of the computed drawing is bounded by the width of Γ_P.

The idea behind the above described approach is based on Theorem 1 and on the following lemma, that shows how to compute an augmenting k-colored Hamiltonian path by using $< \Gamma_G, \Gamma_P >$. An illustration of such an idea is shown in Figure 1.

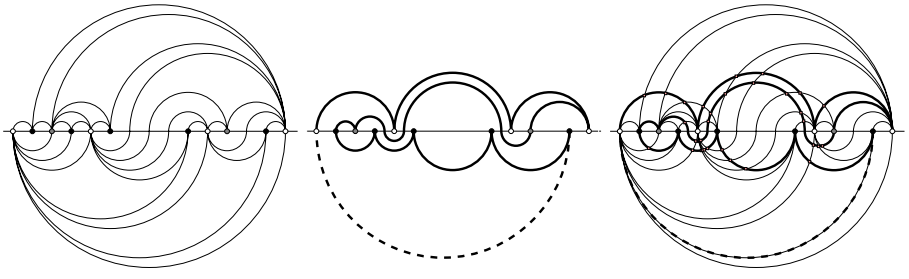


Fig. 1. (a) A monotone topological book embedding Γ_G of a planar 3-colored graph G. (b) An external topological book embedding Γ_P of a path P. Notice that $< \Gamma_G, \Gamma_P >$ is a simultaneous k-colored book embedding of G and P and that $\mathsf{w}(\Gamma_P) = 3$. (c) An augmenting k-colored Hamiltonian path of G consistent with the k-colored sequence induced by P and inducing at most 5 division vertices per edge.

Lemma 1. *Let G and P be a k-colored planar graph and a k-colored path that are compatible. Let $< \Gamma_G, \Gamma_P >$ be a simultaneous k-colored book embedding of G and P, such that Γ_G is a monotone topological book embedding and Γ_P is external. Let σ be the k-colored sequence induced by P. Then G admits an augmenting k-colored Hamiltonian path consistent with σ that induces at most $2\,\mathsf{w}(\Gamma_P) + \mathsf{cw}(\Gamma_P) + 2$ division vertices for every s-shaped edge of Γ_G and at most $2\,\mathsf{w}(\Gamma_P) + 1$ division vertices for every u-shaped edge of Γ_G.*

Sketch of Proof: Consider the simultaneous k-colored book embedding $< \Gamma_G, \Gamma_P >$. Since Γ_P is external the two endvertices u and v of P are accessible either from the top page or from the bottom page. Vertices u and v can be connected by means of an edge e' consisting of at most two arcs. Namely, if they are both accessible from the same page, say the top one, then we connect them with an arc in the top page; if they are accessible from different pages, assume that u is accessible from the top page and v is accessible from the bottom page (the other case is symmetric). We create an arc connecting v to a point p of the spine that is to the right of the rightmost point (vertex or spine crossing) of Γ_P; then we add an arc connecting p to u. $\Gamma' = \Gamma_G \cup \Gamma_P \cup e'$ is a (possibly non-planar) drawing of a (possibly non-planar) graph G' such that $\mathcal{C} = P \cup e'$ is a k-colored Hamiltonian cycle consistent with the k-colored sequence σ induced by P. If G' is not planar, since both Γ_G and Γ_P are planar, a crossing in Γ' is possible only between the edges of Γ_G and the edges of Γ_P. We replace each crossing with a division vertex. It may happen that edge e' is also subdivided (this happens if the endvertices of P are not on the same face of Γ_G). In all cases, there exists a portion of e' that is contained in a face of Γ_G. The graph obtained by removing this portion (possibly coincident with e' itself) is the augmented Hamiltonian form of G. The concatenation of P with the portion of e' that is not removed forms an augmenting k-colored Hamiltonian path of G consistent with the k-colored sequence induced by P. In order to compute the number of division vertices on each edge of G, we first count the number of crossings between an edge of G and the edges of P, and then we count the extra division vertices introduced when adding edge e'. Let $e = (u, v)$ be an u-shaped edge of Γ_G. Assume that e is in the top page of Γ_G. The number of crossings between e and the edges of P is $c = \mathsf{lw}_{\mathrm{t}}(\Gamma_P, p_u) + \mathsf{lw}_{\mathrm{t}}(\Gamma_P, p_v) \leq 2\,\mathsf{w}(\Gamma_P)$. Let $e = (u, v)$ be an s-shaped edge and let a_1 and a_2 be the two arcs that form e. Arc a_1 has p_u and d as its endpoints, where d is the point where e crosses the spine. Arc a_2 has d and p_v as its endpoints. The number of crossings between e and the edges of P is $c = \mathsf{lw}_{\mathrm{t}}(\Gamma_P, p_u) + \mathsf{clw}(\Gamma_P, d) + \mathsf{lw}_{\mathrm{b}}(\Gamma_P, p_v) \leq \mathsf{w}(\Gamma_P) + \mathsf{cw}(\Gamma_P) + \mathsf{w}(\Gamma_P) = 2\,\mathsf{w}(\Gamma_P) + \mathsf{cw}(\Gamma_P)$. Since e' consists of at most two arcs in different pages, each arc of Γ_G can have one additional division vertex caused by the addition of e'. Therefore an u-shaped edge (s-shaped edge) can have at most $2\,\mathsf{w}(\Gamma_P) + 1$ ($2\,\mathsf{w}(\Gamma_P) + \mathsf{cw}(\Gamma_P) + 2$) division vertices per edge. □

4 Ordered k-Colorings

Let S be a k-colored set of points such that for every pair of points p_1 and p_2 with the same color, there is no point q such that $x(p_1) < x(q) < x(p_2)$ and $c(q) \neq c(p_1) = c(p_2)$. We say that S is an *ordered k-colored set of points*. In other words, an ordered k-colored set of points is such that all points of each color are consecutive according to the x-coordinate ordering. Analogously, we define an *ordered k-colored path* P to be a path where all vertices of the same color appear consecutively walking along P, and an *ordered k-colored sequence* to be a k-colored sequence where all elements with the same color appear consecutively

in the sequence. The technique behind this proof is a variant of the algorithm used in [3] to compute an augmenting k-colored Hamiltonian cycle of a k-colored simple cycle.

Lemma 2. *Let G and P be a planar k-colored graph and an ordered k-colored path that are compatible. There exists a simultaneous k-colored book embedding $< \Gamma_G, \Gamma_P >$ of G and P, such that Γ_G is a monotone topological book embedding, Γ_P is external and $\mathsf{cw}(\Gamma_P) \leq k$. Furthermore, $< \Gamma_G, \Gamma_P >$ can be computed in $\mathcal{O}(k\ n)$ time.*

Theorem 3. *Let G be a planar k-colored graph, and let σ be an ordered k-colored sequence. Then G admits an augmenting k-colored Hamiltonian path $\mathcal{H}$ consistent with σ that induces at most $3k+2$ division vertices per edge; at most 4 of these are pointy division vertices. Furthermore, $\mathcal{H}$ can be computed in $\mathcal{O}(k\ n)$ time.*

Sketch of Proof: Let P be an ordered k-colored path that is compatible with G and such that the k-colored sequence induced by P coincides with σ. By Lemma 1, G admits an augmenting k-colored Hamiltonian path $\mathcal{H}$ consistent with σ and inducing at most $2\,\mathsf{w}(\Gamma_P) + \mathsf{cw}(\Gamma_P) + 2$ division vertices for every s-shaped edge of Γ_G and at most $2\,\mathsf{w}(\Gamma_P) + 1$ division vertices for every u-shaped edge of Γ_G. Since by Lemma 2 $\mathsf{cw}(\Gamma_P) \leq k$ and since $\mathsf{w}(\Gamma_P) \leq \mathsf{cw}(\Gamma_P)$, it follows that $\mathcal{H}$ induces at most $3k + 2$ division vertices for every s-shaped edge and at most $2k + 1$ division vertices for every u-shaped edge. We now count the number of pointy division vertices. We first recall (see Lemma 1) that an edge can have one or two division vertices caused by the addition of an edge e' that transforms P into a cycle. Such division vertices are necessarily pointy division vertices. Namely, since a portion of e' is removed to obtain $\mathcal{H}$, then all the division vertices caused by the addition of e' appear at the beginning of $\mathcal{H}$ or at its end. Let V'_d be the set of these division vertices and let v_d be one of them. Let (u', v_d) and (v_d, v') be the two edges incident to v_d that are not in $\mathcal{H}$. Vertices (either real or division vertices) u' and v' are not in V'_d and therefore they are encountered both after v_d or both before v_d when walking along $\mathcal{H}'$, i.e. v_d is a pointy division vertex. Concerning the division vertices that are not created by the addition of e', it can be proved that at most one, in the case of u-shaped edges, or two, in the case of s-shaped edges, of these division vertices are pointy. It follows that an u-shaped edge can have at most two pointy division vertices and an s-shaped edge can have at most four pointy division vertices. □

A consequence of Theorem 3 and Theorem 1 is the following.

Theorem 4. *Let G be a k-colored planar graph with n vertices and let S be an ordered k-colored set of points compatible with G. There exists an $\mathcal{O}(n \log n + k\ n)$-time algorithm that computes a k-colored point-set embedding of G on S having curve complexity at most $3k + 7$.*

Theorem 3 can be applied also to another special k-coloring. Namely, let $G = (\bigcup_{i=0}^{k-1} V_i, E)$ be a k-colored planar graph; we say that the coloring of G is an

unbalanced k-colorings if $|V_i| = 1$ $(i = 0, 1, \ldots, k-2)$ and $|V_{k-1}| = n - k + 1$. In [1] it has been proved that if a k-colored planar graph G has an unbalanced k-colorings, then G admits a k-colored point-set embedding on any given set of points compatible with G with curve complexity at most $9k - 1$. We use Theorem 3 to improve this bound.

Theorem 5. *Let G be an n-vertex k-colored planar graph with an unbalanced coloring and let S be a k-colored set of points compatible with G. There exists an $\mathcal{O}(n \log n + k\ n)$-time algorithm that computes a k-colored point-set embedding of G on S having curve complexity at most $6k + 4$.*

5 *h*-Bend *k*-Colored Universal Sets and Grids

Let $\mathcal{F}$ be a family of k-colored planar graphs such that every element of $\mathcal{F}$ has n vertices and $1 \leq k \leq n$; let S be a k-colored set of points. We say that S is an *h-bend k-colored universal set* for $\mathcal{F}$ if, for every $G \in \mathcal{F}$, G has a k-colored point-set embedding on S having at most h bends per edge. In this section we shall use Theorem 3 to describe h-bend k-colored universal sets of points that can either have real coordinates (Subsection 5.1) or form an integer grid (Subsection 5.2).

5.1 *h*-Bend *k*-Colored Universal Sets

Let $\mathcal{F}_k$ be the family of all k-colored planar graphs with n vertices $(1 \leq k \leq n)$. In this section we show that there exist h-bend k-colored universal sets for $\mathcal{F}_k$ such that the number of points in the sets is $O(n)$ and h does not depend on n; we also show a lower bound on the size of such sets for the family $\mathcal{F}_2$. We start with a lemma that shows an h-bend k-colored universal set for a sub-family of all k-colored planar graphs with n vertices. Let $\mathcal{F}'_k$ be the family of k-colored planar graphs such that every graph of $\mathcal{F}'_k$ has n vertices and every two graphs of the family have the same number of vertices with color i $(1 \leq i \leq k)$. It is known that every k-colored set S of n points compatible with the graphs in $\mathcal{F}'_k$ is an h-bend k-colored universal set for $\mathcal{F}'_k$ with $h = \mathcal{O}(n)$ [1,9]. The next lemma shows that by adding $O(n)$ extra points to S, the curve complexity can become independent of n.

Lemma 3. *Let $G = (\bigcup_{i=0}^{k-1} V_i, E)$ be a k-colored planar graph with n vertices $(1 \leq k \leq n)$ with $|V_i| = n_i$ $(i = 0, 1, \ldots, k-1)$; let $S = \bigcup_{i=0}^{k-1} S_i$ be any k-colored set of points such that $|S_i| = k(n_i - 1) + 1$. There exists an $\mathcal{O}(n \log n + k\ n)$-time algorithm that computes a k-colored point-set embedding of G on S having curve complexity at most $3k + 7$.*

Sketch of Proof: We prove the statement by showing that it is possible to remove $(k-1)(n_i - 1)$ points from S_i $(i = 0, 1, \ldots, k-1)$ in such a way that the remaining $n = \sum_{i=0}^{k-1} n_i$ points form a set S' which induces an ordered k-colored sequence compatible with G. This, along with Theorem 3, implies that G admits an augmenting k-colored Hamiltonian path consistent with $\sigma = \mathsf{seq}(S')$ that

induces at most $3k+2$ division vertices; at most 4 of these division vertices are pointy division vertices. Such an augmenting k-colored Hamiltonian path can be computed in $\mathcal{O}(k\ n)$. By Theorem 1, there exists an $\mathcal{O}(n \log n)$-time algorithm that computes a k-colored point-set embedding of G on S' having curve complexity at most $3k+7$. Since there is a one-to-one mapping between the points of S and the elements of $\sigma_k = \mathsf{seq}(S)$, in the rest of this proof we concentrate on the sequence σ_k and prove that elements can be removed from σ_k in order to obtain an ordered k-colored sequence σ'_k compatible with G. More precisely, we prove that given a k-colored sequence σ_k such that the number of elements colored i is at least $k(n_i-1)+1$, it is possible to remove some elements from σ_k in order to create an ordered k-colored sequence σ'_k compatible with G. The proof is by induction on the number of colors k. If $k=1$ it is sufficient to arbitrarily remove $(k-1)(n_0-1)$ points (i.e. to remove no point) and the obtained sequence is an ordered 1-colored sequence compatible with G. If $k>1$, let $\sigma_k = c_0, \ldots, c_{k(n-k+1)-1}$. We denote as $\sigma_{i,j}$ the subsequence $c_i, c_{i+1}, \ldots, c_j$ of σ_k. Let $j_i = \min\{j \mid \sigma_{0,j}$ contains n_i elements whose value is $i\ \}$ for $i = 0, \ldots, k-1$ and let $j = \min_i\{j_i\}$. Without loss of generality, assume that $j = j_0$. The sequence $\sigma_{0,j}$ contains n_0 elements whose value is 0 and at most $n_i - 1$ elements whose value is i $(i = 1, 2, \ldots, k-1)$. Therefore $\sigma_{j+1,k(n-k+1)-1}$ contains at least $(k-1)(n_i-1)+1$ elements whose value is i $(i = 1, 2, \ldots, k-1)$. The sequence $\sigma_{(k-1)} = \sigma_{j+1,k(n-k+1)-1} \setminus \{c_j \mid c_j = 0\}$ is a $(k-1)$-colored sequence such that the number of elements colored i is at least $(k-1)(n_i-1)+1$. Thus, by induction, one can remove elements from $\sigma_{(k-1)}$ in order to obtain an ordered $(k-1)$-colored sequence σ'_{k-1} compatible with $G \setminus \{v \in V \mid col(v) = 0\}$. It follows that the sequence $\sigma'_k = \sigma_{0,j} \setminus \{c_j \mid c_j \neq 0\} \cup \sigma'_{k-1}$ is an ordered k-colored sequence compatible with G. □

Theorem 6. *Let $\mathcal{F}_k$ be the family of all k-colored planar graphs with n vertices $(1 \leq k \leq n)$. Any k-colored set of points S such that S contains $k\ n-k^2+1$ points for each color is a $(3k+7)$-bend k-universal set for $\mathcal{F}_k$. Furthermore, there exists an $\mathcal{O}(n \log n + k\ n)$-time algorithm that computes a k-colored point-set embedding on S of any $G \in \mathcal{F}_k$ with curve complexity at most $3k+7$.*

Sketch of Proof: Let G be a graph of $\mathcal{F}_k$. For each color i, G has at most $n-k+1$ vertices of color i. Since S has $k(n-k)+1$ points of color i, the result follows from Lemma 3. □

The total number of points in a k-colored set of points that satisfies the statement of Theorem 6 is $k^2n - k^3 + k$. One can ask whether $n + o(n)$ points are sufficient to guarantee a curve complexity that does not depend on n. As the next theorem shows, this question has a negative answer for the case $k=2$.

Theorem 7. *Let c be a constant such that $c > 1$. For every integer $n > 2\frac{c}{c-1}$ there exists a 2-colored planar graph $G = (V_0 \cup V_1, E)$ and a 2-colored set of points $S = S_0 \cup S_1$ consisting of $n + \frac{n}{c}$ points such that: (i) $|V_0| = |V_1| = \frac{n}{2}$; (ii) $|S_0| = |S_1| = \frac{n}{2} + \lceil \frac{n}{2c} \rceil$; and (iii) any 2-colored point-set embedding of G on S has one edge with at least $\frac{2}{3}\left\lfloor \frac{c-1}{2c} n \right\rfloor - 1$ bends.*

5.2 h-Bend k-Colored Universal Grid

Let $\mathcal{F}_k$ be the family of all k-colored planar graphs with n vertices ($1 \leq k \leq n$). An *h-bend k-colored universal grid* for $\mathcal{F}_k$ is a k-colored set of points S such that: (i) S is an integer grid; (ii) any element of $\mathcal{F}_k$ has a k-colored point-set embedding Γ on S with curve complexity at most h; and (iii) the bends of Γ are at grid points. The drawing Γ is called a *k-colored point-set grid embedding*. In this section we study the size of an h-bend k-colored universal grid that supports k-colored point-set grid embeddings whose curve complexity does not depend on the input size. Let S be the k-colored set of points that contains points $p = (x, y)$ such that $x, y \in \mathbb{Z}$ and $0 \leq x, y < 2kN$ where $N = (n - k + 1)(3n - 5)$. Let each point $p = (x, y)$ of S have color $col(p) = \lfloor \frac{x}{2N} \rfloor$. We call S the *(n, k)-strip grid*.

Theorem 8. *Let $\mathcal{F}_k$ be the family of all k-colored planar graphs with n vertices ($1 \leq k \leq n$). The (n, k)-strip grid is a $(6k + 5)$-bend k-universal grid for $\mathcal{F}_k$. Furthermore, there exists an $\mathcal{O}(k\ n)$-time algorithm that computes a k-colored point-set grid embedding on S of any $G \in \mathcal{F}_k$ with curve complexity at most $6k + 5$.*

Sketch of Proof: Let G be a k-colored planar graph with n vertices and m edges. Let σ be an ordered k-colored sequence compatible with G and such that elements colored i appear before than elements colored $i + 1$ ($i = 0, \ldots, k - 2$). By Theorem 3, G admits an augmenting k-colored Hamiltonian path $\mathcal{H}$ consistent with σ. Since the (n, k)-strip grid S contains more than n points for each color we can arbitrarily choose a subset S' of S such that $\mathsf{seq}(S) = \sigma$ and use Theorem 1 to compute a k-colored point-set embedding of G on S. However, the technique behind Theorem 1 does not guarantee that the division vertices are at grid point even if the point in S' are grid points. We describe in the following a variant of this technique that places bends at grid points. Let $w_0, w_1, \ldots, w_{n'}$ be the vertices (either division vertices or real vertices) of $\mathcal{H}$ in the order they appear in $\mathcal{H}$. Since σ is ordered we have that the real vertices of G in $\mathcal{H}$ are ordered along $\mathcal{H}$ except for the presence of the division vertices, i.e. if we ignore the division vertices then all vertices of the same color appear consecutively walking along $\mathcal{H}$. Define the following indices: $j_i = \max\{\ j \mid \mathrm{col}(w_j) = i\ \}$. All the division vertices w_j such that $j_{i-1} < j < j_i$ are given color i, where we set $j_{-1} = -1$. With this coloring of the division vertices, we have that $\mathcal{H}$ is an ordered k-colored path, i.e. it consists of a set of vertices (either division vertices or real vertices) colored c_0, followed by a set of vertices colored c_1, etc. Since σ has been chosen so that elements colored i appear before than elements colored $i + 1$ ($i = 0, \ldots, k - 2$), then $c_i = i$ ($0 \leq i \leq k - 1$), i.e. $\mathcal{H}$ consists of a set of vertices colored 0 followed by a set of vertices colored 1, etc. The number of real vertices of a given color i is at most $n - k + 1$ (because at least one vertex for any other color must exist), and the number of division vertices between a pair of consecutive real vertices is at most m (because each edge of $\mathcal{H}$ is crossed at most once by an edge connecting two real vertices), which, in turn, is at most $3n - 6$ since the graph is planar. It follows that we have at most $(n - k + 1)(m + 1) \leq N$ vertices of each color. Let w_j be a vertex (either a real or a division vertex) whose color is i,

then $j_{i-1} < j \leq j_i$. Vertex w_j is drawn at point having coordinates $(2x, 2x)$ where $x = iN + (j - j_{i-1}) - 1$. Since $0 \leq 2((j - j_{i-1}) - 1) < 2N$, then $\lfloor \frac{2x}{2N} \rfloor = i$, i.e. point $(2x, 2x)$ is colored i. Let $e = (w_{j_a}, w_{j_b})$ be an edge (either a real edge of G or a portion of an edge of G). Since the endvertices of $\mathcal{H}$ are on the same face of $\mathsf{Ham}(G)$, there exists a planar embedding of $\mathsf{Ham}(G)$ such that w_0 and $w_{n'-1}$ are on the external face. In such an embedding every edge not in $\mathcal{H}$ is either on the left-hand side of $\mathcal{H}$ or on the right-hand side of $\mathcal{H}$ when walking from w_0 to $w_{n'-1}$. If e belongs to $\mathcal{H}$, it is drawn as a straight-line segment. If e is to the left of $\mathcal{H}$, then it is drawn with only one bend whose coordinates are $(2x_a + 1, 2x_b - 1)$, where $(2x_a, 2x_a)$ is the point representing vertex w_{j_a} and $(2x_b, 2x_b)$ is the point representing vertex w_{j_b}. If e is to the right of $\mathcal{H}$, then it is drawn with only one bend whose coordinates are $(2x_b - 1, 2x_a + 1)$. □

Corollary 1. *Every n-vertex planar graph admits a k-colored point-set grid embedding with curve complexity $6k + 5$ on a grid whose size is $\mathcal{O}(k\ n^2) \times \mathcal{O}(k\ n^2)$.*

References

1. Badent, M., Di Giacomo, E., Liotta, G.: Drawing colored graphs on colored points. In: Dehne, F.K.H.A., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 102–113. Springer, Heidelberg (2007)
2. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing. Prentice Hall, Upper Saddle River, NJ (1999)
3. Di Giacomo, E., Didimo, W., Liotta, G., Meijer, H., Trotta, F., Wismath, S.K.: k-colored point-set embeddability of outerplanar graphs. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 318–329. Springer, Heidelberg (2007)
4. Di Giacomo, E., Didimo, W., Liotta, G., Wismath, S.K.: Curve-constrained drawings of planar graphs. Computational Geometry 30, 1–23 (2005)
5. Di Giacomo, E., Liotta, G., Trotta, F.: On embedding a graph on two sets of points. IJFCS, Special Issue on Graph Drawing 17(5), 1071–1094 (2006)
6. Kaneko, A., Kano, M.: Discrete geometry on red and blue points in the plane - a survey. In: Discrete & Computational Geometry, Algorithms and Combinatories, vol. 25, pp. 551–570. Springer, Heidelberg (2003)
7. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
8. Kaufmann, M., Wiese, R.: Embedding vertices at points: Few bends suffice for planar graphs. Journal of Graph Algorithms and Applications 6(1), 115–129 (2002)
9. Pach, J., Wenger, R.: Embedding planar graphs at fixed vertex locations. Graph and Combinatorics 17, 717–728 (2001)

Colorability in Orthogonal Graph Drawing*

Jan Štola

Department of Applied Mathematics, Charles University
Malostranské nám. 25, Prague, Czech Republic
Jan.Stola@mff.cuni.cz

Abstract. This paper studies the question: What is the maximum integer $k_{b,n}$ such that every $k_{b,n}$-colorable graph has a b-bend n-dimensional orthogonal box drawing?

We give an exact answer for the orthogonal line drawing in all dimensions and for the 3-dimensional rectangle visibility representation. We present an upper and lower bound for the 3-dimensional orthogonal drawing by rectangles and general boxes. Particularly, we improve the best known upper bound for the 3-dimensional orthogonal box drawing from 183 to 42 and the lower bound from 3 to 22.

1 Introduction

The visualization of relational information has many applications in various domains. The domain entities are usually modeled as vertices and the relationships among entities are represented by edges.

There have been many graph drawing styles studied in the literature. In this paper we study the orthogonal box drawing. This drawing has received a wide attention recently due to its applications: 2-dimensional variants in VLSI routing, circuit board layout, CASE tools etc. and 3-dimensional variants for example in packaging algorithms [1,5,6,12].

The orthogonal box drawing represents vertices by axis-parallel boxes. Every edge is drawn as an axis-parallel polyline with ends on boundaries of boxes that correspond to vertices of the edge. Edges don't intersect other boxes and with the exception of the 2D drawing an edge cannot intersect another edge.

We call the drawing b-bend if each edge consists of at most $b+1$ line segments. A 0-bend drawing is called a straight-line drawing. If all edges in a straight-line box drawing in $\mathbb{R}^3$ are parallel then we can ignore the thickness of the boxes in this direction. We obtain a representation known as a 3D rectangle visibility drawing. The same operation in $\mathbb{R}^2$ gives us a bar-visibility drawing.

It turns out that the recognition of graphs with the given type of orthogonal drawing is difficult. For example, Shermer [7] shows that the recognition of graphs with 2-dimensional straight-line orthogonal drawing is NP-complete. Fekete et al. [8] establish NP-completeness of recognition of graphs with a 3D rectangle visibility drawing by squares.

* Supported by the Institute for Theoretical Computer Science, Charles University, Prague, project No. 1M0545 of the Czech Ministry of Education.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 327–338, 2007.

If we cannot effectively decide whether a graph has a drawing of the given type then it is natural to look for classes of graphs for which this decision is possible. The previous research was concentrated mainly on complete graphs e.g. on the determination of the maximum size of a complete graph with a drawing [2,3,4]. Unfortunately such results don't tell us much about drawing of graphs with more vertices.

Our search for a more practical class of graphs has been inspired by the open problem presented by Wood [1]:

> What is the maximum $k \in \mathbb{Z}^+$ such that every k-colorable graph has a straight-line 3D orthogonal box drawing?

We study graphs with bounded colorability in this paper. Every k-colorable graph is a subgraph of a k-partite graph that is itself k-colorable. Therefore it is sufficient to study drawing of k-partite graphs.

Definition 1. *The multipartite number of the given type of drawing is the maximum $k \in \mathbb{N}$ such that every k-partite graph has a drawing of that type. We say that the multipartite number is infinite when every multipartite graph has such a drawing.*

Wood [1] proves that the multipartite number of the straight-line orthogonal box drawing is at least 3. On the other hand Fekete and Meijer [2] shows that it is at most 183.

We improve the lower bound from 3 to 22 and the upper bound from 183 to 42. We also determine the exact value of the multipartite number of the orthogonal drawing by line segments and of the rectangle visibility drawing. Table 1 summarizes the results presented in this work.

Table 1. Multipartite number of d-dimensional b-bend orthogonal drawing by v-dimensional boxes

v	d	b	multipartite number	
1	1	0	1	Theorem 1
	2	≥ 2	∞	Section 3.3
		1	2	Theorem 2
		0	1	Theorem 1
	3	≥ 1	∞	Theorem 2
	≥ 3	0	3	Theorem 1
2	2	≥ 1	∞	Section 4.3
		0	1	Section 4.2
	3	0	$\in \langle 22, 42 \rangle$	Theorems 4, 5
3	3	≥ 1	∞	Section 5.2
		0	$\in \langle 22, 42 \rangle$	Theorems 4, 5
rectangle visibility drawing			8	Theorem 3

2 Preliminaries

The next lemma is a simple application of the pigeon-hole principle. We include it because we use this formulation several times in the sequel.

Lemma 1. *Let $k, n, c \in \mathbb{N}$ and G be a complete k-partite graph whose each part has at least $c(n-1)+1$ vertices and each vertex has one of c colors. Then G contains a complete k-partite subgraph whose each part is monochromatic and contains at least n vertices.*

Proof. Each part contains at least $c(n-1)+1$ vertices. Therefore there are at least n vertices with the same color in each part. These monochromatic sets form the k-partite subgraph with the required properties. □

We use this lemma when each vertex from a drawing must have one property from a finite set of properties and we have to ensure that the vertices from the same part have the same property. A similar situation occurs when the properties are assigned to edges.

Lemma 2. *Let $k, n, c \in \mathbb{N}$ and G be a complete k-partite graph whose each edge has one of c colors. There exists $N_{k,n,c} \in \mathbb{N}$ such that if each part of G has at least $N_{k,n,c}$ vertices then G contains a complete k-partite subgraph whose each part has at least n vertices and for each pair of parts the edges among elements of these parts are monochromatic.*

Proof. Recall that the bipartite Ramsey number $b_c(H)$ is the minimum m such that every c-coloring of $E(K_{m,m})$ yields a monochromatic copy of H. Chvátal [10] and Bieneke-Schwenk [11] proved that $b_c(K_{p,q}) \leq (q-1)c^p + O(c^{p-1})$.

If we fix two parts P_1 and P_2 that have at least $b_c(K_{n,n})$ vertices then there is a subgraph of G that is also complete k-partite, has at least n vertices from P_i, $i = 1, 2$ and the edges among these vertices are monochromatic. The required subgraph can be obtained by a repeatable application of this fact. □

Sometimes we need to separate the parts with respect to some function on the set of vertices.

Lemma 3. *Let $k, n \in \mathbb{N}$ and $G(V, E)$ be a complete k-partite graph whose each part has at least $(n-1)k+1$ vertices. For each $\ell : V \to \mathbb{R}$ there exists a complete k-partite subgraph G' of G whose each part has at least n vertices and whose parts are ℓ-separated e.g. for each pair P_1, P_2 of parts it is either $\forall x \in P_1 \forall y \in P_2\ \ell(x) \leq \ell(y)$ or $\forall x \in P_1 \forall y \in P_2\ \ell(y) \leq \ell(x)$.*

Proof. Sort the vertices $v \in V$ according to their value $\ell(v)$. Let P be the part whose n-th vertex (with respect to this order) has the lowest index in the sequence. Remove the vertices before the selected one from the sequence. Put the first n vertices of P into G' and remove the elements of P from the sequence. Continue in the same way until the sequence is empty.

Let's fix some part P. If P is not the selected part then at most $n-1$ of its vertices are removed from the sequence. Therefore at most $(n-1)(k-1)$ of

its vertices are removed before the part is selected. P has at least $(n-1)k+1$ elements. So, the described algorithm selects n vertices from each part. The resulting graph G' obviously has the required property. □

Lemmas 1, 2 and 3 ensure the existence of a subgraph $G'(V', E')$ of a graph $G(V, E)$ such that $|V'| \geq f(|V|)$, where f is a non-decreasing function unbounded from above. These properties of f ensure that the size of G' can be made arbitrarily big if we take the original graph G sufficiently large. We use this fact many times in the sequel because we usually want to prove the existence of a large graph G' with some properties and are not interested in the exact size of G that must be taken to find a subgraph of the required size.

3 Line Drawing

3.1 Straight-Line Line Drawing

In this section we determine the multipartite number of the straight-line line drawing in the n-dimensional space e.g. the drawing where each vertex is represented by an axis-parallel line segment in $\mathbb{R}^n$.

Lemma 4. *The multipartite number of the straight-line line drawing in* $\mathbb{R}^n$ *is at most* 3 *for* $n > 2$ *and it is 1 for* $n = 1, 2$.

Proof. Let's suppose that we have a straight-line line drawing in $\mathbb{R}^n$ of a k-partite graph G. If we color the vertices according to their direction then Lemma 1 tells us that there exists a large k-partite subgraph G' with parallel vertices in the individual parts.

Now color the edges of the graph G' according to their direction. Let G'' denote the result of the application of Lemma 2 on the graph G'. The edges between arbitrary two parts of G'' are parallel.

We know that the vertices in the individual parts are parallel. We claim that the vertices from the different parts cannot be parallel. Suppose that the opposite holds e.g. there are parts P and Q such that the vertices from $P \cup Q$ are parallel (to a vector $\boldsymbol{e}_1$). The edges between P and Q are parallel (to a vector $\boldsymbol{e}_2$) due to the definition of G''. This means that P and Q together with the edges between them lie in a plane (given by vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$). So, we have a bar-visibility graph of $K_{|P|,|Q|}$, but the sets P and Q can be made arbitrarily large. That is in a contradiction with the planarity of bar-visibility graphs.

If P and Q are two parts of G'', P parallel to $\boldsymbol{e}_1$, Q parallel to $\boldsymbol{e}_2$ and the edges between P and Q parallel to $\boldsymbol{e}_3$ then the drawing of $K_{|P|,|Q|}$ lies in $S = p + \boldsymbol{e}_1\mathbb{R} + \boldsymbol{e}_2\mathbb{R} + \boldsymbol{e}_3\mathbb{R}$, where p is a point of some line in $P \cup Q$. If $k > 3$ then there must exist a part R parallel to vector $\boldsymbol{e}_4 \notin \boldsymbol{e}_1\mathbb{R} + \boldsymbol{e}_2\mathbb{R} + \boldsymbol{e}_3\mathbb{R}$. The edges between P and R are parallel to a vector $\boldsymbol{v}$.

Choose $l \in R$. $|l \cap S| \leq 1$ because $\boldsymbol{e}_4 \perp S$.

If $l \cap S = \{q\}$ then the edges between P and R have one end in q. There can be at most two such edges (from the directions $\boldsymbol{v}$ and $-\boldsymbol{v}$). Therefore $|P| \leq 2$, but P can be arbitrarily large.

If $l \cap S = \emptyset$ then $l + \boldsymbol{v}\,\mathbb{R}$ intersects S in at most one point. This means that l can be connected to at most one vertex from P and we have a contradiction with the size of P again. Hence $k \leq 3$.

The case $n = 1$ is obvious. If $n = 2$ then $G(V, E)$ is a union of two planar bar-visibility graphs and as such has less than $12|V|$ edges. Therefore no sufficiently large bipartite graph has a 2D straight-line line drawing. □

The previous proof utilizes the fact that we can choose from an arbitrary complete multipartite graph a subgraph that is itself a complete multipartite graph, has some specific property and can be made arbitrarily large if the original graph is sufficiently big. The additional property allows us to simplify the proof. We use this method in many of the following proofs.

The lower bound on the multipartite number that matches our upper bound e.g. the construction of a 3-dimensional straight-line line drawing of $K_{a,b,c}$ for arbitrary positive integers a, b, c is given by Wood [1].

Theorem 1. *The multipartite number of the straight-line line drawing in* $\mathbb{R}^n$ *is* 3 *for* $n > 2$ *and it is 1 for* $n = 1, 2$.

3.2 1-Bend Line Drawing

Theorem 2. *The multipartite number of the* 1*-bend line drawing in* $\mathbb{R}^n$ *is 2 for* $n = 2$ *and is infinite for* $n > 2$.

Proof. The proof is similar to the proof of Theorem 1. See [9] for details. □

3.3 2-Bend Line Drawing

The Figure 1 shows that every complete graph has a 2-bend 2-dimensional line drawing. Therefore the multipartite number of this drawing is infinite.

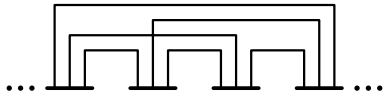


Fig. 1. 2-bend 2-dimensional line drawing of K_k

4 Rectangle Drawing

In this section we study rectangle drawing e.g. box drawing where vertices are represented by 2D boxes. We work with rectangles in parallel planes as if they were in the same plane. Operations on such rectangles should be understood as operations on the projections (into one of the planes) and the projection of the result of the operation (for example the intersection of some rectangles) back into the individual planes of the rectangles.

4.1 Rectangle Visibility Drawing

Definition 2. *Let B be a box in an orthogonal drawing. $x^+(B)$ denotes the maximum coordinate of a point in the box B. Similarly we define x^-, y^+, y^-, z^+ and z^-.*

Note that $z^+ = z^-$ in the rectangle visibility drawing. We denote this function simply by z there.

Lemma 5. *The multipartite number of the rectangle visibility drawing is at most 8.*

Proof. Suppose that we have a rectangle visibility drawing of a complete k-partite graph G. Apply Lemma 3 on this graph consecutively with functions z, x^+, x^-, y^+ and y^-. We obtain a graph G' with parts separated with respect to these functions.

Take an arbitrary part P_i of G' and sort its elements according to their z-coordinates. Due to Erdős-Szekeres theorem we can choose from this sequence a subsequence P'_i of length at least $|P_i|^{1/16}$ that is monotone in x^+, x^-, y^+ and y^- coordinates. Denote by G'' the complete k-partite graph with parts P'_i. From the construction of G'' it is obvious that its parts can be made arbitrarily large if we take G with sufficiently large parts.

We claim that we can suppose that the orthogonal projections (along the z-axis) of rectangles from G'' have a common intersection. Rectangles from the different parts must intersect to be able to see each other. So, it is sufficient to show that each part has a common intersection. That happens if and only if each two rectangles from this part intersect.

Let P_1 be a part without a common intersection. There must be two elements $r_1, r_2 \in P_1$ that don't intersect. Without loss of generality it is $x^+(r_1) < x^-(r_2)$.

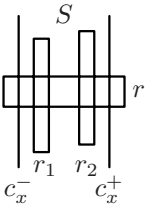


Fig. 2.

G'' is a complete k-partite graph. Hence a rectangle r from a different part (to see both r_1 and r_2) must have $x^-(r) < x^+(r_1)$ and $x^+(r) > x^-(r_2)$.

Let's modify the part P_1 to have a common intersection. Let c_x^+ (respectively c_x^-) be a maximum (resp. minimum) x-coordinate of a point of a rectangle in P_1. Denote by S the y-parallel strip between c_x^- and c_x^+ (see Figure 2).

The proved inequalities together with the x^+, x^--separability of parts ensures that $x^-(r) < c_x^-$ and $c_x^+ < x^+(r)$ for each rectangle $r \notin P_1$. Therefore if two

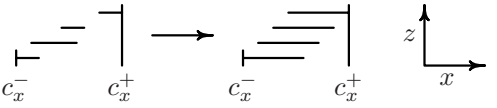


Fig. 3. Modification of stairs

rectangles not in P_1 can see each other through a point (x, y) in the strip S then they can see each other also through a point $((c_x^- - \varepsilon, y)$ or $(c_x^+ + \varepsilon, y)$ for a sufficiently small $\varepsilon > 0)$ outside the strip.

The rectangles in P_1 (ordered according to the z-coordinate) are monotone in x^+ and x^- coordinates and don't have a common intersection. So, they must be either increasing or decreasing in both these coordinates - the rectangles form stairs (see Figure 3).

We claim that if we change x^+ and x^- coordinates of the stair rectangles to ensure a common intersection (as shown in Figure 3) then we don't destroy the completeness of the k-partite visibility representation of G''.

Only the rectangles in the strip S are modified. So, the visibility among rectangles not in P_1 is not affected because they can see each other through points outside the strip. It remains to show that the rectangles from P_1 can see all other rectangles.

Sides y^+ and y^- of each rectangle not in P_1 cross the whole width of the strip S. They mark on the strip (orthogonal) sub-strips. The rectangles not in P_1 can see the rectangles from P_1 only through their sub-strips.

No visibility is destroyed if we move the x^+ and x^- coordinates of rectangles from P_1 such that the same rectangles remain visible through each sub-strip, but that is exactly what our stair-modification technique does.

We have shown that we can expect each part of G'' to have a common intersection.

We know that if we sort the rectangles from some part P of G'' according to their z-coordinates then we obtain a sequence monotone also in x^+, x^-, y^+ and y^- coordinates. Moreover if P has a common intersection then we can consider P to form a frame with sides oriented up and down (see Figure 4). The orientation determines the direction from which the corresponding sides of rectangles are visible.

We also know that the parts are x^+, x^-, y^+, y^--separated. Thus two corresponding sides of frames cannot intersect (see Figure 5). The interiors of frames

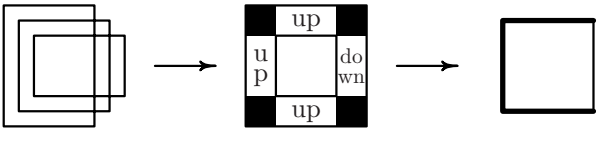


Fig. 4. Transformation of one part into a frame and an oriented rectangle

intersect because all rectangles have a common intersection. Due to these facts we can shrink the frames into rectangles with oriented sides. Now two parts can see each other if the boundaries of their oriented rectangles intersect and the intersecting sides have a correct orientation.

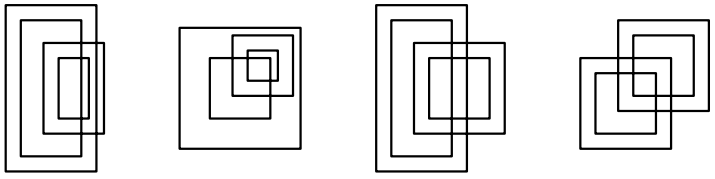


Fig. 5. Examples of invalid (a), (b) and valid (c), (d) intersections of frames

It remains to prove that a complete graph with this modified oriented rectangle visibility representation has at most 8 vertices.

Lemma 6. *If K_n has a modified visibility representation by rectangles with oriented sides (as described in the previous proof) then $n \leq 8$.*

Proof. We proceed in a similar way as Fekete et al. [3] in the proof of the nonexistence of a visibility representation of K_8 by unit squares e.g. by a computer search. Our algorithm is based on their algorithm. We modify it to generate visibility representations with general rectangles (not only squares). We also add a code that assigns an orientation to the individual sides. When the next rectangle is added the new code also checks whether the orientation requirements are satisfied. See [9] for details (including the source code).

We were able to process all valid configurations in 26 hours on Intel Centrino 1.7 GHz machine and verified that there is no representation of K_9 with the required properties.[1] □

Lemma 7. *The multipartite number of the rectangle visibility drawing is at least 8.*

Proof. The Figure 6 shows (in the form of oriented rectangles) a rectangle visibility drawing of a complete 8-partite graph. The numbers in the lower right corner determine the order of parts with respect to the z-coordinate. The thick sides are oriented up, the thin sides are oriented down. The small circles show areas where the individual parts see each other. □

If we put together Lemmas 5, 6 and 7 we obtain the following theorem.

Theorem 3. *The multipartite number of the rectangle visibility drawing is 8.*

[1] The algorithm was run several times on different hardware configurations in fact. Check sums were used to recognize computations affected by a potential hardware failure.

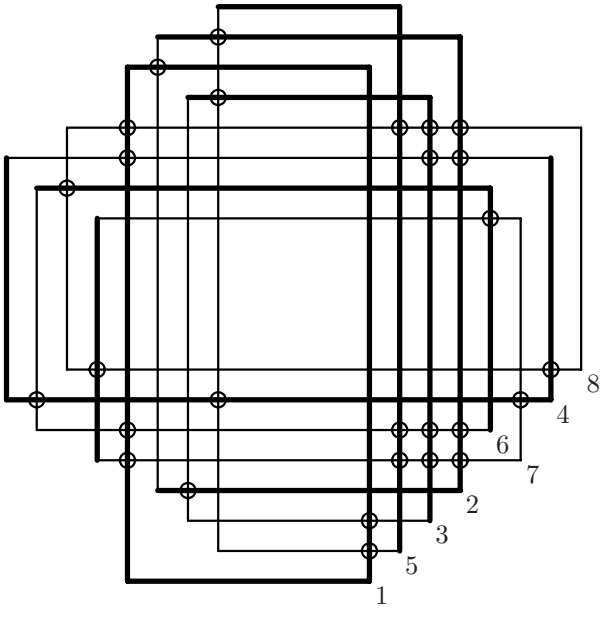


Fig. 6. Rectangle visibility representation of $K_{a,b,c,d,e,f,g,h}$

4.2 Straight-Line Rectangle Drawing

A graph with a 2-dimensional straight-line rectangle drawing is a union of two planar bar-visibility graphs. Therefore the multipartite number of such a drawing is one.

The 3-dimensional straight-line rectangle drawing has similar properties to the 3-dimensional straight-line box drawing. The upper bound on the multipartite number of the 3-dimensional straight-line box drawing proved in the next section is also the best known bound for the straight-line rectangle drawing. On the other hand the following drawing of a complete 22-partite graph provides also the best known lower bound on the multipartite number of the 3-dimensional straight-line box drawing.

Theorem 4. *The multipartite number of the straight-line rectangle drawing in* $\mathbb{R}^3$ *is at least 22.*

Proof. The Figure 7 shows a rectangle visibility drawing of a complete 6-partite graph. Take a copy of the construction from the Figure 6 and rotate it to be parallel to xz-plane. Place the copy between the third and the fourth part. Ensure that x^+ (resp. x^-) coordinates of the rectangles from the copy are bigger (resp. smaller) than the coordinates of the rectangles from the 6-partite graph.

Take another copy of the construction from the Figure 6 and rotate it to be parallel to yz-plane. Place the copy again between the third and the fourth part either below or over the first copy. Ensure that y^+ (resp. y^-) coordinates of the

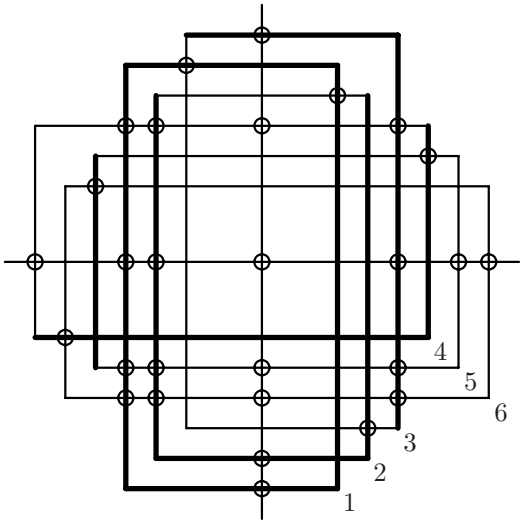


Fig. 7. Straight-line rectangle drawing of a complete 22-partite graph

rectangles from this copy are bigger (resp. smaller) than the coordinates of the rectangles from the 6-partite graph.

The copies are schematically displayed on the Figure 7 by the horizontal and the vertical line. The small circles show the areas where the individual parts see each other. It can be easily verified that the resulting construction is a straight-line rectangle drawing of a complete 22-partite graph. □

4.3 1-Bend Rectangle Drawing

It can be shown (using a construction similar to Figure 1) that the multipartite number of 1-bend rectangle drawing in $\mathbb{R}^2$ is infinite – if we place the rectangles on a diagonal then we need only one bend to connect any pair of rectangles.

5 Box Drawing

5.1 Straight-Line Box Drawing

Definition 3. *Let A and B be axis-parallel boxes in $\mathbb{R}^3$. We write $A \prec_x B$ if A and B can see each other and $x^+(A) < x^-(B)$. Similarly we define $A \prec_y B$ and $A \prec_z B$.*

It can be easily verified that these relations are partial orders on the boxes in a straight-line box drawing of a complete graph. We say that some boxes from a drawing form an x-chain (resp. x-antichain) if they form a chain (resp. antichain) in the partial order $\prec_x$.

Fekete and Meijer [2] showed the following properties of these orders.

Lemma 8. *Let C be a maximum length x-chain in a 3-dimensional orthogonal box drawing of a complete graph. There cannot be an x-chain D of length greater than 4 such that $C \cap D = \emptyset$.*

Lemma 9. *If there is no chain (x-chain, y-chain or z-chain) longer than 4 in a 3-dimensional orthogonal box drawing of a complete graph G then G can have at most 18 vertices.*

Lemmas 8 and 9 allow us to estimate the maximum size of a complete graph with a drawing with the bounded length of chains.

Lemma 10. *If k is the maximum length of a chain that appears in the partial order $\prec_x$, $\prec_y$ or $\prec_z$ in a 3-dimensional straight-line box drawing of K_n then $n \leq 3k + 18$.*

Proof. Let C_x, C_y resp. C_z denotes the maximum x-chain, y-chain resp. z-chain in the drawing. If we remove the boxes in $C_x \cup C_y \cup C_z$ from the drawing then by Lemma 8 there cannot remain a chain of length 5. There remain at most 18 boxes by Lemma 9. Hence, $n \leq |C_x| + |C_y| + |C_z| + 18 \leq 3k + 18$. □

The presented properties of the partial orders $\prec_x$, $\prec_y$ and $\prec_z$ can be utilized to prove an upper bound for the multipartite number of a box drawing.

Theorem 5. *The multipartite number of the straight-line box drawing in $\mathbb{R}^3$ is at most 42.*

Proof. Let G be a large complete k-partite graph that has a 3D straight-line box drawing. Color its edges by three colors according to their direction. Apply Lemma 2 on G and denote by G' the resulting graph.

Select one box from each part of G'. These boxes form a 3D straight-line box drawing of K_k. We claim that this drawing doesn't contain a chain of length greater than 8. If we prove this then by the previous lemma $k \leq 3.8 + 18 = 42$.

Suppose by contradiction that there exists an x-chain of length 9. The boxes from the parts with a member in this chain also (due to the selection of G') see each other along the x-axis. Therefore they correspond to a rectangle visibility representation of a 9-partite graph that can be made arbitrarily large if G is taken sufficiently large. This is in a contradiction with Theorem 3. □

5.2 1-Bend Box Drawing

The infinity of the multipartite number of the 1-bend box drawing comes immediately from the infinity for the 1-bend 3D line drawing.

6 Conclusion

We determine the multipartite number of several types of drawings. This significantly enlarges the class of graphs that are known to have such drawings. For example Fekete and Meijer show in [2] that each graph with at most 56 vertices has a 3-dimensional straight-line orthogonal box drawing. Comparing to this we prove that any 22-colorable graph has such a drawing.

References

1. Wood, D.R.: Three-Dimensional Orthogonal Graph Drawing, Ph.D. Thesis, School of Computer Science and Software Engineering, Monash University (2000)
2. Fekete, S.P., Meijer, H.: Rectangle and box visibility graphs in 3D. Int. J. Comput. Geom. Appl. 97, 1–28 (1997)
3. Fekete, S.P., Houle, M.E., Whitesides, S.: New results on a visibility representation of graphs in 3D. In: Proc. Graph Drawing, vol. 95, pp. 234–241 (1995)
4. Bose, P., Dean, A., Hutchinson, J., Shermer, T.: On rectangle visibility graphs. In: Proc. Graph Drawing, vol. 96, pp. 25–44 (1996)
5. Hutchinson, J.P., Shermer, T., Vince, A.: On representations of some thickness-two graphs. Comput. Geom. 13(3), 161–171 (1999)
6. Dean, A.M., Hutchinson, J.P.: Rectangle-visibility Layouts of Unions and Products of Trees. J. Graph Algorithms Appl., 1–21 (1998)
7. Shermer, T.: Block visibility representations III: External visibility and complexity. In: Fiala, K., Sack (eds.) Proc. of 8th Canadian Conf. on Comput. Geom. Int. Informatics Series, vol. 5, pp. 234–239. Carleton University Press, Ottawa (1996)
8. Fekete, S.P., Houle, M.E., Whitesides, S.:: The wobbly logic engine: proving hardness of non-rigid geometric graph representation problems, Report No. 97.273, Angewandte Mathematik und Informatik Universität zu Köln (1997)
9. Štola, J.: Chromatic invariants in graph drawing, Master's Thesis, Department of Applied Mathematics, Charles University, Prague, Czech Republic (2006), http://kam.mff.cuni.cz/~stola/chromaticInvariants.pdf
10. Chvátal, V.: On finite polarized partition relations. Canad. Math. Bul. 12, 321–326 (1969)
11. Beineke, L.W., Schwenk, A.J.: On a bipartite form of the Ramsey problem. In: Proc. 5th British Combin. Conf. 1975, Congr. Numer., vol. XV, pp. 17–22 (1975)
12. Biedl, T., Shermer, T., Whitesides, S., Wismath, S.: Bounds for orthogonal 3D graph drawing. J. Graph Alg. Appl. 3(4), 63–79 (1999)

A Note on Minimum-Area Straight-Line Drawings of Planar Graphs*

Fabrizio Frati and Maurizio Patrignani

Dipartimento di Informatica e Automazione – Università di Roma Tre
{frati,patrigna}@dia.uniroma3.it

Abstract. Despite a long research effort, finding the minimum area for straight-line grid drawings of planar graphs is still an elusive goal. A long-standing lower bound on the area requirement for straight-line drawings of plane graphs was established in 1984 by Dolev, Leighton, and Trickey, who exhibited a family of graphs, known as *nested triangles graphs*, for which $(2n/3 - 1) \times (2n/3 - 1)$ area is necessary. We show that nested triangles graphs can be drawn in $2n^2/9 + O(n)$ area when the outer face is not given, improving a previous $n^2/3$ area upper bound. Further, we show that $n^2/9 + \Omega(n)$ area is necessary for any planar straight-line drawing of a nested triangles graph. Finally, we deepen our insight into the $4/9n^2 - 4/3n + 1$ lower bound by Dolev, Leighton, and Trickey, which is conjectured to be tight, showing a family of plane graphs requiring more area.

1 Introduction

Area minimization is recognized to be an important aesthetic requirement in Graph Drawing. Besides, drawing planar graphs in the minimum area is a long-standing and fascinating combinatorial problem. In 1984, Dolev *et al.* [5] first exhibited a family of graphs, called *nested triangles* graphs, to show an area lower bound of $(2n/3-1) \times (2n/3-1)$ for straight-line drawings of plane graphs, that is, graphs with a fixed combinatorial embedding and a fixed outer face. Grids of sizes $(2n-4) \times (n-2)$, $(n-2) \times (n-2)$, and $\lfloor 2(n-1)/3 \rfloor \times (4\lfloor 2(n-1)/3 \rfloor - 1)$ were shown to be sufficient for straight-line drawings of plane graphs by de Fraysseix, Pach, and Pollack [4], Schnyder [8], and Chrobak and Nakano [3], respectively.

Very little is known when the combinatorial embedding and the outer face of the graphs can be changed. Namely, while area upper bounds for plane graphs trivially extend to planar graphs, there is, as far as we know, no non-trivial lower bound for the area required by planar graphs. In particular, one could ask whether the family of nested triangles graphs, used to show the lower bound for plane graphs, is a good candidate for providing a lower bound in the variable embedding setting. Thus determining the area requirements of nested triangles

* Work partially supported by EC - Fet Project DELIS - Contract no 001907 and by MUR under Project "MAINSTREAM: Algoritmi per strutture informative di grandi dimensioni e data streams".

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 339–344, 2007.

graphs when the outer face can be changed is regarded as one among the most challanging and interesting problems for the Graph Drawing community [2].

In this paper, we show an algorithm to produce straight-line drawings of nested triangles graphs in $(\frac{n}{3}+3)\times(\frac{2n}{3}+6)=\frac{2}{9}n^2+O(n)$ area (Section 3). Such a bound improves the previous best bound of $(\frac{n}{2})\times(\frac{2n}{3})=\frac{n^2}{3}$, due to Ossona de Mendez [7]. In the same section we show that any planar straight-line drawing of a nested triangles graph requires $n^2/9+\Omega(n)$. This result provides, as far as we know, the first non-trivial lower bound for the area required by straight-line drawings of planar graphs in the variable embedding setting. Section 2 contains some background and Section 4 contains our conclusions and some open problems.

2 Preliminaries

A *straight-line grid drawing* of a graph (or *drawing*, for short) is a mapping of each vertex to a distinct point of the plane with integer coordinates and of each edge to a segment between its endpoints. The *bounding box* of a drawing Γ is the smallest rectangle with sides parallel to the axes that completely covers Γ. The *height* (*width*) of Γ is the height (width) of its bounding box. The *area* of Γ is the height of Γ multiplied by its width.

A *planar drawing* is such that no two edges intersect except, possibly, at common endpoints. A planar drawing of a graph determines a circular ordering of the edges incident to each vertex. Two drawings of the same graph are *equivalent* if they determine the same circular ordering around each vertex. A *planar embedding* is an equivalence class of planar drawings. A planar drawing partitions the plane into topologically connected regions, called *faces*. The unbounded face is the *outer face*. Two equivalent drawings of the same graph may differ for the outer face. A graph together with a planar embedding and a choice for its outer face is called *plane graph*. A *k-connected* graph G is such that removing any $k-1$ vertices leaves G connected. A 3-connected (or triconnected) planar graph G admits a unique planar embedding. Hence, different plane graphs can be obtained from G only by choosing different outer faces.

Let t_1 and t_2 be two disjoint 3-cycles of a graph G, and let Γ be a planar drawing of G. We say that t_2 is *nested into* t_1 in Γ if t_2 is drawn in the bounded region of the plane delimited by t_1. We denote such a relationship by $t_1 > t_2$. A *nested triangles graph* G with n vertices (n is a multiple of 3) is a triconnected graph admitting a planar drawing Γ in which $n/3$ disjoint triangles $t_1, t_2, \ldots, t_{n/3}$ can be found such that $t_1 > t_2 > \ldots > t_{n/3}$. A nested triangles graph is *maximal* if all its faces are triangles.

Property 1. Let Γ be any planar drawing of a graph G, and let t_1 and t_2 be two disjoint 3-cycles of G such that $t_1 > t_2$ in Γ. The height (width) of t_1 in Γ is at least two units bigger than the height (width) of t_2.

Based on Property 1, given an n-vertex nested triangles graph G any drawing of G such that $t_1 > t_2 > \ldots > t_{n/3}$ requires $(\frac{2n}{3}-1)\times(\frac{2n}{3}-1)$ area [5,4]. Such a bound can be achieved with a drawing like the one represented in Fig. 1(a).

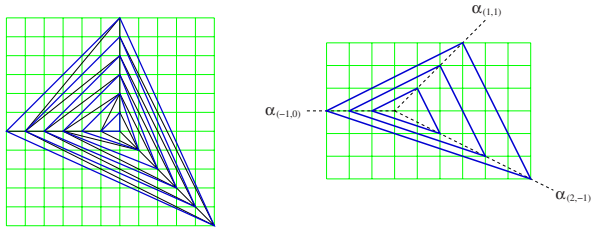


Fig. 1. (a) A minimum area drawing of a maximal nested triangles graph. (b) Construction of a three-semi-axes drawing.

Given two integer numbers a and b, the half-line starting at the origin and passing through vertices $(k \cdot a, k \cdot b)$, for any positive integer k, is denoted $\alpha_{(a,b)}$. Let $\alpha_{(a,b)}$, $\alpha_{(c,d)}$, and $\alpha_{(e,f)}$ be three half-lines such that the angle determined by any two consecutive half-lines is less than π. Given an n-vertex nested triangles graph G admitting a planar drawing in which $n/3$ disjoint triangles $t_1, t_2, \ldots, t_{n/3}$ can be found such that $t_1 > t_2 > \ldots > t_{n/3}$, a *three-semi-axes drawing* Γ of G with axes $\alpha_{(a,b)}$, $\alpha_{(c,d)}$, and $\alpha_{(e,f)}$ is a planar straight-line drawing obtained by suitably placing the vertices of t_i on points $(i \cdot a, i \cdot b)$, $(i \cdot c, i \cdot d)$, $(i \cdot e, i \cdot f)$, with $1 \leq i \leq n/3$. Fig. 1(b) shows an example of construction of a three-semi-axes drawing of a nested triangles graph. Observe that, since from any vertex of triangle t_i a straight-line segment can be drawn to any vertex of triangle t_{i+1} without introducing intersections, a three-semi-axes drawing is always planar.

3 Area Bounds for Nested Triangles Graphs

The following lemma proves the claimed upper bound.

Lemma 1. *Every n-vertex nested triangles graph admits a planar straight-line drawing in $\frac{2}{9}n^2 + O(n)$ area.*

Proof: To prove the statement we restrict to maximal graphs, since any non-maximal nested triangles graph can be augmented to maximal by adding dummy edges. Consider any maximal nested triangles graph G admitting a planar drawing Γ^* in which $n/3$ disjoint triangles $t_1, t_2, \ldots, t_{n/3}$ can be found such that $t_1 > t_2 > \ldots > t_{n/3}$. We show how to construct a drawing Γ of G in $\frac{2}{9}n^2 + O(n)$ area. Observe that, since G is triconnected, Γ and Γ^* have the same plane embedding (up to a reversal of their adjacency lists) with the exception, possibly, of the choice of the outer face. Also, since n is a multiple of 3, either n is also multiple of 6 (n is even) or n is not multiple of 6 (n is odd).

Suppose n is even. Consider the subgraph G' of G induced by the vertices of $t_{n/6}$ and $t_{n/6+1}$. We label the vertices of $t_{n/6}$ with labels v_1, v_2, and v_3, and those of $t_{n/6+1}$ with v_4, v_5, and v_6. Such a labeling is based on the degree and the adjacencies of the vertices in G'. Two are the cases, either all vertices of G'

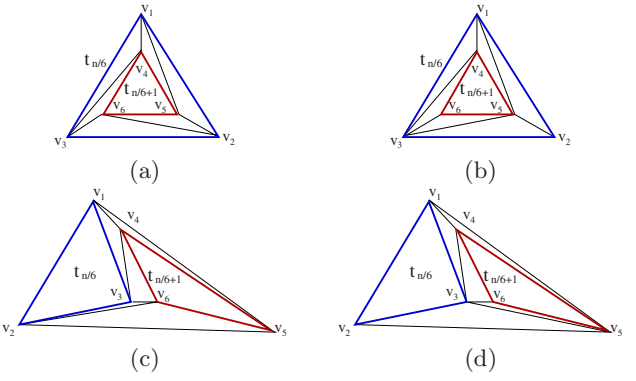


Fig. 2. Cases for the proof of Lemma 1

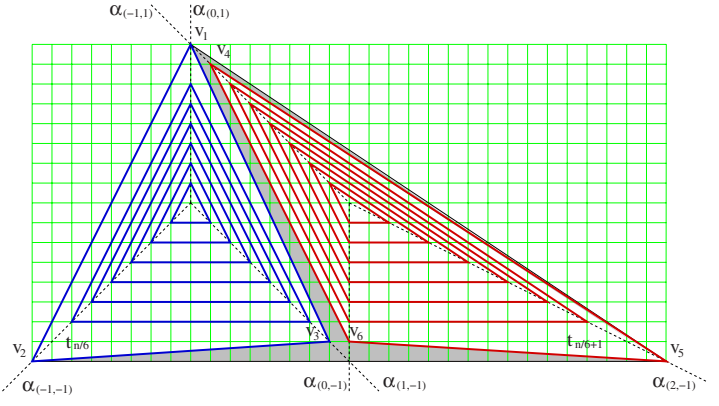


Fig. 3. A drawing of a nested triangles graph with optimal $\frac{2}{9}n^2 + O(n)$ area

have degree four, or not. In the first case, label the vertices of G' as depicted in Fig. 2(a). In the second case, label the vertices of G' as depicted in Fig. 2(b).

Choose as outer face of Γ the face incident to v_1, v_2, and v_5. Due to the choice of the outer face, Γ contains two sequences S_1 and S_2 of nested triangles, with $S_1 : t_{n/6} > t_{n/6-1} > \ldots > t_1$ and $S_2 : t_{n/6+1} > t_{n/6+2} > \ldots > t_{n/3}$ (see Fig. 3). Separately construct the drawings of S_1 and S_2 as follows. For S_1 construct a three-semi-axes drawing Γ_{S_1} with axes $\alpha_{(0,1)}$, $\alpha_{(-1,-1)}$, and $\alpha_{(1,-1)}$, such that v_1, v_2, and v_3 lie on axes $\alpha_{(0,1)}$, $\alpha_{(-1,-1)}$, and $\alpha_{(1,-1)}$, respectively. Further, shift v_1 and v_2 to the next available grid point along $\alpha_{(0,1)}$ and $\alpha_{(-1,-1)}$, respectively. For S_2 construct a three-semi-axes drawing Γ_{S_2} with axes $\alpha_{(-1,1)}$, $\alpha_{(2,-1)}$, and $\alpha_{(0,-1)}$, such that v_4, v_5, and v_6 lie on axes $\alpha_{(-1,1)}$, $\alpha_{(2,-1)}$, and $\alpha_{(0,-1)}$, respectively. Also, shift vertex v_5 to the next available grid point along $\alpha_{(2,-1)}$. Drawings Γ_{S_1} and Γ_{S_2} are combined in such a way that v_3 is one unit to the left of v_6 (see Fig. 3).

We now show that Γ is a planar drawing of G with area $\frac{2}{9}n^2 + O(n)$. First, observe that shifting vertices v_1, v_2, and v_5 does not compromise the planarity of Γ_{S_1} and Γ_{S_2}. Second, observe that, independently of the degrees of $v_1, \ldots, v_6$ in G', the edges between vertices of $t_{n/6}$ and vertices of $t_{n/6+1}$ do not intersect (see Figs. 2(c) and 2(d)). Concerning the area of Γ, we have that the height of Γ is equal to the height of Γ_{S_1}, which is $2\frac{n}{6}+2$, while the width of Γ can be obtained by summing the horizontal lengths of edges (v_2, v_3), (v_3, v_6), and (v_6, v_5), yielding $(2\frac{n}{6}+1)+1+(2\frac{n}{6}+2)$. Hence, the area of Γ is $(\frac{n}{3}+2)\times(2\frac{n}{3}+4) = \frac{2}{9}n^2 + O(n)$.

If n is odd, we add a triangle $t_0 > t_1$ in Γ^*, and arbitrarily augment Γ^* to a maximal nested triangles graph by adding dummy edges. Applying the construction described above we obtain a drawing with area $(\frac{n+3}{3}+2)\times(2\frac{n+3}{3}+4) = \frac{2}{9}n^2 + O(n)$. □

Next, we prove a lower bound on the area needed by any straight-line drawing of a nested triangles graph when the outer face is not fixed.

Lemma 2. *Every n-vertex nested triangles graph requires $\frac{n^2}{9} + \Omega(n)$ area in any planar straight-line drawing Γ.*

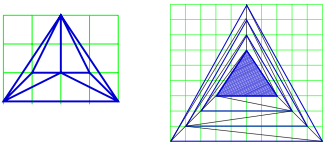
Proof: Consider any nested triangles graph G admitting a planar drawing Γ^* in which $n/3$ disjoint triangles $t_1, t_2, \ldots, t_{n/3}$ can be found such that $t_1 > t_2 > \ldots > t_{n/3}$. Suppose that n is a multiple of 6. Choose any face f to be the outer face of Γ. Three are the cases: (i) $f = t_1$; (ii) $f = t_{n/3}$; and (iii) f is contained into the region of the plane delimited by t_k and t_{k+1} in Γ^*, for some $k \in \{1, \ldots, n/3-1\}$. In Cases (i) and (ii), we have $n/3$ disjoint nested triangles in any planar drawing of G and, by Property 1, $(\frac{2n}{3}-1)^2 = \frac{4}{9}n^2 + \Omega(n)$ area is required. In Case (iii), we have that in any planar drawing Γ of G there are two sequences S_1 and S_2 of nested triangles such that $S_1 : t_{k+1} > t_{k+2} > \ldots > t_{n/3}$ and $S_2 : t_k > t_{k-1} > \ldots > t_1$. One between S_1 and S_2 has at least $n/6$ nested triangles and, by Property 1, $(2\frac{n}{6}-1)^2 = \frac{n^2}{9} + \Omega(n)$ area is required. □

4 Conclusions and Open Problems

In this note we have shown that $2n^2/9 + O(n)$ area is sufficient and that $n^2/9 + \Omega(n)$ area is necessary to construct straight-line planar drawings of nested triangles graphs. Closing the gap between the upper and the lower bound is a natural open question. We conjecture the following:

Conjecture 1. Any maximal nested triangles graph requires $2n^2/9 + \Omega(n)$ area in any straight-line planar grid drawing.

Our conjecture is motivated by the following considerations: Choosing an arbitrary outer face for a maximal nested triangles graph composed of $n/3$ nested triangles produces two sequences S_1 and S_2 of h and $n/3-h$ nested triangles, respectively. Such sequences lie in disjoint regions of the plane. Hence, it is possible to find a lower bound for the convex hull area of the whole drawing by summing up the convex hull areas of S_1 and S_2. It appears to be the case that the area of the convex hull of h nested triangles is at least $2h^2 + \Omega(h)$ (the statement is



(a) (b)

Fig. 4. (a) A plane graph and a minimum-area straight-line drawing of it. (b) Construction of a plane graph requiring $4n^2/9 - 2n/3$ area.

trivial when one side of the external triangle is parallel to one of the axes). Provided that the above statement is true, we can show that the minimum convex hull area of the whole drawing is $n^2/9 + \Omega(n)$, obtained for $h = n/6$. This would imply that the minimum area of the bounding box is $2n^2/9 + \Omega(n)$.

Renown problems in this field are those of determining the area required by straight-line drawings of general planar and plane graphs. Concerning the latter, in [6,9,1] it is reported the long-standing conjecture that the nested triangles graph is a worst case, i.e., that any plane graph can be drawn in $\lceil 2n/3 - 1 \rceil \times \lceil 2n/3 - 1 \rceil$ area, which, for $n \equiv 0 \bmod 3$, gives $4n^2/9 - 4n/3 + 1$. We remark that such a bound neglects at least a linear area term. Consider, in fact, the six-vertex graph G shown in Fig. 4(a). By case study we can prove that the smaller drawings of G have 2×6 and 3×4 bounding-boxes. The graph obtained by nesting G into $n/3 - 2$ nested triangles (see Fig. 4(b)) has at least $(2(n/3 - 2) + 3) \times (2(n/3 - 2) + 4) = (2n/3 - 1) \times (2n/3) = 4n^2/9 - 2n/3$ area.

References

1. Bonichon, N., Felsner, S., Mosbah, M.: Convex drawings of 3-connected plane graphs. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 60–70. Springer, Heidelberg (2005)
2. Brandenburg, F.J., Eppstein, D., Goodrich, M.T., Kobourov, S.G., Liotta, G., Mutzel, P.: Selected open problems in graph drawing. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 515–539. Springer, Heidelberg (2004)
3. Chrobak, M., Nakano, S.-I.: Minimum-width grid drawings of plane graphs. Comp. Geom. (11), 29–54 (1998)
4. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10(1), 41–51 (1990)
5. Dolev, D., Leighton, F.T., Trickey, H.: Planar embedding of planar graphs. Advances in Computing Research - vol. 2. VLSI Theory (1984)
6. Miura, K., Nakano, S., Nishizeki, T.: Grid drawings of 4-connected plane graphs. Discr. Comp. Geom. (26), 73–87 (2001)
7. de Mendez, P.O.: Some problems: Grid drawings of planar graphs, http://www.ehess.fr/centres/cams/person/pom/langen/openpb.html
8. Schnyder, W.: Embedding planar graphs on the grid. In: Proc. SODA 1990, pp. 138–148 (1990)
9. Zhang, H., He, X.: Compact visibility representation and straight-line grid embedding of plane graphs. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 493–504. Springer, Heidelberg (2003)

Universal Sets of n Points for 1-Bend Drawings of Planar Graphs with n Vertices*

Hazel Everett[1], Sylvain Lazard[1], Giuseppe Liotta[2], and Stephen Wismath[3]

[1] LORIA, INRIA Lorraine, Nancy Université, Nancy, France
{Hazel.Everett,Sylvain.Lazard}@loria.fr
[2] Dip. di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia
liotta@diei.unipg.it
[3] Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Alberta, Canada
wismath@cs.uleth.ca

Abstract. This paper shows that any planar graph with n vertices can be point-set embedded with at most one bend per edge on a universal set of n points in the plane. An implication of this result is that any number of planar graphs admit a simultaneous embedding without mapping with at most one bend per edge.

1 Introduction

Let S be a set of m distinct points in the plane and let G be a planar graph with n vertices ($n \leqslant m$). A *point-set embedding* of G on S is a planar drawing of G such that each vertex is drawn as a point of S and the edges are drawn as poly-lines. The problem of computing point-set embeddings of planar graphs has a long tradition both in the graph drawing and in the computational geometry literature (see, e.g., [5,6,8]). Considerable attention has been devoted to the study of universal sets of points. A set S of m points is said to be *h-bend universal* for the family of planar graphs with n vertices ($n \leqslant m$) if any graph in the family admits a point-set embedding onto S that has at most h bends along each edge.

Gritzman, Mohar, Pach and Pollack [5] proved that every set of n distinct points in the plane is 0-bend universal for the all outerplanar graphs with n vertices. De Fraysseix, Pach, and Pollack [3] and independently Schnyder [9] proved that a grid with $O(n^2)$ points is 0-bend universal for all planar graphs with n vertices. De Fraysseix et al. [3] also showed that a 0-bend universal set of points for all planar graphs having n vertices cannot have $n + o(\sqrt{n})$ points. This last lower bound was improved by Chrobak and Karloff [2] and later by Kurowski [7] who showed that linearly many extra points are necessary for a 0-bend universal set of points for all planar graphs having n vertices. On the other

* Research supported by NSERC and the MIUR Project "MAINSTREAM: Algorithms for Massive Information Structures and Data Streams". Work initiated during the "Workshop on Graph Drawing and Computational Geometry", Bertinoro, Italy, March 2007. We are grateful to the other participants, and in particular to W. Didimo and E. Di Giacomo, for useful discussions.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 345–351, 2007.

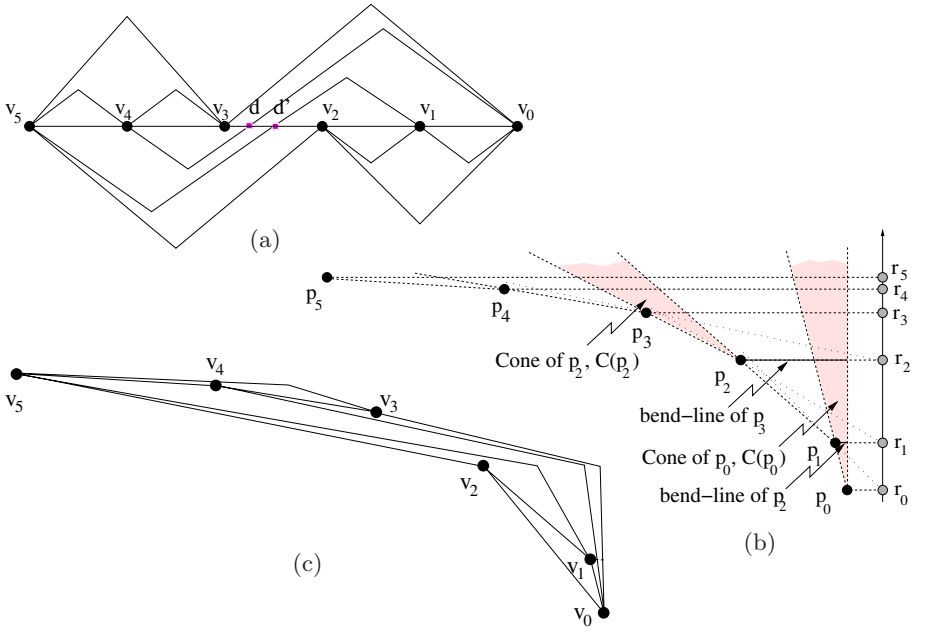


Fig. 1. (a) A proper monotone topological book embedding. The spine crossings d and d' are proper. (b) A necklace of six points. The cone of p_0, the cone of p_2, the bend-line of p_2, and the bend-line of p_3 are highlighted. (c) A point-set embedding computed by Algorithm 1-`bend Universal Drawer`.

hand, if two bends along each edge are allowed, a tight bound on the size of the point-set is known: Kaufmann and Wiese [6] proved that every set of n distinct points in the plane is 2-bend universal for all planar graphs with n vertices.

In this paper we study the minimum size of a universal set of points for all planar graphs with n vertices under the assumption that at most one bend per edge is allowed in the point-set embedding. We prove the following theorem.

Theorem 1. *Let $\mathcal{F}_n$ be the family of all planar graphs with n vertices. There exists a set of n distinct points in the plane that is* 1*-bend universal for $\mathcal{F}_n$.*

The proof is constructive; an example is shown in Figure 1. We define a set S of n points and show how to compute an embedding of any planar graph with n vertices on S such that the resulting drawing has at most one bend per edge. The drawing procedure starts by computing a special type of book embedding defined in Section 2, and then uses this book embedding to construct the point-set embedding with the algorithm described in Section 3.

Our universal set of n points can be defined either (i) with algebraic coordinates such that they are the vertices of a convex chain with unit-length edges or (ii) on a regular grid of size $n2^n$ by n. In the former case all planar graphs of $\mathcal{F}_n$ can be drawn on that point set with all bend-points and vertices in a square

of size n by n at distance at least $\frac{1}{2d}$ apart, where d is the maximum degree of the graph. In the latter case, the graphs can be drawn with all bend-points on the grid points of the $n2^n$ by n grid.

We conclude this introduction by noting a result that is immediately implied by Theorem 1. Two planar graphs G_1 and G_2 with the same set of vertices are said to admit a *simultaneous embedding without mapping* if there exists a set of points in the plane that supports a point-set embedding of both G_1 and G_2 [1]. It is not known whether any two planar graphs admit a simultaneous embedding without mapping such that all edges are straight-line segments. A consequence of [5] is that a planar graph has a straight-line simultaneous embedding without mapping with any number of outerplanar graphs. A consequence of [6] is that any two planar graphs have a simultaneous embedding without mapping such that each edge is drawn with at most two bends. Theorem 1 implies the following.

Corollary 1. *Any number of planar graphs sharing the same vertex set admit a simultaneous embedding without mapping with at most one bend per edge.*

2 Monotone Topological Book Embeddings

Consider the Cartesian coordinate system (O, x, y) and let p, q be two points in the plane. We say that p is *left of* q and we denote it as $p < q$ if the x-coordinate of p is less than the x-coordinate of q; we shall also use the notation $p \leqslant q$ to mean that either p is left of q or p coincides with q; we define similarly $p > q$ and $p \geqslant q$. A *spine* is a horizontal line. Let ℓ be a spine and let p, q be two points of ℓ. Let $p < q$ and let b be a point of the perpendicular bisector of $\overline{pq}$, at positive distance from ℓ. An *arc* connecting p to q, denoted as (p, q), is a polygonal chain consisting of two segments: segment $\overline{pb}$ and segment $\overline{bq}$. Point p is the *left endpoint* of (p, q), point q is the *right endpoint* of (p, q), and point b is the *bend-point* of (p, q). Arc (p, q) can be either in the half-plane above the spine or in the half-plane below the spine (such half-planes are assumed to be closed sets); in the first case we say that the arc is in the *top page* of ℓ, otherwise it is in the *bottom page* of ℓ. From now on, when we denote an arc as (p, q) we shall implicitly assume that p is its left endpoint.

Let $G = (V, E)$ be a planar graph. A *monotone topological book embedding* of G, denoted Γ, is a planar drawing such that all vertices of G are represented as points of a spine ℓ and each edge is either represented as an arc in the bottom page, or as an arc in the top page, or as a poly-line that crosses the spine and consists of two consecutive arcs. Let $e = (u, v)$ be an edge of a monotone topological book embedding that crosses the spine at a point d; assuming that u is left of v along the spine, e is such that: (i) $u < d < v$, (ii) arc (u, d) is in the bottom page, and (iii) arc (d, v) is in the top page. Point d is called the *spine crossing* of (u, v). Refer to Figure 1(a). Also, let u' be the rightmost vertex along the spine of Γ such that $u' < d$ and let v' be the leftmost vertex of the spine of Γ such that $d < v'$. We say that u' and v' are the two *bounding vertices* of d. We say that d is a *proper spine crossing* if its bounding vertices u' and v' are such that $u < u' < d < v' < v$. (The spine crossings d and d' of Figure 1(a)

are both proper and both bounded by v_3 and v_2). A monotone topological book embedding is *proper* if all of its spine crossings are proper. Di Giacomo et al. [4] proved that, for every planar graph, a monotone topological book embedding exists and can be computed (in linear time in the size of the graph). Since an edge that crosses the spine with a non-proper spine crossing can be replaced by a single arc, we obtain the following lemma.

Lemma 1. *Every planar graph has a proper monotone topological book embedding which can be computed in linear time in the size of the graph.*

Let now Γ be a proper monotone topological book embedding of a planar graph G. If we insert a dummy vertex for each spine crossing of Γ, we obtain a new topological book embedding Γ' such that Γ' represents a planar subdivision G' of G obtained by splitting with a vertex some of the edges of G. We call the graph G' an *augmented form* of G and the drawing Γ' an *augmented topological book embedding* of G. A vertex of G' that is also a vertex of G is called a *real vertex* of Γ'; a vertex of G' that corresponds to a spine crossing of Γ is called a *division vertex* of Γ'. Note that every division vertex of Γ' has degree two and that every edge of Γ' is either an arc in the top page or an arc in the bottom page. The *bounding vertices of a division vertex* d of Γ' are the two real vertices that form the bounding vertices of the spine crossing corresponding to d in Γ. The following property is a consequence of the planarity of Γ'.

Property 1. Let $a = (u, v)$ and $a' = (u', v')$ be two distinct arcs of Γ' that are in the same page and such that $u < u'$. Then, (i) $u < v \leqslant u' < v'$ or (ii) $u < u' < v' \leqslant v$.

3 Proof of Theorem 1

We prove Theorem 1 by first defining a family of sets of n points in convex position (Subsection 3.1) and then by describing an algorithm that computes a point-set embedding of any planar graph with n vertices on the n-point element of the family (Subsection 3.2).

3.1 Necklaces, Cones, and Bend-Lines

Let p_0 be any point on the x-axis strictly left of O and p_1 be any point strictly in the top-left quadrant of p_0. We construct p_{i+2}, for $0 \leqslant i \leqslant n-2$, from p_i and p_{i+1} as follows. Let r_i be the projection of p_i on the vertical y-axis. Point p_{i+2} can be chosen anywhere on or below the line through r_i and p_{i+1} and strictly above the horizontal line through p_{i+1}. Let S be any set of n points defined by the above procedure; we call S a *necklace* of n points. See Figure 1(b).

The *cone of* p_0, denoted as $C(p_0)$, is the wedge with apex p_0 and bounded by the vertical half-line above p_0 and by the ray emanating from p_0 and through p_1. The *cone of* p_i $(1 \leqslant i \leqslant n-2)$, denoted as $C(p_i)$, has p_i as its apex and is bounded by two rays emanating from p_i with directions $\overrightarrow{p_{i-1}p_i}$ and $\overrightarrow{p_ip_{i+1}}$. In what follows we assume that $C(p_i)$ is an open set $(0 \leqslant i \leqslant n-1)$.

The *bend-line* of p_i $(i > 1)$ is the relatively-open horizontal segment from p_{i-1} to the vertical line through p_0. The following properties follow from the definition of a necklace and can be proved with elementary geometric arguments. Let $S = \{p_0, p_1, \dots p_{n-1}\}$ be a necklace of n points and let $CH(S)$ be its convex hull. Note that $p_0, \dots, p_{n-1}$ are ordered from right to left, *i.e.*, $p_{n-1} < \dots < p_0$.

Property 2. Let $p_h < p_t$ $(t > 1)$ be two points of S and let q be a point on the bend-line of p_t. Segments $\overline{p_h q}$ and $\overline{p_t p_{t-1}}$ intersect in their relative interior.

Property 3. Let $p_{h'} \leqslant p_h < p_t$ $(t > 1)$ be three points of S and let $q' < q$ be two points on the bend-line of p_t. Segments $\overline{p_h q}$ and $\overline{p_{h'} q'}$ do not intersect each other.

3.2 Computing 1-Bend Point-Set Embeddings

We describe a drawing algorithm, called 1-`bend Universal Drawer`, that receives as input a planar graph G with n vertices and a necklace S of n points and returns a point-set embedding of G on S such that every edge of G is drawn with at most one bend. Algorithm 1-`bend Universal Drawer` consists of the following steps.

Step 1: Compute a proper monotone topological book embedding Γ of G and the corresponding augmented proper topological book embedding Γ'. Let ℓ be the spine of Γ'. Label the real vertices of Γ' on ℓ by $v_{n-1}, \dots, v_0$ in that order from left to right (*i.e.*, $v_i < v_{i-1}$). Map each real vertex v_i to point p_i of the necklace $(0 \leqslant i \leqslant n-1)$.

Step 2: Draw the bends of the arcs of the top page of Γ' as follows. For each vertex v_i of Γ' mapped to point p_i $(0 \leqslant i \leqslant n-1)$ do the following. Let $a_{i0}, a_{i1}, \cdots, a_{i(k-1)}$ be the sequence of arcs in the top page of Γ' whose right endpoint is v_i; assume that $a_{i0}, a_{i1}, \cdots, a_{i(k-1)}$ are encountered in this order when going clockwise around v_i by starting the tour from a point on ℓ slightly to the left of v_i. For each a_{ij} $(0 \leqslant j \leqslant k-1)$ do:

- Draw a ray r_{ij} emanating from p_i such that: (i) r_{ij} is inside the cone $C(p_i)$ of p_i, and (ii) $r_{i(j+1)}$ is to the right of r_{ij} $(0 \leqslant j \leqslant k-2)$.
- Let v_h be the left endpoint of a_{ij} in Γ' and b_{ij} the bend-point of a_{ij}. If v_h is a real vertex of Γ', draw b_{ij} at the intersection point, q, between r_{ij} and the bend-line of p_h (through p_{h-1}).[1] Else, if v_h is a division vertex of Γ' and the two real vertices bounding v_h in Γ' are v_t and v_{t-1}, draw b_{ij} at the intersection point, q, between r_{ij} and the bend-line of p_t.

Step 3: Draw the division vertices of Γ' as follows. For each division vertex d of Γ', do the following. Let (v_i, d) and (d, v_j) be the two arcs of Γ' sharing d

[1] If p_h and p_i are consecutive vertices of S $(h-1=i)$, the ray r_{ij} and the bend-line of p_h do not intersect, though their closures intersect at p_i. For consistency, we draw b_{ij} at this intersection point $q = p_i$. In Step 4, the arc (v_h, v_i) is drawn as the poly-line consisting of segment $\overline{p_h q}$ followed by $\overline{q p_i}$, which is reduced to point p_i.

such that (v_i, d) is in the bottom page and (d, v_j) is in the top page. Let q be the point computed in Step 2 such that q represents the bend of (d, v_j). Draw d at the intersection point between $\overline{p_i q}$ and $CH(S)$.

Step 4: Draw the arcs of Γ' as follows. For each arc (u, v) of Γ' do the following. Let p_u, p_v be the points representing u and v along $CH(S)$.

- If (u, v) is an arc in the bottom page, draw it as the chord $\overline{p_u p_v}$.
- If (u, v) is an arc in the top page of Γ', let q be the point computed at Step 2 that represents the bend-point of (u, v). Draw (u, v) as the poly-line consisting of segment $\overline{p_u q}$ followed by $\overline{q p_v}$.

Step 5: Let $\hat{\Gamma}$ be the drawing computed at the end of Step 4. Compute a drawing of G by removing from $\hat{\Gamma}$ those points that represent the division vertices of Γ'.

The proof of Theorem 1 is now completed by showing that Algorithm 1-`bend Universal Drawer` correctly computes a point-set embedding of G on S such that each edge has at most one bend. The idea is to show that the drawing computed at the end of Step 5 maintains the topology of Γ and that the geometric properties of the proper monotone topological book embedding and of the necklace make it possible to point-set embed the graph without edge-crossings and with at most one bend per edge. In particular, we show that $\hat{\Gamma}$ is a planar drawing by exploiting Properties 1-3; the proof is however omitted here due to lack of space.

Observe that every real vertex of Γ' is drawn as a point of S in $\hat{\Gamma}$. Since $\hat{\Gamma}$ does not have edge crossings, removing the division vertices from $\hat{\Gamma}$ gives a point-set embedding of G on S. Also, by construction, the two edges incident on a division vertex of $\hat{\Gamma}$ form a flat angle, and thus removing the division vertices from $\hat{\Gamma}$ does not increase the number of bends. It follows that the drawing computed by Algorithm 1-`bend Universal Drawer` is a point-set embedding of G on S such that each edge has at most one bend. Therefore, any necklace of n vertices is a 1-bend universal set for all planar graphs having n vertices, which concludes the proof of Theorem 1. We omit here the proofs on the size of the drawings.

4 Conclusion

We leave as an open problem to find a universal point-set for one-bend drawing of planar graphs in a polynomial-size regular grid.

References

1. Braß, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous planar graph embeddings. Comput. Geom. 36(2), 117–130 (2007)
2. Chrobak, M., Karloff, H.: A lower bound on the size of universal sets for planar graphs. SIGACT News 20(4), 83–86 (1989)

3. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10, 41–51 (1990)
4. Di Giacomo, E., Didimo, W., Liotta, G., Wismath, S.K.: Curve-constrained drawings of planar graphs. Computational Geometry 30, 1–23 (2005)
5. Gritzmann, P., Mohar, B., Pach, J., Pollack, R.: Embedding a planar triangulation with vertices at specified points. Amer. Math. Monthly 98(2), 165–166 (1991)
6. Kaufmann, M., Wiese, R.: Embedding vertices at points: Few bends suffice for planar graphs. Journal of Graph Algorithms and Applications 6(1), 115–129 (2002)
7. Kurowski, M.: A 1.235 lower bound on the number of points needed to draw all n-vertex planar graphs. Inf. Process. Lett. 92(2), 95–98 (2004)
8. Pach, J., Wenger, R.: Embedding planar graphs at fixed vertex locations. Graph and Combinatorics 17, 717–728 (2001)
9. Schnyder, W.: Embedding planar graphs on the grid. In: SODA 1990. Proc. 1st ACM-SIAM Sympos. Discrete Algorithms, pp. 138–148 (1990)

LunarVis – Analytic Visualizations of Large Graphs*

Robert Görke, Marco Gaertler, and Dorothea Wagner

Universität Karlsruhe (TH), Germany
{rgoerke,gaertler,wagner}@ira.uka.de

Abstract. The analysis and the exploration of complex networks nowadays involves the identification of a multitude of analytic properties that have been ascertained to constitute crucial characteristics of networks. We propose a new layout paradigm for drawing large networks, with a focus on decompositional properties. The visualization is based on the general shape of an annulus and supports the immediate recognition of a large number of abstract features of the decomposition while drawing all elements. Our layouts offer remarkable readability of the decompositional connectivity and are capable of revealing subtle structural traits.

1 Introduction

Current research activities in computer science and physics aim at understanding the structural characteristics of large and complex networks such as the Internet [1,2], networks of protein interactions [3,4], social networks [5] and many others. A multitude of laws of evolution and scaling phenomena have been investigated [6,7], alongside studies on community structure, e.g. [8], and traditional network analyses [9]. Heavily relying on mathematical models and abstract characteristics, many of these techniques highly benefit from, or even depend on feasible advance information about structural properties of a network, in order to properly guide or find starting points for an analysis. Adequate visualization methods for complex networks are a crucial step towards such advance information. Furthermore, due to the diversity of such analyses, customized visualizations concentrating on user defined characteristics are required.

Along the lines of the more general issue in the field of information visualization, see e.g. [10], visualizations of large networks naturally suffer a trade-off between the level of detail and the visible amount of information. In other words, a detailed representation of a graph often antagonizes the immediate perceptibility of abstract analytic information. In this work we propose a layout paradigm that tackles the task of detailed analytic visualizations for large graphs. Our approach incorporates the strengths of abstract layouts, while individually placing all nodes and edges, i.e. without hiding away potentially crucial details. The

* The authors gratefully acknowledge financial support from the European Commission within FET Open Projects DELIS (contract no. 001907) and the DFG under grant WA 654/13-3.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 352–364, 2007.

general underlying shape of the layout is a (partial) annulus. Subgraphs, defined by a decomposition, are then individually molded into annular segments. The annulus has been chosen for three primary reasons, first, it offers immediate readability of hierarchies and decompositional characteristics. Second, it allows for an insightful segment-internal layout, and third, it provides a large area for the drawing of edges, permitting the perception of segment connectivity at a glance, which is a major focus of many applications.

The technique works in three phases. In the first, abstract phase, a network decomposition determines the general shape of the layout, defining and arranging the drawing bounds of each annular segment. The second phase initializes the drawing of individual nodes and the third phase determines the final layout by means of sophisticated force-directed methods. Our paradigm offers many degrees of freedom that can incorporate any desired analytic property, allowing for well readable simultaneous visualizations of complementary properties. Simple user parameters tune the focus of our visualizations to either inter- or intra-segment characteristics, and furthermore permit a scalable trade-off between the overall quality and the required computational effort.

This paper is organized as follows. Sect. 2 sets our work into the context of related work. Then, after giving some definitions and notation in Sect. 3, we present our new layout paradigm in Sect. 4. An empirical study, using real world examples of the physical Internet, collaboration graphs and road networks, is given in Sect. 5. Finally, we conclude the paper in Sect. 6 with a brief summary.

2 Foundations and Previous Work

In the past, several layout techniques have been developed driven by the ambitious goal to properly visualize complex networks such as the Autonomous Systems (AS) network. Two important approaches are the landscape metaphor [11] and network fingerprinting [12], examples of which are shown in Fig. 1 and Fig. 2, respectively. Introduced by Baur et al., the former modifies a conventional layout

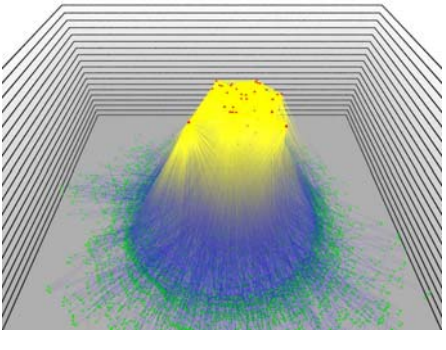


Fig. 1. A 2.5-dimensional layout (*landscape metaphor*) of the AS network [11]

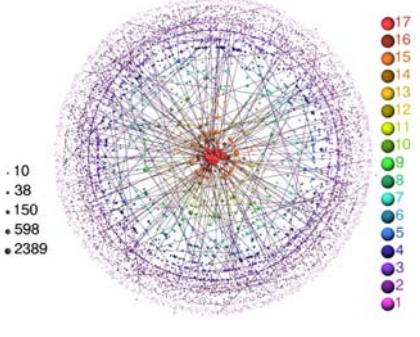


Fig. 2. A fingerprint of the AS network made with LaNet-vi [12]

technique by a framework of constraints that are based on analytic properties. The global shape of the network is induced by the position of structurally important elements, which automatically conceal inferior parts. Thus, it reflects the 'landscape' of importance, either in two or three dimensions. The latter approach, LaNet-vi [12] uses analytic properties to define the global shape, which consists of concentric rings of varying thickness, one for each level of the *core*-decomposition (see Sect. 3). Then, nodes are placed within these bounds and the overall readability is achieved by showing only a small sample of edges.

The above techniques and similar ones have succesfully been applied in numerous tasks, serving as an aide in network analyses. The method we present in the following synergizes assets of previous approaches and remedies a number of shortcomings in order to provide a layout technique that fingerprints a network (as LaNet-vi), but adds to this a much clearer visual realization of a number of analytic properties, thus offering a high informative potential. Before describing our visualization technique, we discuss the necessary preliminaries.

3 Preliminaries

Let $G = (V, E)$ be an undirected graph. We call a partition $P = \{P_0, \ldots, P_k\}$ of the set V of nodes a *decomposition* with *shells* P_i. Furthermore, a *nested decomposition* H is a nesting of subsets V_i of V such that $(V = V_0 \supseteq V_1 \supseteq \cdots \supseteq V_k \neq \emptyset)$. The sets V_i of H are called *layers*, giving rise to the *height* of H being k and the *height* of a node v, defined as the index i such that $v \in V_i \setminus V_{i+1}$. The partition P_H induced by a nested decomposition H is canonically defined as $P_H = \{V_0 \setminus V_1, V_1 \setminus V_2, \ldots, V_k\}$. Edges between or within shells are canonically called *inter-* or *inter-shell* edges.

The choice of suitable network decompositions primarily depends on the field of application. In this work we focus on four different exemplary decompositions, *k-cores*, *clustering*, *reach* and *betweenness centrality*. The concept of *k-cores* was originally introduced in [13]. Stated in a procedural definition, the *i-core* of a graph is the unique subgraph obtained by iteratively removing all nodes of degree less than i, thus k-cores constitute a hierarchical decomposition. Graph clusterings commonly capture large scale inhomogeneities by grouping nodes into clusters by some formalization of the paradigm of *intra-cluster density* versus *inter-cluster sparsity*, see [9] for an overview. In the following we use a well known *modularity*-based graph clustering technique [8]. The betweenness centrality of a node is, roughly speaking, the number of shortest paths passing through it [14], reach is a similar concept used in transportation networking [15]. These decompositions are highly relevant in the analysis of large networks, such as protein network analyses [3] recommendation networks [8] and social sciences.

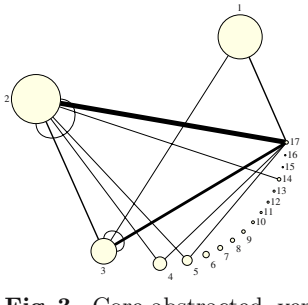


Fig. 3. Core-abstracted version of an AS graph

Visualizations of large networks usually suffer a trade-off between the details of visually shown elements and the amount of represented information. Widely known concepts resolving this are *abstraction*, as in Fig. 3 and the *reduction* of data to specific shells or parts of interest, illustrated in Fig. 4. While abstracted visualizations offer the best readability of these properties, much detail is lost, as in Fig. 3.

In contrast, zoomed visualizations as in Fig. 4 allow for the exploration of small scale subgraphs and structural subtleties. We overcome this compromise by using the layout of an abstracted graph as a blueprint but still draw all elements. Our goal is the visualization of all nodes and edges in a manner both pleasing and informative on intra shell characteristics, in addition to revealing the characteristics of the given hierarchical decomposition. We focus on properties like the size of shells and the connectivity within and between shells.

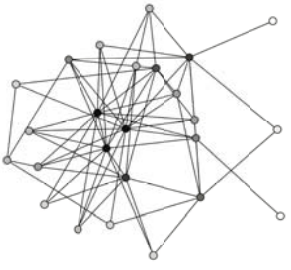


Fig. 4. Reduction of the 16-shell of Fig. 3

4 The Layout Technique

In the following we detail our construction technique for LunarVis. Roughly speaking, our approach divides up into three distinct phases, the first of which sets out the abstract layout attributes of the annular layout, such as the number of segments, their dimension and their placement. Based on these, a heuristic computation of suitable parameters follows, which will then be employed in the third and last step. This last, and by far the most intricate and computationally demanding step can be regarded as an iterative, segment-wise application of spring forces. These forces determine the final placement of each single node based on neighborhood attraction and repulsion both inside and between segments. In the end, we scale the annulus to the desired angular range and radial spreading and finally draw edges as straight lines with a high degree of transparency. Optionally, the size of a node and its color may serve as additional dimensions of information, yet ample use of these potentially overburdens a visualization. Algorithm 1 gives an overview of these three phases, which we describe in detail in the following sections.

4.1 Abstract Attributes

By any means, the informative potential of the our technique heavily relies on a suitable rough, abstract layout. We propose as the general underlying shape of the visualization an annulus, as shown in Fig. 5. The shells s_i are lined up along a predefined angular range (here a full circle), placing the bottom (s_1) and the top shell (s_8) at the extremes. Thus, shells correspond to annular segments. User-defined properties then determine the individual dimensions of these segments, namely the *angular width* α_i and the *radial extent* r_i. In order to increase readability, small gaps β_i that separate neighboring segments can be included.

Algorithm 1. LunarVis

Input: Graph $G = (V, E)$
Output: LunarVis Layout
1 Initialize abstract layout
2 Calculate parameters. Initialize random node placement within segments
3 **for** $i = 1, \ldots, \ell_{out}$ **do**
  forall *shells* s **do**
    Project layout of s to middle square $\overline{s}$
    for $k = 1, \ldots, \ell_{inter}$ **do**
      Apply inter-shell forces $\overline{s}$
    for $j = 1, \ldots, \ell_{intra}$ **do**
      Apply intra-shell forces to $\overline{s}$
    Project new layout of $\overline{s}$ to annular segment s
4 Finalize and scale annulus, draw transparent edges, color and resize nodes

The underlying annulus has an inner radius r_{in} and an outer radius r_{out}, which, together with the angular range, define the total drawing area.

In our experiments, setting the annular segments to touch the inner rim and sizing them such that the largest shell also touches the outer rim, offered the best readability. For consistency, we let the number of nodes per shell define the angular width and the number of intra-shell edges define the radial extent throughout this paper, since these properties are generally of immediate interest. Molded into the underlying shape of annular segments, the shells can now be layouted individually. To give an impression of this step, and to point out the utility of an additional scaling function for the abstract layout, Fig.6 shows three layouts of the same network, using different scaling functions for the radial extent and the angular width of a shell. As canonic scaling functions, we used the strictly growing functions *square root* and *logarithm*. The network is a snapshot of the AS

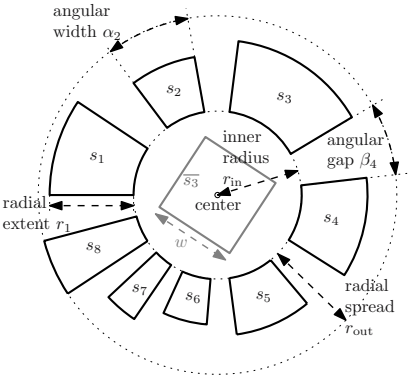


Fig. 5. The LunarVis annulus

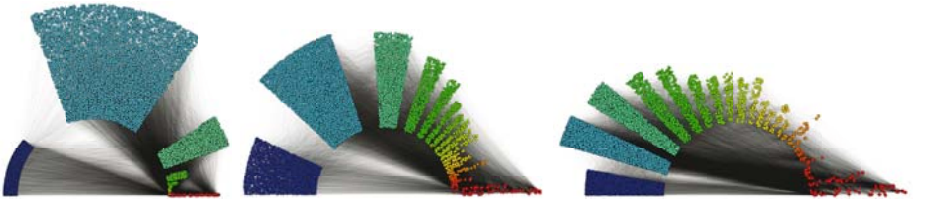


Fig. 6. Visualizations of the AS (1st March, 2005) using different scaling options. Radial/angular scaling is linear/linear (left), log/sqrt (middle), log/log (right).

network, decomposed into its core hierarchy. Individual nodes are left with a random placement, and the total angle is π. Linear scaling enables the immediate comparison of sizes, however, large values overshadow more subtle variations that do not become obvious without a logarithmic scaling of the radial extent. The inter-shell edge distribution is revealed by logarithmically scaling angular widths. Next, we describe how individual nodes are placed. For the sake of a better understanding we describe our parameter settings afterwards in Sect. 4.3.

4.2 Force-Directed Node Placement

Placing the individual nodes is by far the most computationally demanding task. Simple strategies offer an easy recognition of the shells' shapes, however, more sophisticated techniques can additionally reveal the internal structure of the shells while requiring more time and storage. Based on the forces proposed by Fruchterman and Reingold [16] we use spring- and repulsion forces to iteratively have the nodes of each shell adjust their position as suggested by their adjacencies and. In the following we describe this procedure in detail.

As sketched out in Alg. 1, our layout algorithm cycles through all shells a set number (ℓ_{out}) of times by line 3. The nodes of a shell are then first subjected to inter-shell spring forces (ℓ_{inter} repetitions), thus moving towards their inter-shell adjacencies, and then, as a relaxational step, to intra-shell forces (ℓ_{intra} repetitions). To this end, we maintain a mapping of each shell, i.e. annular segment s_i, to a square $\overline{s_i}$ of size $w = 2/3 \cdot r_{\text{in}}$, centered at the origin and rotated such that it faces its original annular segment, see Figure 5. Forces are applied to the copies of nodes in the square $\overline{s_i}$, and then, the new coordinates of nodes in $\overline{s_i}$ are mapped back to the annular segment s_i and its nodes are moved accordingly. Note that nodes in s_i themselves exert inter-shell forces on their copies in $\overline{s_i}$.

Figures 7 and 8 illustrate the intention of this approach. First, note that a node coordinate $(x_{\overline{v}}, y_{\overline{v}})$ in a square shaped working copy $\overline{s_i}$ is obtained by transforming the circle coordinates (ρ_v, ϕ_v) in the annular segment s_i in a canonical way, such that the angular position ϕ_v of v within s_i is linearly mapped to the the x-coordinate $x_{\overline{v}}$ within $\overline{s_i}$, and the radial position ρ_v to $y_{\overline{v}}$. The rotation of $\overline{s_i}$ then aligns the y-axis of $\overline{s_i}$ with the middle axis (ϕ_{mid}) of s_i.

The crucial idea behind this setup is that inter-shell forces pull nodes towards a specific side of the square, thus indicating their linkage tendency, while intra-shell forces relax

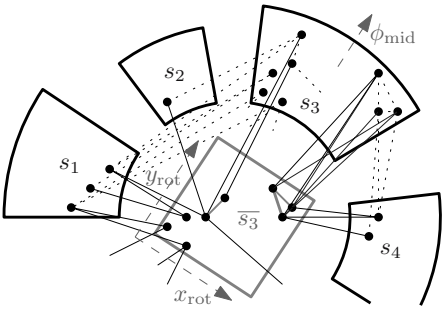


Fig. 7. Forces for $\overline{s_3}$ (excerpt). Inter-shell forces are caused by edges that link $\overline{s_3}$ with segments (solid, black). Intra-shell forces are attraction and repulsion of nodes within $\overline{s_3}$. Dotted edges are irrelevant during this stage.

the resulting crowding and unmask community structure and disconnected components. In Fig. 7, inter-shell forces draw the triangle of nodes in the right of $\overline{s_3}$ towards s_3 and s_4, while the nodes on the left, primarily being linked to other shells are pulled towards s_1, s_2 and other adjacencies. The subsequent application of intra-shell forces will keep the triangle grouped and separated, and relax the disconnected nodes on the left.

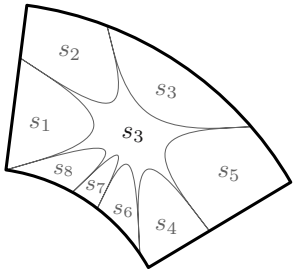


Fig. 8. Sketch of the preferred node locations in s_3

The areas of s_3 in Fig. 8 roughly sketch out where nodes, with a majority of adjacencies in shells as indicated, are drawn by inter-shell forces, before intra-shell forces relax the layout. The size and placement of these areas are induced by the abstract layout of the annular segments, see Fig. 5 for comparison. This segmentation of each shell allows for a sophisticated interpretation of a node's position.

Needless to say, we augmented our force-based algorithms with several well known techniques, such as soft clipping [16] to guarantee containment within shells, sentinel nodes that uncrowd segment borders [16] and an increased sluggishness of nodes with high degree [17]. However, (anti-)gravitational forces as well as simulated annealing [18], a randomized node ordering or an impulse history [17] yielded no substantial increase in quality, since our technique does not aim at a highly optimized local layout. We apply a simple exponential cooling, such that the movement of nodes is increasingly slowed. This proved necessary to avoid stubborn oscillations, especially if intra-shell forces are used purely relaxational.

An important observation is, that applying inter- and intra-shell forces at the same time naturally encourages force equilibria, but does not allow for a structurally targeted analysis. On the contrary, the separate application of inter- and intra-shell forces allows for a user-defined emphasis on either shell-internal properties or global connectivity.

4.3 Parameters

Heuristic or experimental assessment of parameters is inevitable when using customized force-directed methods. We base our forces on those proposed by Fruchterman and Reingold [16]. Alternative force models as proposed e.g. by Eades [19] or Frick et al. [17] did not prove more suitable but increased the running time, partly due to the fact that we do not enforce equilibria.

For intra-shell forces we set the base spring length to $C_i \cdot \sqrt{(\text{area}/\#\text{ vertices})}$, with the factor C_i boosting the intra-shell spring length of dense shells. Depending on the decomposition, global factors for repulsion forces and spring lengths between 1 and 1.5 and 1.2 and 1.5, respectively, worked best. In fact, these two parameters were the only ones that required adjustment. Our inter-shell forces work with a base spring length of half the inner radius. Both the spring length and the spring force hardly needed additional tuning. Moreover, setting the edge

length w of the squares $\overline{s_i}$ to significantly smaller values than $2/3 \cdot r_{\text{in}}$ blurred inter-shell forces, while much larger values exaggerated their range of effect.

As mentioned above, the iteration counters ℓ_{out}, ℓ_{inter} and ℓ_{intra} are pure user parameters, since these govern the interaction and the emphasis of intra-shell and inter-shell aspects. In fact, surprisingly low iteration numbers already yields very nice results, a good starting point are $\ell_{\text{out}} = 10$, $\ell_{\text{inter}} = 10$, $\ell_{\text{intra}} = 5$. In the majority of drawings we used the logarithm for most scalings, as it copes best with power-law distributions and generally dampens overshadowing maxima.

5 Results

In the following, we present a selection of LunarVis visualizations, all offering many immediate insights. Nevertheless, knowledge about the drawing process, i.e. how nodes are placed, allows for a more structurally oriented interpretation.

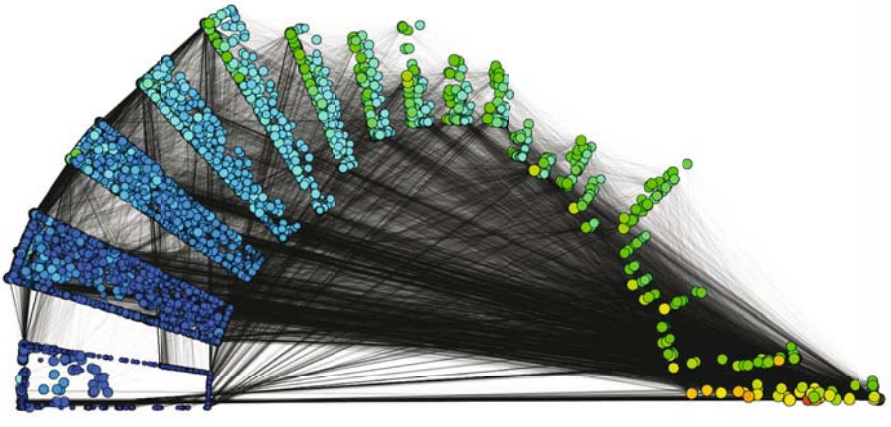


Fig. 9. A snapshot of the AS network taken at the 01.01.2006, decomposed by k-cores. Nodes with a high (low) degree are colored blue (red) and the area of a node is proportional to its betweenness centrality (all on a logarithmic scale). We chose a half circle for the total angular range and set the maximum shell at the right end.

Figure 9 reveals numerous characteristics of the core decomposition of the AS network at a glance. The well investigated fact that all shells primarily link to the core is immediate, alongside the observation that the internal communities of the first five shells are well interconnected (connectivity near outer rim), but not those of other shells. To name a few subtle facts visible in this drawing, note that mid-degree nodes can already be found in the 3-shell, that nodes with low betweenness are exclusively found in low shells while the opposite is not true, and that in low- to mid-shells nodes with higher degrees primarily link to lower shells, as they sit on the upper left. We used a time-sequence of such visualizations for an analysis of the temporal evolution of the AS network.

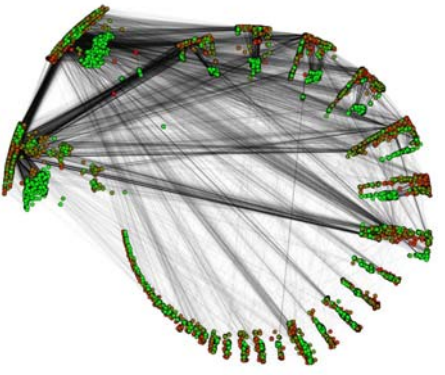


Fig. 10. The AS network, decomposed by a clustering. Nodes with a high (low) betweenness are colored red (green).

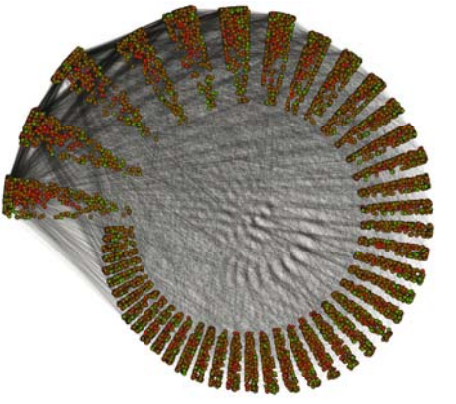


Fig. 11. A network created with BRITE [20], designed to emulate the AS. All parameters are as in Fig. 10.

For Fig. 10 and 11 a full annulus has been chosen due to the high number of shells (56 and 45). Figure 10 diplays the AS network, decomposed by community structure that has been identified by a greedy modularity based clustering algorithm [8]. The clusters are sorted by size. Figure 11 shows the same decomposition for a topology with the same number of nodes and edges, created with BRITE [20], an AS topology simulator. Quite clearly, BRITE fails to feature any of the peculiarities the AS network exhibits, such as high inhomogeneity in community sizes, the large number of tiny clusters or the fact, that most shells are almost exclusively connected to the two largest shells. An analysis yields clustering coefficients of 0.002 and 0.375 for BRITE and the AS network, respectively, and transitivities of 0.011 and 0.001, which agrees with these observations.

Figure 12 illustrates the core decomposition of an email network. The nodes represent computer scientists at Universität Karlsruhe, color coded by their department and sized by their betweenness, and edges are email contacts over the past eight months. As an exception, we used the sum of degrees for the radial extent with a square-root scaling for this LunarVis layout. From the multitude of observable features we point out the fact that community structure within departments is

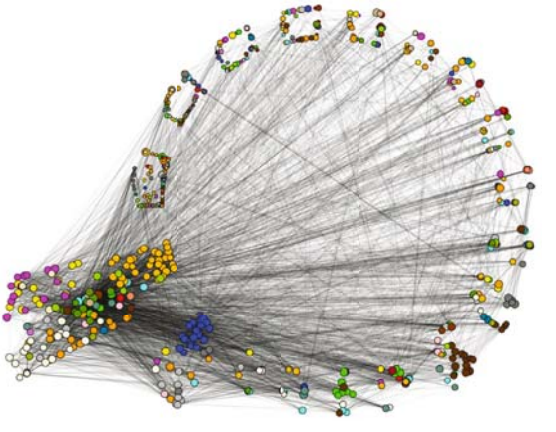


Fig. 12. Email network of the computer science department at Universität Karlsruhe

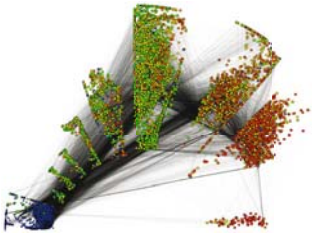


Fig. 13. Luxembourg roads, decomposed by betweenness, color indicates reach

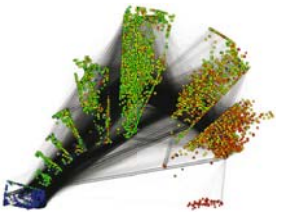


Fig. 14. München roads, decomposed by betweenness, color indicates reach

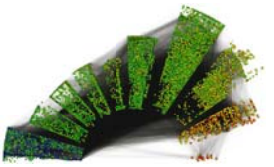


Fig. 15. European railroads, decomposed by betweenness, color indicates reach

Fig. 16. Luxembourg roads, decomposed by reach, color indicates betweenness

Fig. 17. München roads, decomposed by reach, color indicates betweenness

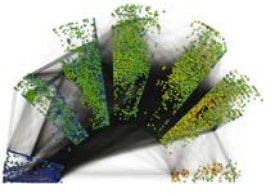


Fig. 18. European railroads, decomposed by reach, color indicates betweenness

been corroborated by the groupings in the top cores. As an example, the dark blue department, although being well interconnected (gathered), seems to have many contacts to lower shells, thus it sits at the inner rim of core 17.

Modern algorithms for route planning exploit numerous characteristics of road graphs for efficient shortest path computations, for an overview see e.g. [21]. Figures 13-17 display road maps of the Czech Republic and of the city of Munich, provided by PTV AG for scientific use, and Figures 15-18 display the European network of railway connections, provided by HAFAS. On the left hand side betweenness centrality [9], indexed into eleven logarithmically scaled intervals, served as the decomposition, and the figures on the right hand side are decomposed by reach centrality [15], colors are used vice versa. The stunning similarity of all corresponding drawings indicate that transportation networks share strong characteristics with respect to both reach and betweenness. However, several details can be observed that reflect intrinsic differences between these networks. Towards a taxonomy for transportation networks we can immediately observe that the railway network has very few hubs, both with respect to betweenness and reach. These are mainly capitals that, additionally, have exceptionally high degrees. The general correlation between reach and betweenness (color versus shell index) corroborates the fact that railroads constitute a scale-free network. This does not apply to either road network, which is due to the fact that road

networks tend not to have unique shortest paths – recall Munich's surrounding autobahn and Luxembourg's rural nature. The road networks strongly resemble each other, however, obvserve that in Munich, nodes of both maximum (autobahn segments) and minimum (residential dead-end streets) betweenness have a rather small degree. This cannot be observed in Luxembourg, where only nodes of minimum betweenness have an exceptionally small degree. From the facts revealed by the edge connectivity, note that hardly any peripheral nodes are adjacent to nodes of maximum centrality.

For computing our drawings, we used one core of an AMD Opteron 2218 processor clocked at 2.6 GHz, with 1 MB of L2 cache, running SUSE Linux 10.1. Our non-optimized development implementations in Java required drawing times between a few seconds and several hours, depending on the chosen number of iterations and the size of the network.

6 Conclusion

LunarVis is a new paradigm for drawing large graphs with a grand informative potential. Through sophisticated utilization of force directed drawing techniques and the neat design of an apt global shape, our technique creates visualizations of networks that reveal analytic properties of decompositions alongside properties of the shell connectivity at a glance, on the one hand, and offer insights into the interior characteristics of shells on the other hand. An emphasis on either inter- or intra-adjacencies can easily be adjusted.

The scope of application of LunarVis reaches far beyond mere network fingerprinting, as it does not only produce a distinct visual representation of a network but in fact offers the immediate recognition of analytic properties and unmasks structural characteristics and peculiarities. The transparent visualization of the set of inter-shell edges within the spacious interior of the annulus is particularly suitable for analyses on shell connectivity. LunarVis, however, is not a tool for investigating small-scale substructures or for purely esthetic, energy-minimal drawing.

Our results yield that LunarVis is highly feasible and informative in fields of application as diverse as internet studies, route planning and social sciences, employing decompositions by centrality, clustering and k-cores. The dimensions offered for analytic information surpass many existing visualization techniques in terms of perceptibility and detail, while the layout is highly configurable by using different scaling functions, emphasizing either intra- oder inter-shell relationships or simply plugging in complementary analytic properties.

The name of our paradigm *LunarVis* has been inspired by the semblance of our visualizations to the shape of the moon, sometimes waxing, sometimes full, but always a nice sight.

Acknowledgements

We thank Ignacio Alvarez-Hamelin for profound discussions and brainstorming on LunarVis. Moreover, we thank Daniel Delling and Reinhard Bauer for

valuable discussions and data on transportation networks and Bastian Katz for helpful assistance in programming and documentation.

References

1. Pastor-Satorras, R., Vespignani, A.: Evolution and Structure of the Internet: A Statistical Physics Approach. Cambridge University Press, NY, USA (2004)
2. Chen, Q., Chang, H., Govindan, R., Jamin, S.: The origin of power laws in internet topologies revisited. In: Proceedings of INFOCOM 2002, vol. 2, pp. 608–617. IEEE, Los Alamitos (2002)
3. Wuchty, S., Almaas, E.: Peeling the yeast protein network. Proteomics 5(2), 444–449 (2005)
4. Jeong, H., Mason, S.P., Barabási, A.L., Oltvai, Z.N.: Lethality and Centrality in Protein Networks. Nature 411 (2001) (brief communications)
5. Ducheneaut, N., Yee, N., Nickell, E., Moore, R.J.: "Alone Together?": Exploring the Social Dynamics of Massively Multiplayer Online Games. In: CHI 2006. Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 407–416. ACM Press, New York (2006)
6. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graphs over time: densification laws, shrinking diameters and possible explanations. In: KDD 2005. Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, pp. 177–187. ACM Press, New York (2005)
7. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science 286, 509–512 (1999)
8. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Phys. Rev. E 70 (066111) (2004)
9. Brandes, U., Erlebach, T. (eds.): Network Analysis. LNCS, vol. 3418. Springer, Heidelberg (2005)
10. Ware, C.: Information visualization: perception for design. Morgan Kaufmann Publishers Inc., San Francisco (2000)
11. Baur, M., Brandes, U., Gaertler, M., Wagner, D.: Drawing the AS Graph in 2.5 Dimensions. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 43–48. Springer, Heidelberg (2005)
12. Alvarez-Hamelin, J.I., Dall'Asta, L., Barrat, A., Vespignani, A.: Large scale networks fingerprinting and visualization using the k-core decomposition. In: NIPS (2005)
13. Seidman, S.B.: Network Structure and Minimum Degree. Social Networks 5, 269–287 (1983)
14. Freeman, L.C.: A Set of Measures of Centrality Based Upon Betweeness. Sociometry 40, 35–41 (1977)
15. Gutman, R.: Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In: 6th Workshop on Algorithm Engineering and Experiments, pp. 100–111 (2004)
16. Fruchterman, T.M.J., Reingold, E.M.: Graph Drawing by Force-directed Placement. Software - Practice and Experience 21(11), 1129–1164 (1991)
17. Frick, A., Ludwig, A., Mehldau, H.: A fast adaptive layout algorithm for undirected graphs. In: Tamassia, R., Tollis, I(Y.) G. (eds.) GD 1994. LNCS, vol. 894, pp. 388–403. Springer, Heidelberg (1995)

18. Davidson, R., Harel, D.: Drawing graphs nicely using simulated annealing. ACM Trans. Graph. 15(4), 301–331 (1996)
19. Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall, Englewood Cliffs (1999)
20. Medina, A., Lakhina, A., Matta, I., Byers, J.: BRITE: An Approach to Universal Topology Generation. In: Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Tele. (2001)
21. Wagner, D., Willhalm, T.: Speed-Up Techniques for Shortest-Path Computations. In: STACS. 24th International Symposium on Theoretical Aspects of Computer Science, pp. 23–36 (2007)

Visualizing Internet Evolution on the Autonomous Systems Level*

Krists Boitmanis**, Ulrik Brandes, and Christian Pich

Department of Computer & Information Science, University of Konstanz
Krists.Boitmanis@uni-konstanz.de

Abstract. We propose a visualization approach for large dynamic graph structures with high degree variation and low diameter. In particular, we reduce visual complexity by multiple modes of representation in a single-level visualization rather than abstractions of lower levels of detail. This is useful for non-interactive display and eases dynamic layout, which we address in the online scenario.

Our approach is illustrated on a family of large networks featuring all of the above structural characteristics, the physical Internet on the autonomous systems level over time.

1 Introduction

Visualization of large evolving relational data sets is a challenging task, because the size of the data and dynamics are difficult to deal with even in isolation. A visualization problem that encompasses these features simultaneously is the macroscopic view of the evolving Internet topology on the autonomous-systems (AS) level. To the best of our knowledge, there are no dynamic visualization approaches that can produce purely structure-based drawings of a sequence of AS graphs in reasonable time.

In this paper we propose to attack this problem by first applying a few complexity reduction operations, which lead to both considerably smaller graphs and savings of screen space. However, instead of hiding the less important parts of a graph, which is a common approach to reduce complexity, we still show them in the drawing with different representation modes. The reduced graphs are laid out with a stress majorization approach [14] enhanced with a novel scheme for calculating distances between nodes that is specially suited for graphs with extremely skew degree distributions. Also, the flexibility of the stress majorization technique allows to adapt it for the dynamic setting. This is demonstrated in the online scenario, where the previous drawing is respected during the layout for the next time point.

* Research partially supported by DFG under grant Br 2158/3-1 (ECRP). Part of this research was done while the first author was visiting the Institute of Mathematics and Computer Science, University of Latvia.

** Corresponding author.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 365–376, 2007.

The paper is structured as follows. In Sect. 2, we give a brief review of the AS-level Internet topology and related work. The layout method for static snapshots of the graph and our complexity reduction operations are the subject of Sect. 3 and the extension of this approach to dynamic graph visualization and its application to AS graphs are presented in Sect. 4. Section 5 concludes the paper with a short discussion.

2 AS-Level Internet Topology and Related Work

An *autonomous system*, or *AS* for short, is a group of computer networks typically under the same administrative authority, using the same routing policy. The Internet can thus be analyzed in terms of connections and interactions between ASes. The *AS graph* is then a model for the Internet, having ASes as nodes and AS-to-AS connections as edges.

In recent years, analysis of the AS-level Internet topology has attracted interest of many researchers. The common goal is to keep track of structure and dynamics of the Internet, to develop meaningful and robust models explaining such observations, and to come to reasonable interpretations. Technically and economically, the analysis has manifold practical aspects, e.g. for improving reliability, routing efficiency, and fairness.

Interest in the AS graph excelled when power-laws and scale-free distributions were observed to be characteristic features [12]. Since then, various aspects of autonomous systems have been investigated, such as inferring AS graphs from collected data [15], modeling and generating artificial AS graphs [16], and comparison of measured and generated data [23], to name just a few examples. The dynamics of the AS graph are analyzed in [13]; models for the AS graph evolution and a comparison of AS graph inference methods from different data sources are given in [18].

Visualization and visual analysis of AS graphs have been attempted as well, though to a lesser extent. Probably best known are the circular drawings from the Skitter project of CAIDA [9]. HERMES [7] is a system for orthogonal drawings of the Internet hierarchy or parts thereof. Force-directed generation of Internet maps is the approach taken in the Internet Mapping Project [8]. The two-and-a-half dimensional drawings of AS graphs in [3] are based on a hierarchy of increasingly denser cores, which is also used in [2]. Dynamics in the routing behavior of autonomous systems are visualized by LinkRank [17], animations for network performance assessment are described in [6]. To the best of our knowledge, only the layouts in [3] consider the complete AS graph and are purely structure-based.

A number of approaches for drawing general dynamic graphs have been proposed [5], but few principles and frameworks are prevalent [4,10,11].

As a test ground for the methods we developed, we have constructed AS graphs at various time points from the BGP (Border Gateway Protocol) route data available in the archives of the *Route Views* project [21]. The structure of each AS graph is inferred from a collection of AS paths consisting of a sequence of numbers. Two ASes are connected by an undirected edge if their numbers appear consecutively in at least one of the AS paths.

3 Static Layout and Complexity Reduction

Although our ultimate goal is to visualize a sequence of AS graphs, we first restrict ourselves to visualizing a single snapshot $G = (V, E)$.

3.1 Layout Method

We have chosen the stress majorization approach as the graph layout method [14]. This choice was motivated by the quality of the resulting drawings, the flexibility of the approach facilitating adaptations for the dynamic setting, existing speed-up techniques, and simplicity of implementation at least when the localized stress minimization is used. Note, however, that other methods with similar properties, e.g. variants of force-directed methods, could be used equally well.

The basic idea is an iterative minimization of the stress function

$$\text{stress}(X) = \sum w_{uv}(\|X_u - X_v\| - d_{uv})^2 \ , \tag{1}$$

where the sum extends over all unordered pairs of nodes $\{u, v\}$ in V. Here $X_v \in \mathbb{R}^2$ is the position of the node $v \in V$, d_{uv} is the ideal distance between the nodes u and v, which is usually the length of a shortest path in G, and w_{uv} is a non-negative weight allowing different pairs of nodes influence the stress measure differently. Weights $w_{uv} = d_{uv}^{-2}$ are a common choice.

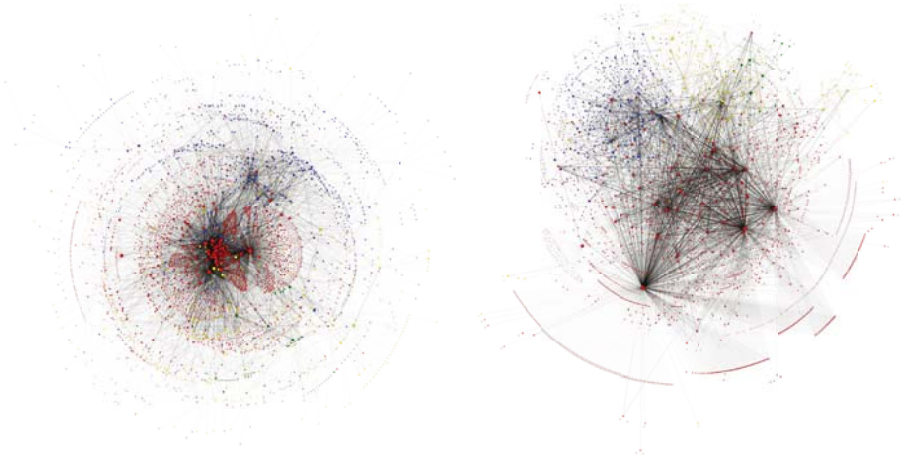
We can confirm the claim that the above strategy "makes the neighborhood of high degree nodes too dense" [14] unless appropriate lengths are assigned to edges (Fig. 1(a)). This is due to the extremely skewed degree distribution of AS graphs; the AS graph in Fig. 1 has 4271 nodes, 75% of which have a degree one or two, while a few extreme nodes have degrees as large as 924, 673, and 470. The problem is somewhat remedied if the geometric mean $\sqrt{d_u d_v}$ of the degrees of nodes u and v is used as the length of an edge $e = \{u, v\} \in E$, because then the high-degree nodes strive to push their neighbors further away (Fig. 1(b)). In Sect. 3.3 we propose a novel method for calculating distances that further improves the quality of drawings.

We use the following graphical conventions throughout the paper.

- The area of a node is proportional to the squared logarithm of its degree.
- The opacity of an edge is proportional to the radius of its smaller end-node. In effect, edges between high-degree nodes attract more attention of an observer.
- The nodes are colored according to the continents the corresponding ASes belong to: we use blue to represent Europe, red for North America, yellow for Asia, purple for South America, brown for Africa, and green for Oceania.

3.2 Visual Complexity Reduction

This section presents our attempts to allay the visual clutter of drawings by using different representation modes without loosing any information.



(a) (b)

Fig. 1. A snapshot of the AS graph in the year 1998 – (a) uniform edge length, (b) degree-dependent edge length

First, consider the typical AS graph in Fig. 1 with its many nodes of degree one. In a standard representation, these result in large fans that form dominant visual features that consume large areas but represent the least interesting structures. To remove this effect, we use radial clustergrams [1,20], a compact representations of trees, as follows:

- Let $T \subset V$ be the set of nodes in the attached trees of G, which can be obtained by an iterative removal of the leaves of G until all remaining nodes have degrees two or more.
- Draw the induced graph $G[V \setminus T]$ in the standard representation with nodes as circles and edges as straight lines.
- Draw the nodes of T as radial clustergrams around the nodes in $V \setminus T$ they are attached to.

Our radial clustergrams are slightly different from those in [1,20] to maintain the degree-area correlation. Suppose that the children $v_1, v_2, \ldots, v_k$ of a node v have to be drawn inside an annulus wedge with the radius r and the angle α (Fig. 2(a)). The desired area S_i of each node v_i is fixed because it is derived from its degree. Moreover, we require that the radial width w of the children of the same node is equal. Clearly, w cannot be less than $w_{\min} = \sqrt{\frac{2}{\alpha}\sum_{i=1}^{k} S_i + r^2} - r$. On the other hand, we would also like to avoid very thin nodes, so $l_i/w \leq c$ must hold for some constant $c > 0$, where l_i is the length of the outer arc of v_i. A possible solution to this inequality is given by the largest root w_i of the cubic equation $cw^3 + 2crw^2 - 2S_i w - 2S_i r = 0$, and consequently the common layer width for all children of v is calculated as $w = \max\{w_{\min}, w_1, w_2, \ldots, w_k\}$. Note, that the annulus wedge is not filled completely if $w > w_{\min}$ (Fig. 2(b,c)).

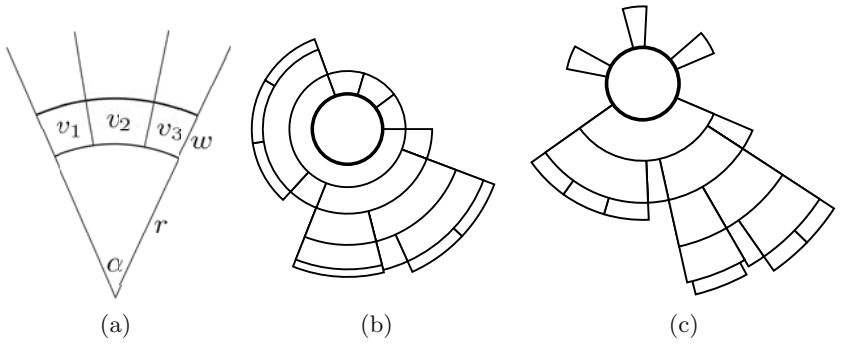


Fig. 2. (a) Children of the same node drawn in a specified annulus wedge. (b) A radial clustergram without restrictions on the radial width of nodes. (c) A radial clustergram of the same tree when the radial width of nodes is bounded from below.

Figure 3 shows a layout of the AS graph with the attached trees drawn as radial clustergrams. Although the clutter is somewhat reduced, there are still plenty of low-degree nodes around the periphery and many of them seem to be connected to the same set of core nodes. The latter is a structural feature that we emphasize by aggregating the equivalent nodes as follows.

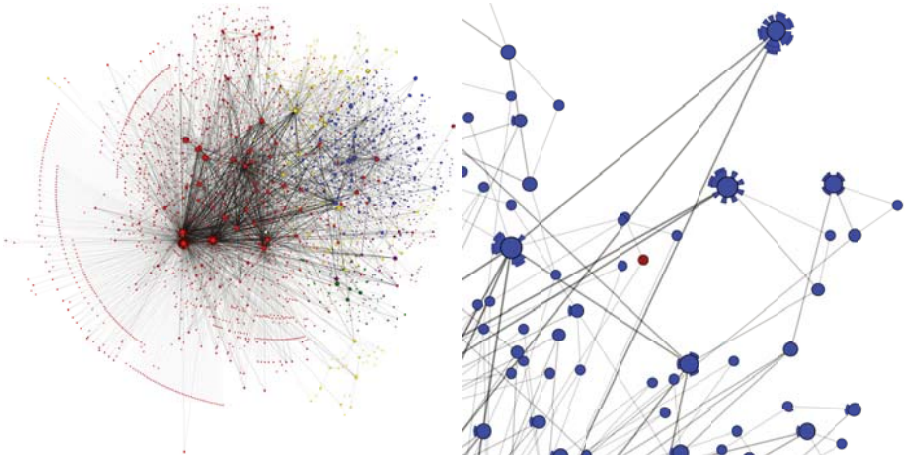
- Construct the equivalence classes of the relation $\{(u, v)|u, v \in V \setminus (T \cup N(T)) \wedge N(u) = N(v)\}$. Note that nodes with attached trees are considered as special and not equivalent to anything else.
- Contract each non-trivial equivalence class $U \subseteq V$ of this relation into a new meta-node v_U before applying the layout.
- After the position of a meta-node v_U has been determined by the layout algorithm, restore the equivalent nodes U and draw them around the position of v_U in a compact way. A good choice is the sunflower placement from [22,19].

As can be seen in Fig. 4(a), some sets of equivalent nodes are quite large and the compact placement shows their neighbors much better.

The final complexity reduction step consists of replacing maximal induced paths $(v_0, v_1, \ldots, v_k)$ by direct edges $\{v_0, v_k\}$ between their ends, provided that the inner nodes v_i $(0 < i < k)$ are not affected by the previous two reductions, i.e. $v_i \notin T \cup N(T) \cup M$, where M is the set of meta-nodes. After the layout of the reduced graph is calculated, the induced paths are restored and drawn straight between their ends (in the rare cases when two or more paths run between the same pair of end nodes, these paths are drawn parallel without mutual overlaps).

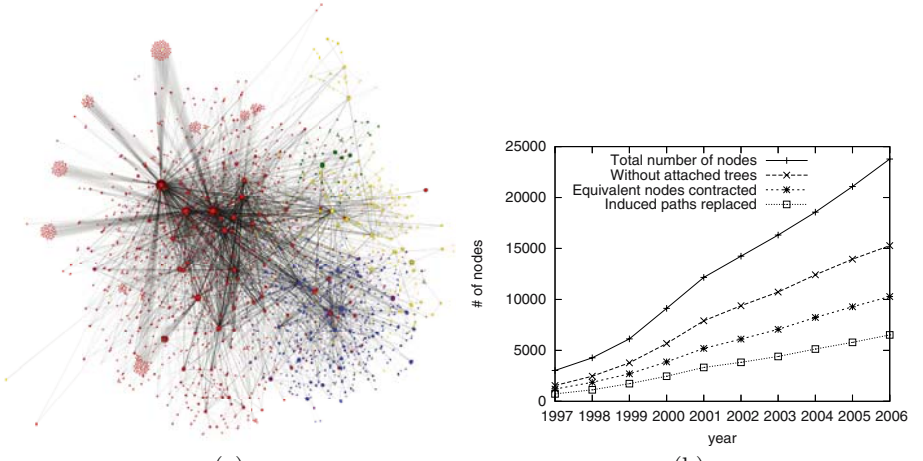
A side effect of these reduction operations is a lower number of nodes, which is a very significant advantage as the full stress majorization considers the distances between every pair of nodes. Figure 4(b) shows the growth of the AS graph over a decade and how many nodes remain after each reduction step.

In what follows, we assume that the graphs are reduced according to these three operations.



(a) (b)

Fig. 3. Full (a) and zoomed-in (b) drawings of the AS graph in the year 1998 with attached trees drawn as radial clustergrams



(a) (b)

Fig. 4. (a) The same AS graph after further complexity reductions. (b) The effect of the reduction operations on the number of nodes.

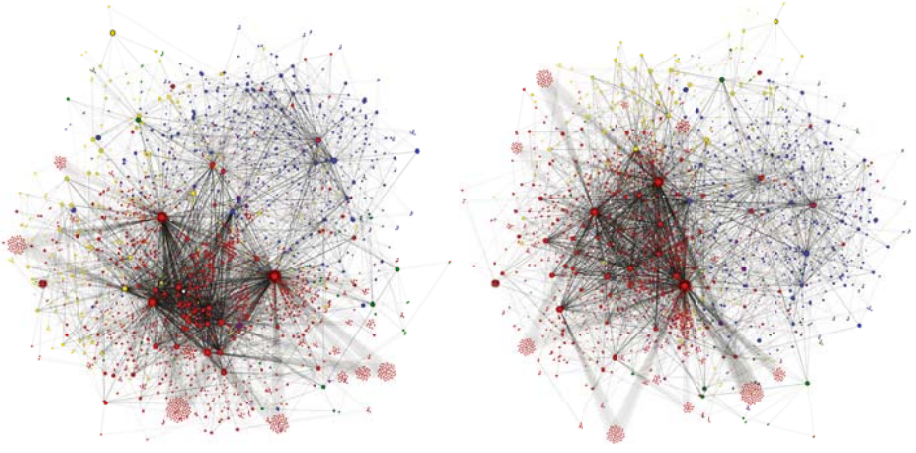
3.3 Layout Method – Revisited

The drawing in Fig. 4(a) leaves something to desire in terms of quality. First, the high-degree nodes are still placed too close to each other obscuring the structure of how they relate to the rest of the graph. Secondly, some low-degree nodes with only high-degree neighbors end up as peaks on the periphery because the length of their incident edges is unnecessarily high. A novel approach for calculating the pairwise distances and their weights solves both of these problems (Fig. 5(a)).

Edge Lengths. The importance of an edge $e = \{u, v\} \in E$ is captured better if its length l_e is an increasing function of the smallest degree $\min\{d_u, d_v\}$ of its ends. In our experiments the best results were obtained with $l_e = \ln(\min\{d_u, d_v\})$. In this way, adjacent nodes of high-degree are placed far apart and their connecting edge is more prominent. On the other hand, the incident edges of low-degree nodes are drawn much shorter so that these nodes are placed close to their neighbors.

Distances. Special care must be taken when calculating pairwise distances from these re-scaled edge lengths. We cannot simply use shortest paths in the weighted graph G, because two high-degree nodes are still very close if they have a common neighbor of low degree. Distances are therefore calculated as $d_{uv} = \max\{l(P)|P \in \mathrm{SUP}(u, v)\}$, where $\mathrm{SUP}(u, v)$ denotes the set of shortest paths between u and v in the unweighted graph G' underlying G and $l(P)$ is the length of the path P in the weighted graph G. In other words, we consider a longest weighted path among those with a minimum number of edges. Such distances can be easily calculated in $O(|V||E|)$ time by performing a breadth-first-search from each node $v \in V$ and determining the longest weighted paths in the shortest paths dag with source v. Also, the unweighted distances $d_{G'}(u, v)$ should be used when calculating the weights in (1), i.e. $w_{uv} = d_{G'}(u, v)^{-2}$, because otherwise the important distances would be outweighed by less important ones. An exception to this rule are the meta-nodes representing groups of equivalent nodes. If two meta-nodes u and v have a common neighbor, we use $w_{uv} = 1$ rather than $1/4$ to make it less likely that the resulting sunflowers would overlap. Moreover, the "degree" of a meta-node v_U representing a set U of equivalent nodes is assumed to be $\sum_{v \in U} d_v$ such that it represents the total "importance" of all nodes in U.

Speed-Up. The final modification of the method concerns its running time. It took 25 minutes to create a drawing of an AS graph having 23,779 nodes and 49,706 edges on a computer with 2 GHz CPU and 2 GB of memory, which is largely due to the use of the full distance matrix. Fortunately, the method can be sped up without affecting layout quality considerably (compare the two drawings in Fig. 5). The idea is to calculate the layout in two phases. First, a small subset of nodes $P \subseteq V$ with the highest degrees is chosen as pivots (we used 200 pivots in our experiments), and these are laid out in the above technique according to the distances d_{uv}, $u, v \in P$. In order to position the nodes in $V \setminus P$, we again utilize stress majorization, but fix pivots and ignore all distances d_{uv}, $u, v \notin P$ unless $\{u, v\} \in E$. In this way, we ignore a very large number of "inessential" distances, and the running time drops from 25 minutes to 44 seconds. It should be noted that this approach is slightly different from the sparse stress approach of [14], although they are similar in that the overall structure of the drawing is determined by some important core nodes, and other nodes are laid out based on distances to those core nodes and nodes in some close neighborhood. The main difference lies in the two applications of the stress majorization, which leads to the pivots being placed independently from the rest of the graph. This two-phase technique turned out to be more successful in our setting.



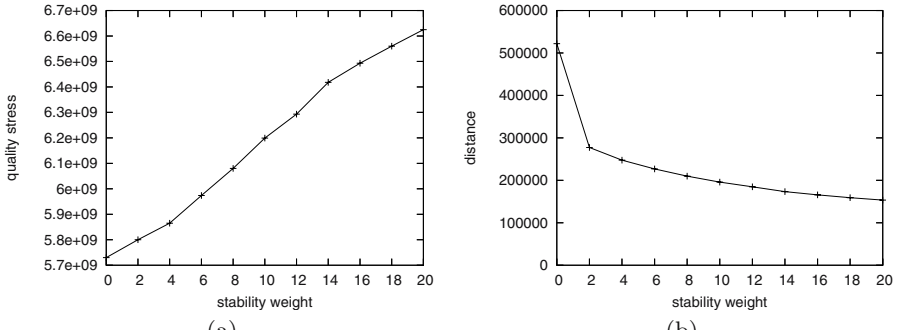
(a) (b)

Fig. 5. Drawings of the same AS graph obtained by the full stress majorization using the modified distances (a) and the fast two-phase method (b)

4 Dynamic Layout

In this section we will modify the above method to be applicable to dynamic graphs in the online scenario, i.e. when an existing drawing of the graph is respected during the creation of a subsequent drawing.

Suppose that besides the graph $G = (V, E)$ we are given the desired positions $p_v \in \mathbb{R}^2$ for nodes v in a subset $U \subseteq V$, which are the result of a preceding layout. In order to preserve the overall view of the evolving graph, we have an additional criterion now to minimize the distance of nodes from their desired positions. Fol-



(a) (b)

Fig. 6. The effect of the stability parameter on the quality of the drawing (a) and the total movement of nodes (b) when the online method is applied to the AS graph in the year 1998. The desired positions are obtained from the layout of the graph at the year before.

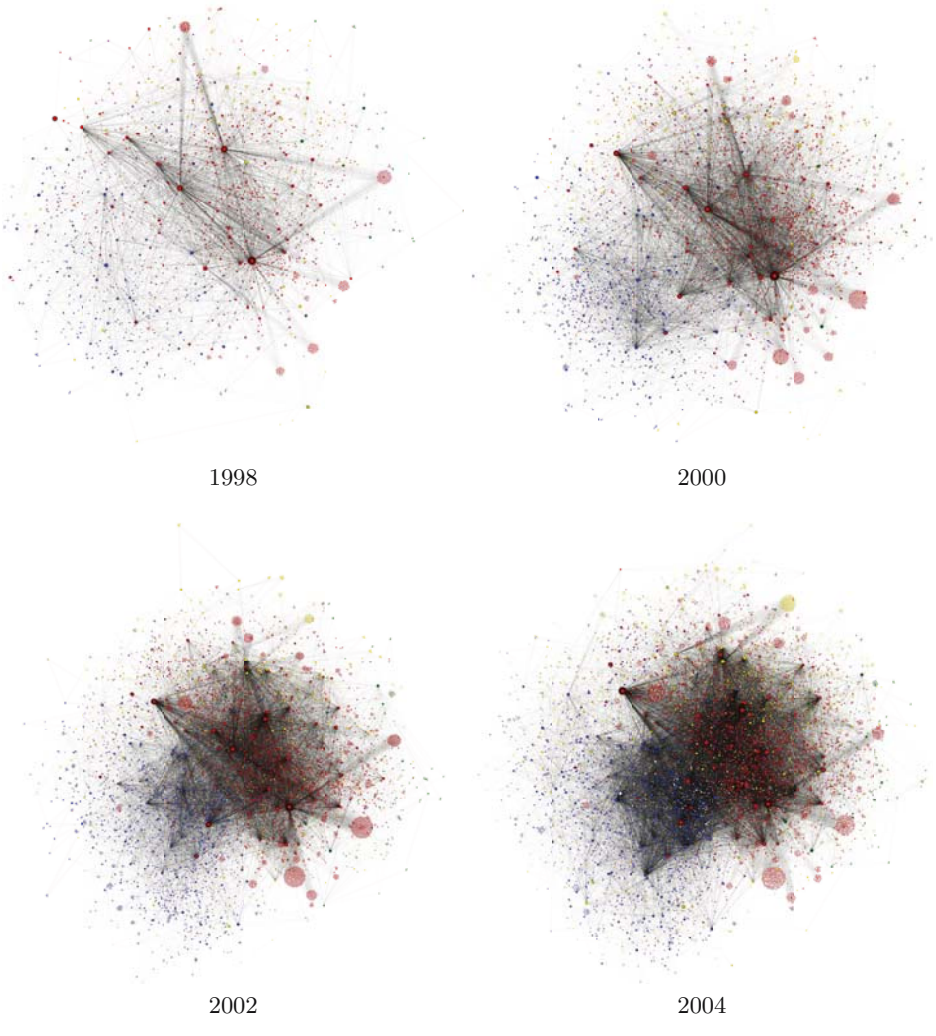


Fig. 7. Drawings of the evolving AS graph obtained from dynamic stress majorization in the online scenario

lowing the ideas in [4], we can do this with the stress majorization technique in a rather straightforward way by augmenting the stress with node displacement penalties, $\text{stress}(X) = \text{stress}_{\text{quality}}(X) + \text{stress}_{\text{stability}}(X)$, where $\text{stress}_{\text{quality}}(X)$ is defined as in (1) and $\text{stress}_{\text{stability}}(X) = \sum_{v \in U} w_{\text{st}} \|X_v - p_v\|^2$. The stability parameter w_{st} can be adjusted to trade the quality of the drawing for the stability. Figure 6 shows how the value of the quality stress function increases and the total movement of nodes decreases when the stability parameter increases.

Figures 7 and 8 show a selection of the resulting drawings when the fast two-phase stress majorization is applied in the dynamic online scenario for annual

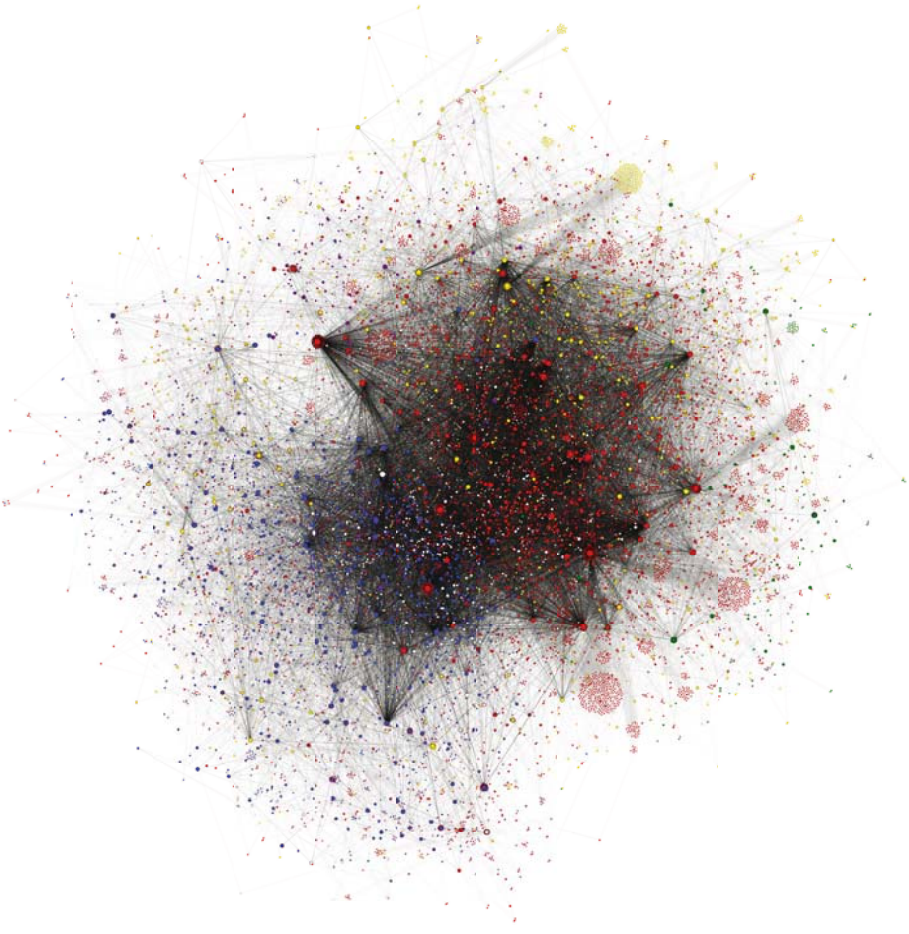


Fig. 8. Drawing of the 2006 AS graph finishing the sequence of Fig. 7

snapshots of the AS graph from 1997 to 2006.[1] A stability of $w_{\text{st}} = 20$ was used for creating these drawings.

5 Conclusion

We combined loss-less complexity reduction operations with tailored stress majorization techniques to produce drawings of a large evolving graph with skewed degree distribution, specifically the Internet on the level of autonomous systems. Even though the density of AS graphs increases rapidly over time, we believe that such a macroscopic view of the Internet can reveal evolution patterns, possibly supported by additional information coded in graphical attributes. It would

[1] The full animated sequence can be downloaded from `http://www.inf.uni-konstanz.de/algo/research/asgraph/`

be very interesting to see if our visualizations can actually help monitoring the evolving Internet.

References

1. Agrafiotis, D.K., Bandyopadhyay, D., Farnum, M.: Radial clustergrams: Visualizing the aggregate properties of hierarchical clusters. Journal of Chemical Information and Modeling 47(1), 69–75 (2007)
2. Alvarez-Hamelin, J.I., Dall'Asta, L., Barrat, A., Vespignani, A.: Large scale networks fingerprinting and visualization using the k-core decomposition. Advances in Neural Information Processing Systems 18, 41–50 (2006)
3. Baur, M., Brandes, U., Gaertler, M., Wagner, D.: Drawing the AS graph in 2.5 dimensions. In: Proc. Graph Drawing 2004, pp. 43–48 (2004)
4. Brandes, U., Wagner, D.: A Bayesian paradigm for dynamic graph layout. In: Proc. Graph Drawing, pp. 236–247 (1997)
5. Branke, J.: Dynamic graph drawing. In: Kaufmann, M., Wagner, D. (eds.) Drawing Graphs. LNCS, vol. 2025, pp. 228–246. Springer, Heidelberg (2001)
6. Brown, J., McGregor, A.: Network performance visualization: Insight through animation. In: Proc. Passive and Active Measurement Workshop, pp. 33–41 (2000)
7. Carmignani, A., Di Battista, G., Didimo, W., Matera, F., Pizzonia, M.: Visualization of the autonomous systems interconnections with hermes. In: Proc. Graph Drawing 2000, pp. 150–163 (2000)
8. Cheswick, B., Burch, H., Branigan, S.: Mapping and visualizing the internet. In: Proc. USENIX Annual Technical Conference, pp. 1–12 (2000)
9. Cooperative Association for Internet Data Analysis (CAIDA). Visualizing Internet topology at a macroscopic scale, `http://www.caida.org/analysis/topology/as_core_network/`
10. Diehl, S., Görg, C.: Graphs, they are changing – dynamic graph drawing for a sequence of graphs. In: Proc. Graph Drawing, pp. 23–30 (2002)
11. Erten, C., Harding, P.J., Kobourov, S.G., Wampler, K., Yee, G.: Graphael: Graph animations with evolving layouts. In: Proc. Graph Drawing, pp. 98–110 (2004)
12. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: Proc. ACM SIGCOMM, pp. 251–262 (1999)
13. Gaertler, M., Patrignani, M.: Dynamic analysis of the autonomous system graph. In: Proceedings Inter-Domain Performance and Simulation, pp. 13–24 (2004)
14. Gansner, E.R., Koren, Y., North, S.: Graph drawing by stress majorization. In: Proc. Graph Drawing, pp. 239–250 (2004), `http://www.research.att.com/~yehuda/pubs/majorization_full.pdf`
15. Gao, L.: On inferring autonomous system relationships in the internet. IEEE/ACM Transactions on Networking 9(6), 733–745 (2001)
16. Jin, C., Chen, Q., Jamin, S.: Inet: Internet topology generator. Technical Report CSE-TR-433-00, University of Michigan (2000)
17. Lad, M., Massey, D., Zhang, L.: Visualizing internet routing dynamics using linkrank. Technical report, UCLA (2005)
18. Mahadevan, P., Krioukov, D., Fomenkov, M., Huffaker, B., Dimitropoulos, X., Claffy, K.C., Vahdat, A.: The internet as-level topology: Three data sources and one definitive metric. ACM SIGCOMM Computer Communication Review 36(1), 17–26 (2006)

19. Neumann, P., Sheelagh, M., Carpendale, T., Agarawala, A.: Phyllotrees: Phyllotactic patterns for tree layout. In: Proc. EuroVis, pp. 59–66 (2006)
20. Stasko, J., Catrambone, R., Guzdial, M., McDonald, K.: An evaluation of space-filling information visualizations for depicting hierarchical structures. Int. J. Hum.-Comput. Stud. 53(5), 663–694 (2000)
21. University of Oregon. Route views project, `http://www.routeviews.org/`
22. Vogel, H.: A better way to construct the sunflower head. Mathematical Biosciences 44, 179–189 (1979)
23. Zhou, S., Mondragón, R.J.: Redundancy and robustness of AS-level internet topology and its models. Electronics Letters 40(2), 151–152 (2004)

Treemaps for Directed Acyclic Graphs*

Vassilis Tsiaras, Sofia Triantafilou, and Ioannis G. Tollis

Institute of Computer Science, Foundation for Research and Technology-Hellas,
Vassilika Vouton, P.O. Box 1385, Heraklion, GR-71110 Greece
{tsiaras,striant,tollis}@ics.forth.gr
http://www.ics.forth.gr
and
Department of Computer Science, University of Crete,
P.O. Box 2208, Heraklion, Crete, GR-71409 Greece
{tsiaras,striant,tollis}@csd.uoc.gr
http://www.csd.uoc.gr

Abstract. Gene Ontology information related to the biological role of genes is organized in a hierarchical manner that can be represented by a directed acyclic graph (DAG). Treemaps graphically represent hierarchical information via a two-dimensional rectangular map. They efficiently display large trees in limited screen space. Treemaps have been used to visualize the Gene Ontology by first transforming the DAG into a tree. However this transformation has several undesirable effects such as producing trees with a large number of nodes and scattering the rectangles associated with the duplicates of a node around the screen. In this paper we introduce the problem of visualizing a DAG as a treemap, we present two special cases, and we discuss complexity results.

Keywords: Treemap, Directed Acyclic Graph (DAG) Visualization, Gene Ontology.

1 Introduction

The Gene Ontology Consortium (GO) [14] databases store thousands of terms that describe information related to the biological role of genes. The information in GO is organized in a hierarchical manner where the terms are placed in layers that go from general to specific. The GO organization can be represented by a directed acyclic graph (DAG) where the set of vertices V is the set of terms and an edge is used to declare the *is_a* or *part_of* relationship between two terms.

Treemaps graphically represent hierarchical information via a two-dimensional rectangular map, providing compact visual representations of large trees through area, color and shading effects [3,5,11]. Treemaps, have also been used to visualize compound graphs that contain both hierarchical relations and adjacency relations [7].

* This work was supported in part by INFOBIOMED code: IST-2002-507585 and the Greek General Secretariat for Research and Technology under Program "ARISTEIA", Code 1308/B1/3.3.1/317/12.04.2002.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 377–388, 2007.

In the context of GO, treemaps have been used to visualize microarray data, where each gene transcript is assigned all possible paths that start from it and terminate to the root (the "all" term) of GO [1]. Symeonidis et al. in [12] proposed to decompose the complete GO DAG into a tree by duplicating the nodes with many in-coming edges, and then to use a treemap algorithm to visualize the tree, see Figure 1. The duplication of a node however triggers the duplication of all of its descendants. Therefore the transformation of a DAG into a tree leads to trees with (potentially exponentially) many more nodes than the original DAG. Symeonidis et al. in [12] reported that the initial GO DAG had $\sim$ 20.000 terms, while the produced equivalent tree had $\sim$ 100.000 terms. Another drawback of duplicating the nodes is that the rectangles associated with the multiple replicas of a node are scattered around the screen. In this paper we introduce the

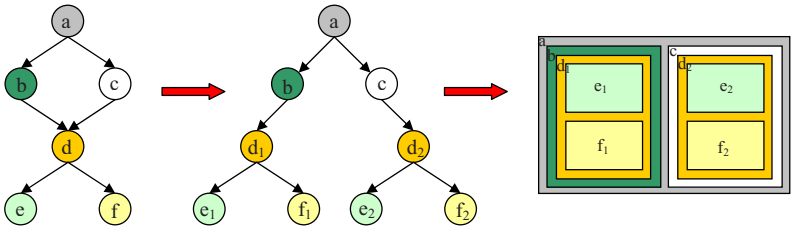


Fig. 1. Example of transforming a DAG into a tree and then drawing it as a treemap

problem of drawing a DAG as a treemap without converting it to a tree first. We consider several variations of the problem, we present some characterizations of simple families of DAGs that admit such a drawing, and provide complexity results for the general problem.

2 Problem Definition

2.1 Notations

Suppose that $G = (V, E)$ is a layered directed acyclic graph (DAG) with a partition of the node set V into subsets $L_1, L_2, \ldots, L_h$, such that if $(u, v) \in E$, where $u \in L_i$ and $v \in L_j$, then $i > j$. Without loss of generality we assume that the layering is proper, since the "long" edges that span more than two layers may be replaced by paths having dummy vertices in the internal layers [2].

Let R_v denote the display region of a node $v \in V$. Every directed edge $e = (u, v)$ from a node u to a node v corresponds to a drawing region $R_e = R_v \cap R_u$ which is the part of the child's drawing region R_v that is drawn within the parent's drawing region R_u.

Given a vertex $v \in V$ we denote the set of its in-coming and out-going edges by $\Gamma^-(v) = \{e \in E / destination(e) = v\}$ and $\Gamma^+(v) = \{e \in E / origin(e) = v\}$ respectively.

2.2 Treemap Drawing Constraints

Treemaps have the invariant that the drawing rectangle of any node (different from the root) is contained within the drawing rectangle of its parent. When the graph is a tree this invariant can easily be satisfied since every node has one parent. When the graph is a DAG, the above invariant should be replaced by the invariant that the drawing rectangle of any node is contained within the union of the rectangles of its parent nodes. Apart from this invariant it is plausible to assume that the drawing rectangles of sibling nodes do not overlap and that the drawing rectangle of a parent node is covered by the drawing rectangles of its children nodes. The above invariant and assumptions are summarized in the following definition.

Definition 1 (Treemap basic drawing constraints). *The drawing is constrained by the following rules.*

B1. The display area of the DAG (screen) is a rectangle.
B2. Every node is drawn as a rectangle (R_v is a rectangle for every $v \in V$).
B3. If two distinct nodes $u, v \in V$ are assigned to the same layer their rectangles do not overlap ($area(R_u \cap R_v) = 0$).
B4. The rectangle of a child node occupies a non-zero area in each one of its parent node rectangles. ($area(R_e) = area(R_u \cap R_v) \neq 0$ if $e = (u, v) \in E$).
B5. The rectangle of a child node is contained in the union of rectangles of its parent nodes ($R_v \subset \cup_{(u,v)\in E} R_u$).
B6. The rectangle of a parent node is covered by the rectangles of its children nodes ($R_u \subset \cup_{(u,v)\in E} R_v$).

The drawing rules of the above definition are quite general since they do not constrain the area of the leaf nodes, and the proportion of a child's node area that is drawn on each one of its parent rectangles. To simplify the analysis of the problem we constraint these two parameters by making the following assumptions.

Definition 2 (Treemap additional drawing constraints)

A1. The leaf nodes are drawn in equal area ($\frac{screen\ area}{number\ of\ leaf\ nodes}$) rectangles.
A2. The drawing rectangle of a child node occupies the same area on each one of its parent rectangles (For every non source node v, $area(R_e) = \frac{area(R_v)}{|\Gamma^-(v)|}$, for every $e \in \Gamma^-(v)$).

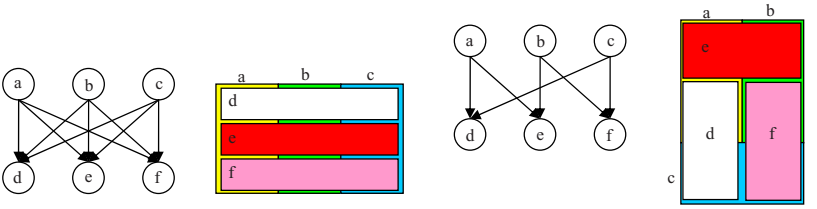
In the following we will use the term treemap drawing to characterize a drawing according to the basic and additional drawing constraints.

Having defined the drawing rules, we can define the following problems:

1. Given a DAG G_1, does G_1 admit a treemap drawing?
2. In case that the answer to the first problem is negative, what is the minimum number of node duplications that are needed to transform G_1 into a DAG G_2 that admits a treemap drawing?

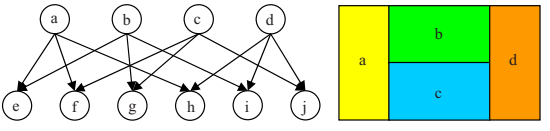
2.3 Examples and Counter-Examples of DAGs That Admit a Treemap Drawing

Examples of DAGs that admit a treemap drawing appear in Figure 2. From the

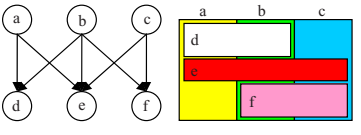


(a) The K_{33} DAG (b) A "circular" DAG

Fig. 2. Examples of DAGs that can be drawn as treemaps



(a) Consecutive conditions violation



(b) Area imbalance

Fig. 3. Examples of DAGs that do not admit a treemap drawing

counter-examples of Figure 3 we see that there are DAGs that cannot be drawn as treemaps. The DAG in Figure 3(a) cannot be drawn due to adjacency constraint violation. The leaf nodes e, f, g, h, i, j constrain the parent nodes a, b, c, d to be drawn in adjacent rectangles. However we cannot have a configuration where all the pairs $\{a, b\}$,$\{a, c\}$,$\{a, d\}$,$\{b, c\}$,$\{b, d\}$,$\{c, d\}$ of rectangles are adjacent. In this case in order to draw the DAG we can either duplicate one of the nodes e, f, g, h, i, j or draw one of these nodes using two disjoint rectangles. The two operations are similar and when applied to child node remove the corresponding adjacency constraint.

In general, due to the four color (map coloring) theorem there exists a counter-example involving five parent nodes and ten children nodes (one child node for every pair of parent nodes), even in the case that we relax constraint B2, allowing drawings to be simply connected regions of the plane.

The example of Figure 3(b) shows a DAG that does not admit a treemap drawing, due to area imbalance among the first layer nodes a, b, c. Assuming

that the leaf nodes have unit area, then nodes a and c have area $1/2 + 1/3$ while node b has area $1 + 1/3$. However, if we relax constraint A2, then this DAG admits a treemap drawing.

2.4 Node Duplication

Usually, a DAG encountered in practice does not admit a treemap drawing. In this case we should relax one or more of constraints B1-B6, A1-A2 or change the form of the DAG. Symeonidis et al. in [12] chose to transform the DAG into a forest of trees by multiple node duplications. An example of a node duplication is shown in Figure 3(b), where after the creation of two replicas of node e, one with two parents and one with one parent, the DAG is transformed into a new DAG which admits a drawing, see Figure 4.

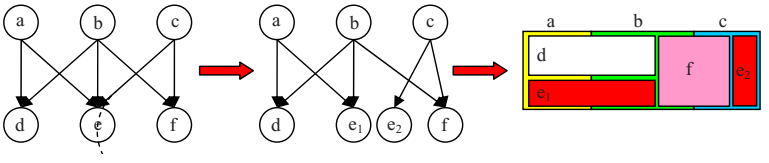


Fig. 4. After the duplication of node e the DAG of Figure 3 is transformed into a DAG that has a treemap drawing

3 Special Cases

We will continue by considering two special cases. The first case is based on a restricted form of DAGs, the second on a restricted form of treemaps.

3.1 Two Terminal Series Parallel Digraphs

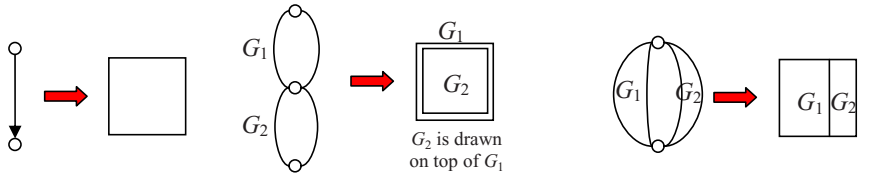
A Two Terminal Series Parallel (TTSP) digraph is recursively defined as follows [2,13]. An edge joining two vertices is a TTSP digraph. Let G_1 and G_2 be two TTSP digraphs. Their series and parallel compositions, defined below, are also TTSP digraphs.

- The series composition of G_1 and G_2 is the digraph obtained by identifying the sink of G_1 with the source of G_2.
- The parallel composition of G_1 and G_2 is the digraph obtained by identifying the source of G_1 with the source of G_2 and the sink of G_1 with the sink of G_2.

Due to its recursive structure a TTSP digraph always admits a treemap drawing. The base TTSP digraph is drawn as a rectangle. In a series composition the rectangle of graph G_2 is drawn on the top of the rectangle of graph G_1. In a parallel composition the rectangle of the composite graph is sliced into the rectangles of G_1 and G_2.

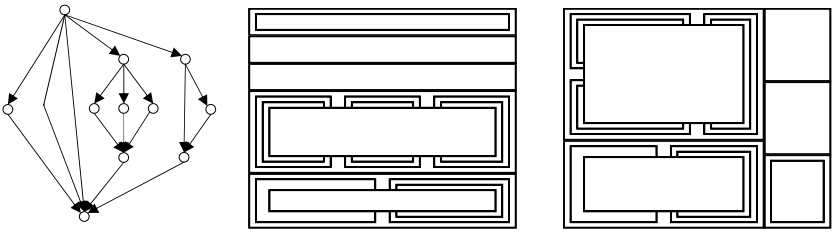
Algorithm

1. Construct the decomposition tree of G [13] and merge the adjacent P-nodes. In the resulting tree the P-nodes may have two or more children.
2. Using the decomposition tree calculate the size of the components.
   (a) In a series composition $size(G) = size(G_1) = size(G_2)$.
   (b) In a parallel composition $size(G) = size(G_1) + size(G_2) + \ldots + size(G_k)$.
3. Using the decomposition tree recursively draw the component rectangles.
   (a) The rectangles have area proportional to the size of the corresponding component.
   (b) In a series composition, the rectangles of the two components coincide.
   (c) In a parallel composition, use any of the existing treemap algorithms to lay out the component rectangles.



(a) The base TTSP digraph (b) Series composition (c) Parallel composition

Fig. 5. Recursive definition of a TTSP digraph and the corresponding recursive treemap drawings



(a) A TTSP (b) Slice and Dice layout (c) Squarified layout

Fig. 6. Example of a TTSP digraph treemap drawing

3.2 One Dimensional Treemaps

Definition 3. *A treemap is called one dimensional if the rectangle representing a node is divided with vertical (or horizontal) lines into smaller rectangles representing its children and the orientation of the lines is the same for all the nodes of a hierarchy.*

Since the height (resp. width) of all the rectangles is constant and equal to the height (resp. width) of the screen, the problem is one dimensional and the rectangles R_q can be represented by intervals I_q. Also only the ordering and not length of the intervals $I_e = I_u \cap I_v$, $e = (u, v) \in E$ constrain the problem. For this reason we consider the out-going edges of every non-leaf node $u \in V$ as subnodes inside the node. The in-coming edges of a node can be considered as in-coming edges of every one of its subnodes. A drawing $\cup_{n_e \in L_k} I_{n_e}$ of the subnodes $n_e \in L_k$ of layer $k \in \{2, \ldots, h\}$ corresponds to an ordering of the subnodes $n_e \in L_k$.

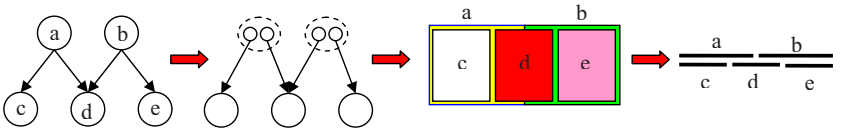


Fig. 7. A one dimensional treemap example. A subnode is created for each out-going edge.

Definition 4 (ONE DIMENSIONAL TREEMAP FOR DAG). *The recognition problem*
INSTANCE: *A DAG G.*
QUESTION: *Can G be drawn as a one dimensional treemap?*

Suppose that u is a node in layer L_k. We will give the necessary and sufficient conditions that the ancestor subnodes of u must satisfy in order to be able to draw I_u as an interval. With the term ancestor subnodes we mean the subnodes reachable from node u if the direction of the edges is reversed.

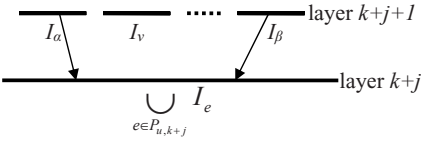
Let $P_{u,i}$ denote the set of ancestor subnodes of node $u \in L_k$ in layer $i \in \{k+1, \ldots, h\}$.

Theorem 1 (Necessary conditions). *Suppose that in a one dimensional treemap drawing of a graph $G = (V, E)$ a node $u \in L_k$ can be drawn as an interval I_u. Then the union of the drawings of the ancestor subnodes of u in layer i, $\cup_{e \in P_{u,i}} I_e$, is an interval for every layer $i \in \{k+1, \ldots, h\}$.*

Proof. By induction on the layers L_{k+j}, $j = 0, \ldots, h-k$.

For $j = 0$, I_u is an interval by the hypothesis.

Now, suppose that for $j = 0, \ldots, i < h-k$, there is an ordering of the subnodes in every one of the layers $k+j, \ldots h$, such that $\cup_{e \in P_{u,k+j}} I_e$ to be an interval, but $\cup_{e \in P_{u,k+j+1}} I_e$ cannot be an interval. Then there is at least one node $v \in L_{k+j+1}$, $v \notin P_{u,k+j+1}$, which is between two nodes $\alpha, \beta \in P_{u,k+j+1}$. Then the interval $\cup_{e \in P_{u,k+j}} I_e$ intersects the intervals I_α and I_β but not the interval I_v, a contradiction.



Theorem 2 (Sufficient condition). *If there is an ordering of the subnodes in L_{k+1} such that the parent subnodes $P_{u,k+1}$, of a node $u \in L_k$, $k < h$ are consecutive in this ordering then u can be drawn as an interval.*

Proof. Since $P_{u,k+1}$ are consecutive $\cup_{e \in P_{u,k+1}} I_e$ is an interval. Then simply draw I_u in this interval.

Algorithm
From the above theorems every non-source node $u \in L_k$ defines constraints on the admissible subnode permutations in each one of the layers L_i, $i \in \{k, \ldots, h\}$. Therefore the decision problem is transformed to $h-1$ consecutive ones decision problems [4,9]. One consecutive ones problem for each layer L_i, $i \in \{2, \ldots, h\}$.

There is one list $list_i$ of constraints for each layer L_i, $i \in \{2, \ldots, h\}$. Initially the lists are empty. Then we add the constraints as follows.

for $i = 2$ to h do
  for $v \in L_i$ do
    add to the $list_i$ the constraint that the subnodes of v are consecutive.

for $i = 1$ to $h - 1$ do
  for $v \in L_i$ do
    for $j = i + 1$ to h do
      add to the $list_j$ the constraint that the subnodes $P_{v,j}$ are consecutive.

Complexity analysis
Without loss of generality we assume that there are no leaf nodes in layers $2, \ldots, h$.

Suppose that $n_i = |L_i|$, $i \in \{1, \ldots, h\}$ and that m_i edges go from layer i to layer $i-1$, $i \in \{2, \ldots, h\}$.

For $i \in \{2, \ldots, h\}$ the list $list_i$ has at most n_i trivial constraints and at most n_j constraints due to nodes at layer L_j, $j < i$. In total it has $n_i + \ldots + n_1$ constraints. Each costraint has size at most m_i.

Therefore the total time is:

$$\sum_{i=2}^{h} O(m_i \cdot (n_i + \ldots + n_1)) = \sum_{i=2}^{h} O(m_i \cdot n) = O(m \cdot n)$$

which is polynomial on the input size $m = \sum_{i=1}^{L} m_i$ and $n = \sum_{i=1}^{L} n_i$

4 The General Case

4.1 The Recognition Problem

Taking the nodes of a layer L_k isolated from the rest of the DAG, the problem is similar to a floorplan problem where the display area is dissected into $n_k = |L_k|$

soft rectangles, i.e., rectangles whose area is fixed but their dimensions may vary. The number of possible dissections (the solutions space) is bounded below by $\Omega(n_k!2^{3n}/n_k^4)$ and above by $O(n_k!2^{5n}/n_k^{4.5})$ [10].

Considering two consecutive layers L_{k+1} and L_k of a DAG, the layouts of the two layers are constrained by the edges among the two layers, according to the drawing rules. The combined solution space may be empty or contain a number of solutions. We will show that deciding whether the solution space is empty or not is NP-complete.

Definition 5 (TREEMAP FOR DAG). *The recognition problem*
INSTANCE: *A DAG G.*
QUESTION: *Can G be drawn as a treemap?*

Theorem 3. *The TREEMAP FOR DAG decision problem is NP-complete even if we restrict it to two layer weakly connected DAGs.*

Proof. Given a dissection of the display area (screen) into $|L_k|$ rectangles for each layer L_k, $k \in \{1, \ldots, h\}$ of a DAG G, we can check in polynomial time if these dissections correspond to a treemap drawing of G. Therefore the problem belongs to NP.

Next we will show that the problem TREEMAP FOR DAG is NP-hard. The proof will be done by reducing the 3-PARTITION problem to a restricted version of the TREEMAP FOR DAG problem. Namely, as input we consider only two layer DAGs. For simplicity of the proof we will allow an input DAG to be composed of several weakly connected components.

Definition 6 (3-PARTITION).
INSTANCE: *A multiset A of $3m$ positive integers $A = \{\alpha_1, \alpha_2, \ldots, \alpha_{3m}\}$ where the α_i's are bounded above by a polynomial in m and $\frac{\Sigma}{4} < \alpha_i < \frac{\Sigma}{2}$, where $\Sigma = \frac{1}{m}(\alpha_1 + \alpha_2 + \ldots + \alpha_{3m})$.*
QUESTION: *Can A be partitioned into m triples $A_1, A_2, ..., A_m$ such that each triple has the same sum. Specifically each triple must sum to Σ.*

The condition $\frac{\Sigma}{4} < \alpha_i < \frac{\Sigma}{2}$ forces every set of α_i's summing to Σ, to have size exactly 3. The 3-PARTITION is strongly NP-complete since it remains NP-complete even when representing the numbers in the input instance in unary [6]. The reduction is done by local replacement and using an enforcer.

Enforcer: The DAG used as enforcer has $2m+2$ nodes at the second layer. The nodes β and $1, 2, \ldots, 2m+1$. At the first layer there are $(m+1)\Sigma + 6m + 4$ nodes. Each of the nodes $1', 2', \ldots, (2m+1)'$ has two parents. One is node β and the other is the corresponding numbered node in the first layer. The β-node rectangle is drawn in one side of the enforcer and precludes any other rectangle to be drawn along this side. Also the β-node together with the γ-node force the nodes $1, 2, \ldots, 2m+1$ to be drawn as consecutive rectangles. For every pair of $(j, j+1)$ of second layer nodes there exists a first layer node which has them as parents and constrains them to be consecutive. The second node which has as parents the nodes j and $j+1$ is used for completing the drawing (garbage

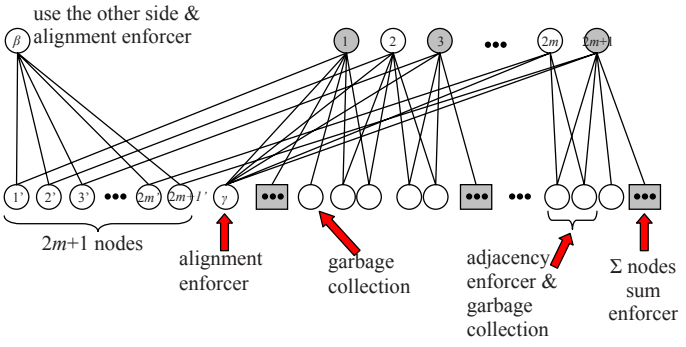


Fig. 8. The enforcer used in the proof

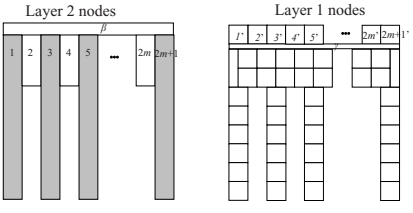
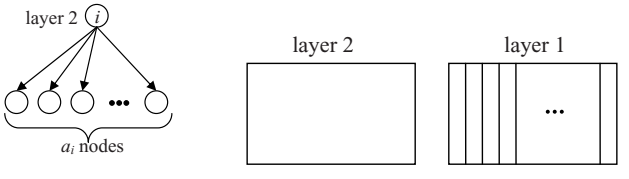


Fig. 9. One possible drawing of the enforcer. The rectangles $1, 2, \ldots, 2m+1$ are forced by the rectangle γ to have the same width.

collection). Also for garbage collection one node is connected to node 1 and one node to node $2m+1$.

Finally, every odd numbered node of the second layer has Σ children nodes.

Local replacement: For each $\alpha_i \in A$ we consider a two layer DAG which has one node at layer two (parent node) and α_i nodes at layer one (children nodes).



The drawing of node i is a rectangle with area α_i, but without any constraint on the aspect ratio of its sides. In order for the final drawing to be a rectangle the $3m$ rectangles should fill the holes of the enforcer.

The reduction from the 3-PARTITION to TREEMAP FOR DAG uses polynomial number of resources since the numbers involved in 3-PARTITION are bounded by a polynomial in m.

The 3-PARTITION instance has a solution if and only the TREEMAP FOR DAG instance has a solution since in every hole can fit exactly three rectangles.

4.2 Minimization of Node Duplication

Definition 7 (MINIMUM DUPLICATION OF NODES).
INSTANCE: *A DAG G_1 and an integer K.*
QUESTION: *Can G_1 be transformed into a DAG G_2 that admits a treemap drawing by duplicating at most K nodes.*

Comment: The problem MINIMUM DUPLICATION OF NODES is NP-complete since its restriction for $K = 0$ is the problem TREEMAP FOR DAG, which is NP-complete.

5 Discussion

In this paper we introduced the problem of drawing a DAG as a treemap. We defined the recognition and minimization problems and we showed that in the general case they are NP-complete and NP-hard respectively. We also considered two special cases by restricting the form of the DAG and of the treemap respectively. We are currently investigating drawing heuristics based on relaxations of the drawing constraints and\or restrictions on the form of DAGs. The results of this research will be published in a subsequent paper concerning the application of these techniques to hierarchically organized ontologies.

References

1. Baehrecke, E., Dang, N., Babaria, K., Shneiderman, B.: Visualization and analysis of microarray and gene ontology data with treemaps. BMC Bioinformatics 5(84) (2004)
2. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of graphs. Prentice - Hall, New Jersey, U.S.A. (1998)
3. Bederson, B., Shneiderman, B., Wattenberg, M.: Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. ACM Transactions on Graphics 21(4), 833 (2002)
4. Booth, S., Lueker, S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. Journal of Computer and System Sciences 13, 335 (1976)
5. Bruls, M., Huizing, K., van Wijk, J.J.: Squarified treemaps. In: Proceedings of Joint Eurographics and IEEE TCVG Symposium on Visualization, p. 33. Springer, Heidelberg (2000)
6. Garey, M., Johnson, D.: Complexity results for multiprocessor scheduling under resource constraints. SIAM Journal on Computing 4(4), 397 (1975)
7. Holten, D.: Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. IEEE Transactions on Visualization and Computer Graphics 12(5), 741 (2006)
8. Hsu, W.-L.: PC-Trees vs. PQ-Trees. In: Wang, J. (ed.) COCOON 2001. LNCS, vol. 2108, p. 207. Springer, Heidelberg (2001)
9. Meidanis, J., Porto, O., Telles, G.: On the consecutive ones property. Discrete Applied Mathematics 88, 325 (1998)

10. Shen, Z.C., Chu, C.: Bounds on the Number of Slicing, Mosaic, and General Floorplans. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 22(10), 1354 (2003)
11. Shneiderman, B.: Tree visualization with tree-maps: 2-d space-filling approach. ACM Transactions on Graphics 11(1), 92 (1992)
12. Symeonidis, A., Tollis, I., Reczko, M.: Visualization of Functional Aspects of microRNA Regulatory Networks Using the Gene Ontology. In: Maglaveras, N., Chouvarda, I., Koutkias, V., Brause, R. (eds.) ISBMDA 2006. LNCS (LNBI), vol. 4345, pp. 13–24. Springer, Heidelberg (2006)
13. Valdes, J., Tarjan, R., Lawler, E.L.: The recognition of Series Parallel digraphs. SIAM Journal on Computing 11, 289–313 (1982)
14. http://www.geneontology.org

Drawing Graphs with GLEE*

Lev Nachmanson, George Robertson, and Bongshin Lee

Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
{levnach,ggr,bongshin}@microsoft.com

Abstract. This paper describes novel methods we developed to lay out graphs using Sugiyama's scheme [16] in a tool named GLEE. The main contributions are: a heuristic for creating a graph layout with a given aspect ratio, an efficient method of edge-crossings counting while performing adjacent vertex swaps, and a simple and fast spline routing algorithm.

1 Introduction

GLEE is a graph drawing tool that is being developed at Microsoft Research. Eiglsperger et al. [8] mention that most practical implementations of directed graph layout engines follow Sugiyama's scheme (or STT); GLEE is not an exception. In spite of the fact that there is lot of research devoted to the scheme, we were confronted with questions during the development process, for which we did not find answers in the literature. In the paper we address some of these questions. To our knowledge, nobody has solved the problem of creating a layered graph layout with a given aspect ratio. We developed a heuristic for creating such a layout. At an earlier stage of the development, GLEE's performance bottleneck was counting edge crossings while swapping adjacent nodes during the ordering step. We designed data structures and procedures to speed up the counting. Furthermore, several previous approaches for drawing splines did not give us satisfactory results. We developed a simple and efficient algorithm, producing aesthetic splines.

2 Layout with a Given Aspect Ratio

When laying out a graph, we would like to better utilize the available space and create an aesthetically pleasing layout. Here we present a heuristic of laying out graphs inside of a rectangle of a given aspect ratio. Figures 1 and 2 show two drawings of the same graph in rectangles of the same size. Both layouts were created by GLEE. We used the default algorithm for Fig. 2, and the heuristic for Fig. 1. The drawing in Fig. 1 better uses the available space and its larger nodes improve the readability.

* The full version of the paper is available at ftp://ftp.research.microsoft.com/pub/tr/TR-2007-72.pdf. GLEE can be downloaded at http://research.microsoft.com/~levnach/GLEEWebPage.htm.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 389–394, 2007.

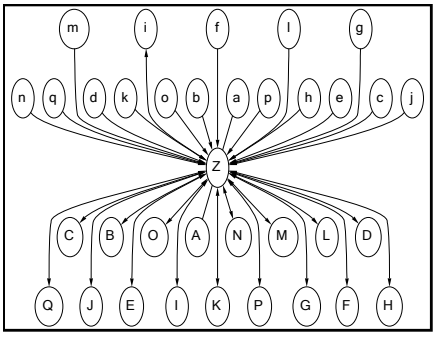


Fig. 1. Using the heuristic

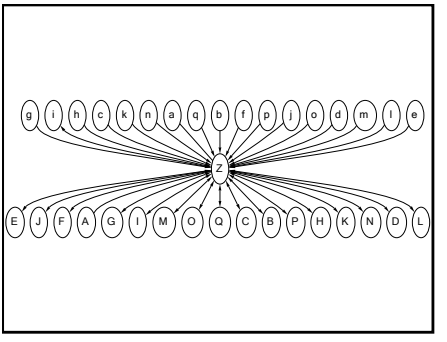


Fig. 2. Using the default layout

2.1 Description of the Heuristic

An algorithm for scheduling computer tasks for parallel processing on multiple processors can be used to compute the layering for STT. A scheduling algorithm deals with a DAG of computer tasks. If there exists a directed path in the DAG from task t_0 to task t_1, then t_0 is called a predecessor of t_1 and t_0 has to be completed before t_1 can be begun. Suppose each task takes a unit of time to complete on any processor. Then a task schedule corresponds to the layering, such that tasks scheduled to time 0 form the top layer, tasks scheduled to time 1 form a layer right below it, and so on. An upper limit on the sum of the node widths in a layer can play the role of the number of processors in a task scheduling algorithm. In particular, we could use Coffman-Graham algorithm [6] for our purposes. For a real number w let's call $A(w)$ a variation of Coffman-Graham algorithm where we require that the sum of node widths in a layer is not greater than w. In addition, $A(w)$ respects *Separation*. There is a tendency that for a larger w algorithm $A(w)$ produces a layout with the less height. We use it to find out the width w giving a good result. Let us call $B(w)$ the following procedure applied to DAG G:

Execute the layering step using $A(w)$
Execute the ordering step
Calculate node x-coordinates of the proper layered graph

We apply a binary search to find W such that $B(W)$ produces the aspect ratio which is close to the given one. Algorithm $B(w)$ can be repeated many times during the binary search. To speed up its execution, starting from graphs of a specific size, we apply the algorithm of [5] for calculation of node x-coordinates. We experimented with a variation of $A(w)$ where the width of edges crossing the layer is taken into account; surprisingly, the results were better when we ignored edge widths.

The application of $B(w)$ alone does not produce good layouts. To achieve a better quality of the layout we apply additional heuristics. These heuristics are node demotion and balancing of virtual and original nodes during the process of swapping of adjacent nodes.

Node demotion. When a processor is available and there is a task which is ready to be executed, Coffman-Graham algorithm immediately assigns the task to the processor. As

a result, some nodes can be positioned too high. We can improve the layout by pulling nodes down, or, in other words, by demoting them. The demotion step that we execute is the promotion step [14] being run in the opposite direction.

Balancing of virtual and original nodes. The heuristic helps in spreading uniformly edges passing a layer and nodes of the layer. We apply the heuristic during the ordering step. The ordering step starts when we already have a proper layered graph, but the order of nodes within a single layer is not yet defined. During the step we traverse the layers up and down several times applying the median method of [9] and create an ordering within the layers. The following sub-step of the ordering step is the swapping of nodes which are adjacent on the same layer. This is done to reduce the number of edge crossings. Here we utilize the sub-step for yet another purpose of spreading evenly virtual and original nodes. Let us describe the way we change the process of swapping of adjacent nodes. For a fixed layer, if the layer has fewer virtual nodes than original ones then we call virtual nodes separators and original nodes nulls. Otherwise we call virtual nodes of the layer nulls and original nodes separators. In the usual swapping process we proceed with a swap if it reduces the number of edge crossings, and do not proceed when it increases the number of edge crossings. In the case when the number of edge crossings does not change as a result of the swap, we have a freedom to apply the heuristic. Consider swapping of separator s with null m. Let K (M) be the set of all null nodes z to the left (right) of s such that no separator is positioned between z and s. Let K' and M' be sets defined the same way but as if s and m are swapped. If $||K'| - |M'|| < ||K| - |M||$ then we proceed with the swap.

Related work. Authors of Graphviz, a popular tool based on STT, mention Coffman-Graham algorithm as one of the approaches [2] to the aspect ratio problem. In [11] and [15] methods are developed to calculate *layering* for a directed graph with constraints on the width and the height of the layering. In the context of [11] and [15] the width of a layering is the maximum number of nodes in its layers, and the height of a layering is the number of its layers. Heuristics of [15] can be used instead of $A(w)$ in our approach, but we have not tried that.

3 Efficient Counting of Edge Crossings During Adjacent Swaps

Counting the crossings of edges connecting two neighboring layers at the ordering step is done by using the technique from [4] and works fast. However we observed a performance bottleneck in counting edge crossings at the phase of swapping adjacent nodes in a layer. The approach and data structures suggested here lead to an efficient implementation.

Proposition 1. *Swap of adjacent layer nodes u and v can be produced with the amortized cost $O(d(u) + d(v))$, where d is the degree of layered graph nodes.*

We give the details in the full version of the paper [1].

4 Spline Routing

In general, in our method we modify the given polyline to avoid nodes, straighten the polyline, and fit Bezier segments into its corners. Before describing the approach we need to define some notions. Let *PG* be the proper layered graph with already defined positions of nodes. For an edge $(u, v) \in E$ there is a unique sequence of nodes $U(e) = [u_0, ..., u_n]$ connecting u and v, such that; $u_0 = u$,$u_n = v$, (u_i, u_{i+1}) is an edge of *PG* for $i = 0, ..., n-1$, and nodes $u_1, ..., u_{n-1}$ are virtual. We call a polyline formed by positions of nodes from $U(e)$ the polyline of edge e. Let us define blocking nodes of an edge. Nodes u and v of *PG* belonging to the same layer are called non-blocking to each other if they are both virtual and some of edges adjacent to u cross some edges adjacent to v, as shown in Fig. 3. Otherwise u and v are called blocking to each other. The intuition is that if u is blocking for v, then a spline passing through v has to be disjoint from u. A node is called blocking for an edge e if it is blocking for some node of $U(e)$. For example, if u, v belong to the same layer and u is an original node, then u,v are blocking to each other. We build a spline for edge e of G and let p be a polyline of e. We proceed by the following steps which are explained below:

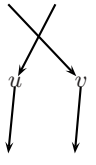


Fig. 3. Non-blocking u, v

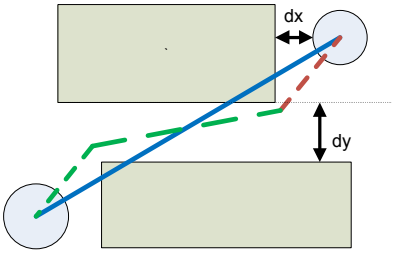


Fig. 4. Polyline refinement step

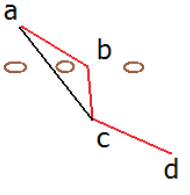


Fig. 5. Forbidden shortcut

1) Refine p if it crosses nodes blocking for e
2) Straighten p by using an inflection heuristic.
3) Smoothen p by fitting Bezier segments into p corners.

In the refinement step, using the node and layer separation, we replace each segment of p intersecting the blocking nodes with a polyine disjoint from the nodes. The inserted polyline can be split up into two connected pieces; one piece turns clockwise and the other counterclockwise as illustrated at Fig. 4. In the inflection heuristic we remove some polyline vertices by shortcutting them. Suppose that a, b, c and d are consecutive nodes of p. If at b polyline p turns, for example, clockwise, and at c it turns counterclockwise, then we remove b from p, but only in the case when triangle a,b,c does not intersect a blocking node for e. Fig.5 shows a situation where we do not shortcut vertex b. Allowing such shortcuts can create additional edge spline crossings or allow the spline to intersect a blocking node.

Before smoothing corners we simplify the polyline, without actually changing its geometry, in a way that no three consecutive vertices of the polyline are collinear. Let

a, b and c be consecutive vertices of the polyline, and k be a real number. Let us set m to $a + k(b - a)$ and n to $c + k(b - c)$. We call $Bz(k)$ the cubic Bezier segment with control points m, $(m + 2b)/3$, $(n + 2b)/3$, and n. We find minimal k of the form $1 - 1/2^i$, for $i = 1, 2, ...$, such that the figure bounded by line segment $[m, b]$, line segment $[b, n]$ and $Bz(k)$ does not intersect blocking nodes for e. Such k exists since p does not intersect the blocking nodes. When we build $Bz(k)$ for each polyline corner, we reach our goal.

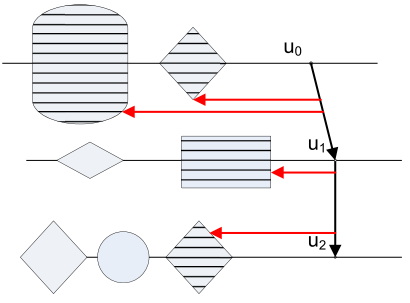


Fig. 6. Set *LS* is composed by dashed nodes

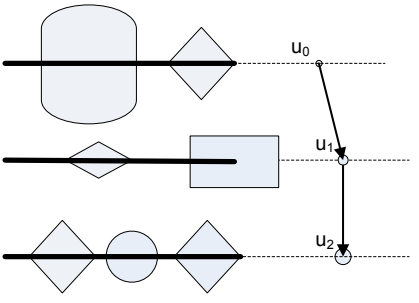


Fig. 7. Set *LT* is formed by thick horizontal line segments

We use the structure of *PG* to efficiently check for intersections. Namely, we do not intersect the polyline or the Bezier segment with all blocking nodes of edge e, but rather only with a subset of these nodes or with a specially constructed set of line segments. Let us show how to build these sets from the left of the edge. Denote by L all nodes of *PG* blocking for e and positioned to the left of e. The selected subset of nodes called *LS* is defined as the set of all nodes of L that are reachable by a horizontal ray starting at a point on p, as illustrated at Fig. 6. Set *LT* is formed by horizontal line segments starting at the left side of the *PG* bounding box and ending at the rightmost blocking node for u_i positioned the left of u_i, for $i = 0, ..., n$, as shown in Fig. 7. Sets *RS* and *RT* are defined by the symmetry. Step 1) checks intersections of the polyline only with nodes from sets *LS* and *RS*, while steps 2) and 3), in addition, intersect segment $[a, c]$ or $Bz(k)$ with sets *LT* and *RT* to detect intersections of triangle a, b, c or the figure bounded by $[m, b], [b, n], Bz(k)$ correspondingly with the blocking nodes. To speed up the calculation we build spatial trees on sets *LS*, *RS*, *LT* and *RT*, and utilize them in the crossing routine.

5 Conclusion and Future Work

We presented several novel methods producing good layouts for directed graphs using Sugiyama scheme. We developed a heuristic to lay out graphs inside of a rectangle of a given aspect ratio, which helps us better utilize the available space and create an aesthetically pleasing layout. We presented a fast edge-crossings counting method for adjacent node swaps. We described an efficient edge routing algorithm, which modifies a given edge polyline to avoid nodes, straightens the polyline, and fits Bezier segments into its corners.

We plan to introduce more balance and symmetry into GLEE layouts. Important features to add to the tool include interactive and incremental layout, and graph editing.

We would like to thank Stephen North and Yehuda Koren for fruitful discussions.

References

1. Drawing graphs with GLEE technical report, Lev Nachmanson, George Robertson and Bongshin Lee, ftp://ftp.research.microsoft.com/pub/tr/TR-2007-72.pdf
2. Graphviz todo list (December 22, 2005), http://www.graphviz.org/doc/todo.html
3. Abello, J., Gansner, E.R.: Short and smooth polygonal paths. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 151–162. Springer, Heidelberg (1998)
4. Barth, W., Jünger, M., Mutzel, P.: Simple and efficient bilayer cross counting. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 130–141. Springer, Heidelberg (2002)
5. Brandes, U., Köpf, B.: Fast and simple horizontal coordinate assignment. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 31–44. Springer, Heidelberg (2002)
6. Coffman, E.G., Graham, R.L.: Optimal Scheduling for Two-Processor Systems. Acta Informatica 1(3), 200–213 (1972)
7. Dobkin, D.P., Gansner, E.R., Koutsofios, E., North, S.: Implementing a general-purpose edge router. In: Di Battista, G. (ed.) Graph Drawing, Rome, Italy, September 18-20, 1997, pp. 262–271. Springer, Heidelberg (1998)
8. Eiglsperger, M., Siebenhaller, M., Kaufmann, M.: An efficient implementation of sugiyama's algorithm for layered graph drawing. In: Pach, J. (ed.) Graph Drawing, New York, pp. 155–166. Springer, Heidelberg (2004)
9. Gansner, E.R., Koutsofios, E., North, S.C., Vo, K.-P.: A Technique for Drawing Directed Graphs. IEEE Transactions on Software Engineering 19(3), 214–230 (1993)
10. Goodrich, M.T., Kobourov, S.G. (eds.): GD 2002. LNCS, vol. 2528. Springer, Heidelberg (2002)
11. Healy, P., Nikolov, N.S.: A branch-and-cut approach to the directed acyclic graph layering problem. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 98–109. Springer, Heidelberg (2002)
12. Lutterkort, D., Peters, J.: Smooth paths in a polygonal channel. In: Symposium on Computational Geometry, pp. 316–321 (1999)
13. Myles, A., Peters, J.: Threading splines through 3d channels. Computer-Aided Design 37(2), 139–148 (2005)
14. Nikolov, N.S., Tarassov, A.: Graph layering by promotion of nodes. Discrete Applied Mathematics 154(5), 848–860 (2006)
15. Nikolov, N.S., Tarassov, A., Branke, J.: In search for efficient heuristics for minimum-width graph layering with consideration of dummy nodes. J. Exp. Algorithmics 10(2.7) (2005)
16. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man and Cybernetics SMC-11(2), 109–125 (1981)

Graph Drawing Contest Report

Christian A. Duncan[1], Stephen G. Kobourov[2], and Georg Sander[3]

[1] Louisiana Tech University, Ruston, LA 71272, USA
duncan@latech.edu
[2] University of Arizona, Tucson, AZ 85721, USA
kobourov@cs.arizona.edu
[3] ILOG, 94253 Gentilly Cedex, France
sander@ilog.fr

Abstract. This report describes the 14th Annual Graph Drawing Contest, held in conjunction with the 2007 Graph Drawing Symposium in Sydney, Australia. The purpose of the contest is to monitor and challenge the current state of graph-drawing technology.

1 Introduction

This year's Graph Drawing Contest had three distinct categories: the Graph Drawing Challenge, the Free-Style category, and a Social Network category. Repeating the focus of the previous year, the Graph Drawing Challenge, which took place during the conference, required the contestants to find minimum-area straight-line planar drawings of the challenge graphs. The Free-Style category 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 Social Network category was an open category asking contestants to develop and present novel ways of viewing and analyzing social networks. Various networks were suggested including using information from FaceBook, MySpace, and data collection sites such as Technorati, which indexes weblogs.

Although there were a total of 10 submissions, half the amount of previous years, most of these submissions came from participation in the Graph Drawing Challenge. Seven teams participated in the Challenge. There were three submissions in the Social Network category and no submissions in the general Free-Style category. 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.

2 Graph Drawing Challenge

Continuing from the previous year, this year's challenge dealt with minimizing the area of straight-line drawings of planar graphs. At the start of the one-hour

[1] http://www.cs.usyd.edu.au/~visual/gd2007/gd_contest.html

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 395–400, 2007.

on-site competition, the contestants were given seven planar graphs ranging in size from 16 nodes to 324 nodes. As opposed to last year, the graphs were presented with an initial plane embedding, though that particular embedding did not have to be maintained. In response to feedback from the manual team participants of the previous year, the judges felt that too large an amount of time was spent finding valid plane embeddings of the graphs rather than compressing the drawings into a minimum area.

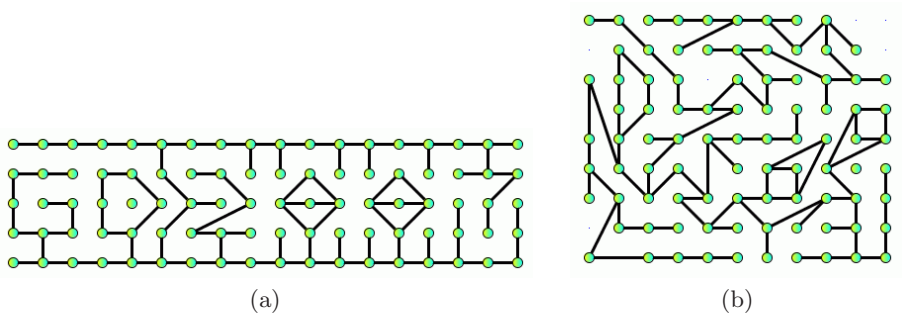
We allowed teams to participate in one of two categories, automated and manual. Manual teams came and solved the problems using ILOG's JViews Utility designed specifically for the Challenge, as a simple graph editing tool and not a specialized area minimization utility. The automated teams were allowed and highly encouraged to use additional software tools to help solve the problems. We also opened up the possibility for remote on-line participation but received no interested parties. This is a strategy we may pursue more aggressively in future challenges. Interestingly, the manual teams performed far better than the automated software. This is in stark contrast to the previous year, which we attribute partially to the fact that the graphs were given a starting, though not optimal, embedding. But, it also highlights the fact that much work still needs to be done to bridge the gap between human and computer performance on this fundamental criterion.

The seven graphs themselves consisted of a varying range of classes. The first graph was a simple graph of 17 nodes with an optimal area of 18, which was found in different representations by three teams. The second graph was a simple maximally planar graph of 16 nodes. The third graph was a bi-connected graph of 64 nodes. The fourth graph was a disconnected graph of 100 nodes and 7 connected components. The fifth graph was a general tree of 192 nodes. The sixth graph was a large graph of 324 nodes and several connected components. The final graph, dubbed GD2007, was a disconnected graph of 90 nodes for which one optimal solution spelled the words: GD2007, see Figure 1. The following table lists the various graphs with their optimal area[2] and the best solutions found by the contestants.

	Graph 1	Graph 2	Graph 3	Graph 4	Graph 5	Graph 6	GD2007
Optimal	18	63	64	100	192	324	90
Best Found	18	63	72	108	204	348	99

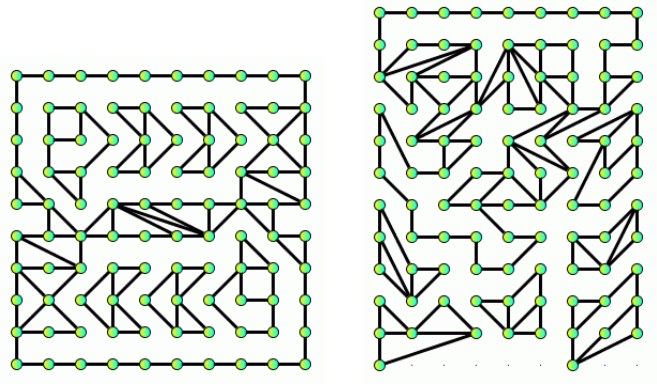
Seven teams participated with one team entering the automated category. The winner was the team of Wolfgang Brunner and Jens Schmidt. Figure 2 shows their winning submission for Graph 4 along with an optimal solution. The judges awarded honorable mentions to Marcus Krug (the sole automated participant), the team consisting of Giuseppe Di Battista, Fabrizio Frati, Michael Kaufmann,

[2] All graphs were constructed in a manner such that the optimal area was known, except for the second graph. As it was constructed in a different manner, the optimal area for the second graph is presumed correct but has not been verified.



(a) (b)

Fig. 1. (a) One optimal solution for the final challenge graph (GD2007). (b) The winning submission by Melanie Badent, Michael Baur, Robert Görke, and Marco Gaertler.



(a) (b)

Fig. 2. (a) One optimal solution for Graph 4. (b) The winning submission from the overall winners Wolfgang Brunner and Jens Schmidt.

Anika Kaufmann, Maurizio Patrignani, and Cagatay Gonsu, the team of Joe Fowler and Michael Schulz, and the team of Melanie Badent, Michael Baur, Robert Görke, and Marco Gaertler.

3 Social Network Category

The Social Network category received 3 submissions, each taking on a different network and having a different approach to its visualization. The judges were impressed with all three submissions.

First prize was awarded to Robert Theron, Rodrigo Santamaria, Juan Garcia, Diego Gomez, and Vadim Paz-Madrid of the VisUsal Group of the University of Salamanca. Their system, Overlapper, was designed to help analyze movies, but their techniques can be extended to other social networks with large collaborations. The tool uses a zone graph representation that is driven by a

force-directed layout algorithm. In their system, nodes represent people involved in a movie, and edges connect two people involved in the same project. However, rather than draw edges explicitly, zones are created. Each movie being a complete subgraph of the people involved is treated as a zone, which is drawn with a semi-transparent hull around it. People involved in more than one movie produce overlapping zones, but the transparency of the zones allows one to visualize the various movies involved. As this can potentially lead to some nodes being covered by zones for which they do not belong, node information is augmented by a pie chart to help discern in which areas the node truly belongs. On a user's demand, nodes are visualized at their position by glyphs identifying the role of the person involved, their corresponding pie chart, and various personal information. Figure 3 shows one snapshot of their animated submission.[3]

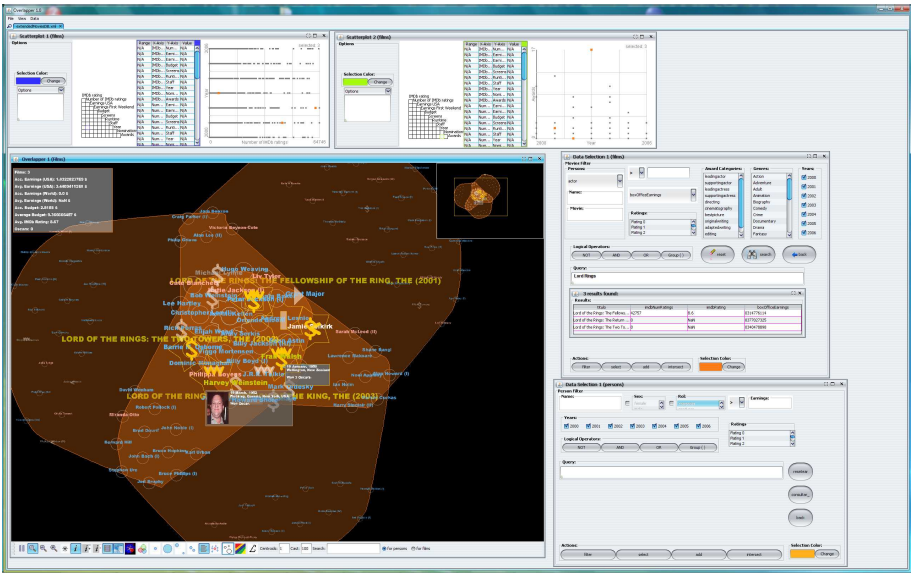


Fig. 3. A snapshot of the Salamanca group's Social Network visualization tool

The second prize submission, awarded to Robert Görke, Thomas Schank, and Dorothea Wagner from the University of Karlsruhe, investigates the co-author network of professors at their university. The nodes consist of all Professors within the Computer Science department along with their co-authors, and the edges are induced by their common publications. The contestants took two approaches to visualize this network. The first approach was a static visualization with measures based on electrical current flow. Between each pair of Professors, they use a uniform potential difference to compute conductivity (connectivity) and the current flowing through each edge. The edges of the graph are drawn

[3] See also http://carpex.usal.es/~visusal/site/

with an intensity based on their accumulated current. The connectivity (conductivity) of each Karlsruhe Professor to their colleagues is visualized by color with more red indicating a stronger connection. For other nodes, only those names with the highest current turnover are shown. Figure 4 shows their resulting static visualization. The authors also consider a dynamic case, visualizing the network for each year from 1999 to 2006 separately. In this case, the emphasis is on preserving the mental map. The details of this approach can be found on the contest's website.[4]

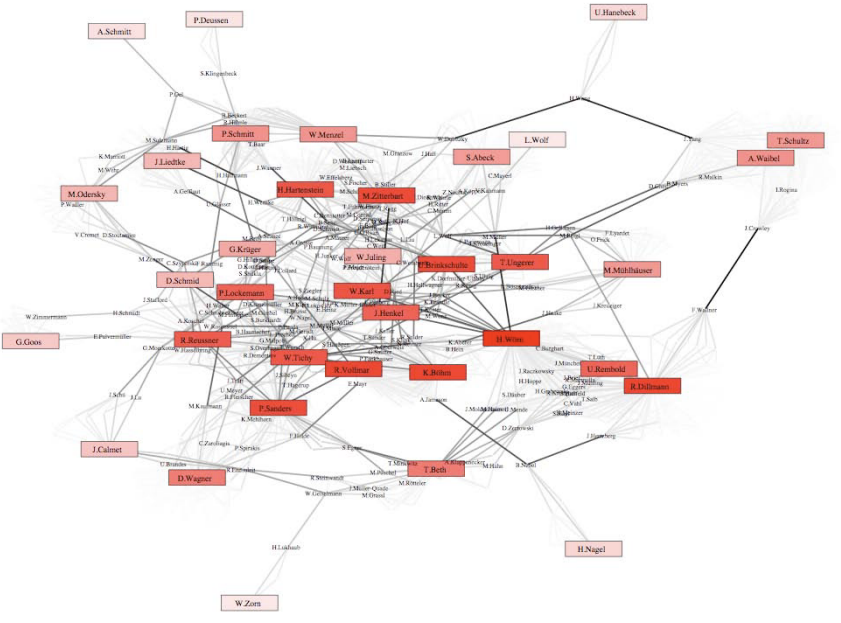


Fig. 4. The static visualization of the Karlsruhe group's co-author network analysis

4 Free-Style Category

Surprisingly, this year we received no submissions for the Free-Style category. However, the judges felt that one of the social network submissions was a worthy Free-Style candidate based on its artistic merit and strong relevance to graph visualization. Therefore, first prize in the Free-Style category was awarded to Felix Heinen for his depiction of the variety and attitudes of members of the Internet community MySpace. The submission consisted of two large poster images. The first poster, Figure 5, shows information gathered from the demographic data of each member's profile, highlighting connections between gender, age, and educational background among the members. The second poster gives the viewer a feel for the geographic distribution of the members. The work was primarily done

[4] See also http://i11www.iti.uni-karlsruhe.de/people/schank/gd07cont/

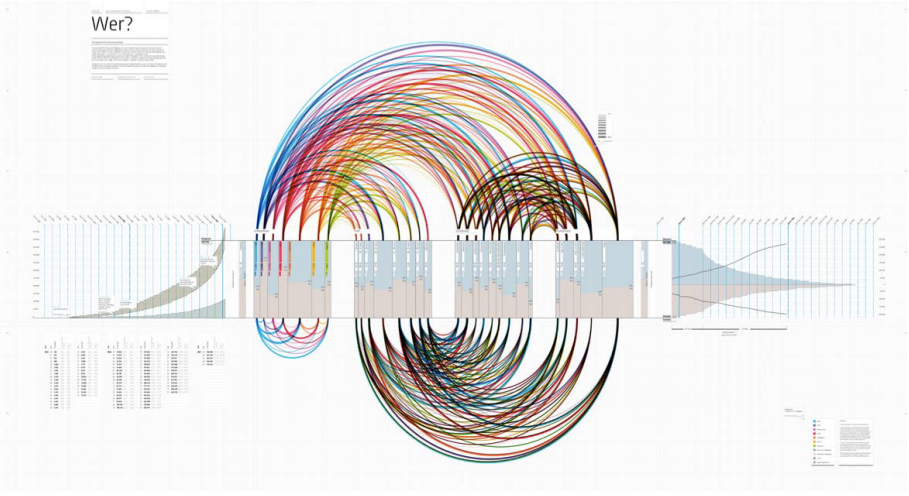


Fig. 5. A visual analysis, created by Felix Heinen, of member profiles of an Internet community

by hand using various Adobe products. For further details visit the contest's website.[5]

Acknowledgments

We wish to thank the contestants and all the sponsors of the symposium, including ILOG, Tom Sawyer, and the HxI Initiative, for their generous support of the Graph Drawing Contest.

[5] See also http://www.felixheinen.de/020.html

Author Index